

Design, Development and Assessment of the  
Java Intelligent Tutoring System

Edward R. Sykes, M.Ed., B.Ed., B.Sc.

Department of Graduate and Undergraduate  
Studies in Education

Submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy

Faculty of Education, Brock University  
St. Catharines, Ontario

© September 2005

**JAMES A GIBSON LIBRARY  
BROCK UNIVERSITY  
ST. CATHARINES ON**

## Abstract

The “Java Intelligent Tutoring System” (JITS) research project focused on designing, constructing, and determining the effectiveness of an Intelligent Tutoring System for beginner Java programming students at the postsecondary level. The participants in this research were students in the School of Applied Computing and Engineering Sciences at Sheridan College. This research involved consistently gathering input from students and instructors using JITS as it developed. The cyclic process involving designing, developing, testing, and refinement was used for the construction of JITS to ensure that it adequately meets the needs of students and instructors. The second objective in this dissertation determined the effectiveness of learning within this environment.

The main findings indicate that JITS is a richly interactive ITS that engages students on Java programming problems. JITS is equipped with a sophisticated personalized feedback mechanism that models and supports each student in his/her learning style. The assessment component involved 2 main quantitative experiments to determine the effectiveness of JITS in terms of student performance. In both experiments it was determined that a statistically significant difference was achieved between the control group and the experimental group (i.e., JITS group). The main effect for Test (i.e., pre- and posttest),  $F(1, 35) = 119.43, p < .001$ , was qualified by a Test by Group interaction,  $F(1, 35) = 4.98, p < .05$ , and a Test by Time interaction,  $F(1, 35) = 43.82, p < .001$ . Similar findings were found for the second experiment; Test by Group interaction revealed  $F(1, 92) = 5.36, p < .025$ . In both experiments the JITS groups outperformed the corresponding control groups at posttest.

## Table of Contents

	Page
Abstract.....	ii
List of Tables .....	v
List of Figures .....	viii
CHAPTER ONE: THE PROBLEM .....	1
Problem Statement.....	2
Rationale .....	4
Definition of Terms.....	7
Assumption and Limitations .....	7
Feasibility.....	8
Outline of Remainder of the Document.....	10
CHAPTER TWO: LITERATURE REVIEW .....	12
The Psychological Framework of ITS .....	12
Student-Teacher Perspectives .....	21
Current State of Development for Intelligent Tutoring Systems .....	24
ACT-R Cognitive Theory for Developing Tutors .....	30
CHAPTER THREE: DESIGN.....	35
Initial Designs for the Java Intelligent Tutoring System .....	35
Motivation for the Design of the Java Error Correction Algorithm .....	37
Java Error Correction Algorithm Design.....	38
Java Intelligent Tutoring System Design and Architecture .....	43
CHAPTER FOUR: IMPLEMENTATION.....	51
Initial Java Error Correction Algorithm (JECA) Implementation .....	51
Initial Java Intelligent Tutoring System Implementation .....	64
Human-Computer Interaction .....	65
Hint Generation.....	69
Initial User Modeling Design.....	79
CHAPTER FIVE: METHODOLOGY AND PROCEDURES.....	80
Subjects .....	80
Statement of Procedures .....	82
Methodology and Procedures: A Summary .....	88
CHAPTER SIX: FINDINGS (ANALYSIS AND EVALUATION).....	92
Summary of JITS Development Research.....	93
Summary of Student Performance Score Analysis.....	96
First Program Development Session: Section #1: JITS Developmental Research.....	100

First Program Development Session: Section #2: JITS Performance Score Analysis.....	127
First Program Development Session: Section #3: Summary and Recommendations for Further JITS Development .....	127
Second Program Development Session: Section #1: JITS Developmental Research.....	140
Second Program Development Session: Section #2: JITS Performance Score Analysis.....	143
Second Program Development Session: Section #3: Summary and Recommendations for Further JITS Development .....	143
Third Program Development Session: Section #1: JITS Developmental Research.....	150
Third Program Development Session: Section #2: JITS Performance Score Analysis.....	170
Third Program Development Session: Section #3: JITS Summary and Recommendations for Further JITS Development .....	182
 CHAPTER SEVEN: SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS	190
Summary .....	190
Conclusions.....	196
Contributions to the Fields of Computer Science and Education.....	197
Implications.....	199
Recommendations.....	201
 References.....	203
 Appendix Brock Ethics Approval.....	208

## List of Tables

Table	Page
1 Student Modeling: Assessment and Instructional Adaptation. Excerpt from ANDES Physics Tutor .....	17
2 Java Reserved Words and Keywords.....	40
3 Extended Java Reserved Words and Keywords.....	41
4 Internal JECA Parse Tree Permutations and Competition for the Selection of the Best Trees .....	63
5 Initial JITS ORACLE Schema Tables .....	70
6 Hint Objects Utilization and Typical Dialogue Between JITS and the Student.....	78
7 Interview Sheet .....	84
8 Performance of Students in JITSC and Control Prior Exposure to JITS and After Exposure to JITS.....	90
9 Sample Database Student Tracking Information Indicating Number of Attempts, Solved (true/false), and Student's Solutions .....	107
10 Sample Database Student Tracking Information Indicating Current Problem Set, Problem_id, Performance Rating, Skill level, Number of Times Connected to JITS, and the Date of Last Connection .....	109
11 Redesigned JITS ORACLE Schema Tables .....	118
12 Redesigned JITS ORACLE Schema Showing the Newly Created Programming Topics and Corresponding Descriptions.....	119
13 Performance of Students in Class JITSC .....	128
14 Performance of Students in Class C.....	129

15	Standard Statistical Measures for C and JITSC .....	130
16	Two-way ANOVA with Repeated Measure: Between-Subjects Effects for C and JITSC .....	131
17	JITS Qualitative Summary Results for JITSC Students .....	134
18	Performance of Students in Class JITSC .....	144
19	Performance of Students in Class C.....	145
20	Standard Statistical Measures for C and JITSC .....	146
21	Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C and JITSC .....	147
22	Redesigned JITS ORACLE Schema Tables to Accommodate Pictures.....	164
23	Performance of Students in JITSC Class Taught by Instructor “A” .....	173
24	Performance of Students in C Class Taught by Instructor “A” .....	174
25	Standard Statistical Measures for C and JITSC Taught by Instructor “A” .....	175
26	Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C and JITSC Taught by Instructor “A” .....	176
27	Performance of Students in JITSC Class Taught by Instructor “B” .....	177
28	Performance of Students in C1 and C2 Classes Taught by Instructor “B” .....	178
29	Standard Statistical Measures for C1, C2, and JITSC Taught by Instructor “B” ..	179
30	Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C1, C2, and JITSC Taught by Instructor “B” .....	177
31	Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C1 and JITSC Taught by Instructor “B” .....	181

32	Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C2 and JITSC Taught by Instructor “B” .....	183
33	JITS Qualitative Summary Results for JITSC Students .....	187

## List of Figures

Figure	Page
1. Architecture of an Intelligent Tutoring System. ....	15
2. First Component of JECA – Scanner Correction Activities. ....	44
3. Second Component of JECA – Parser Correction Activities.....	45
4. Initial design for the JITS User Interface.....	48
5. Model View Controller (MVC) design pattern implemented in JITS. ....	50
6. Keyword object and _keyword data structure. ....	54
7. BestMatch object–used for the refinement process in determining an identifier or a keyword.....	57
8. BestMatch member contains the Transformation string from Edit_Distance algorithm. ....	58
9. Burke-Fisher error correction algorithm with a 4-token queue in the middle of processing a statement production.....	60
10. Burke-Fisher error correction algorithm with a 4-token queue completing the processing of a statement production and commencing a new production. ....	61
11. JITS multithreaded distributed web-based infrastructure. ....	66
12. JITS login screen.....	67
13. Initial JITS User Interface.....	68
14. Hint categories. ....	71
15. A JECA Hint object representing a grammatical error. ....	74
16. Arithmetic sum Java program with grammatical errors and syntax errors.....	75
17. Internally corrected JECA source program for the arithmetic sum problem.....	76



18. Sample performance test for quantitative investigation.....	86
19. Completed version of the JITS User Interface.....	94
20. Showing (a) the two-way Semester by Test interaction due to the smaller gap between pretest and posttest later in the semester, and (b) the two-way Group by test interaction due to the superior performance of the JITS group at posttest...	97
21. Showing the two-way Group by Test interaction for responses indicating superior performance for the JITS group at post-test. ....	99
22. “View Top Hint” results. JITS selects the most significant hint to offer the student. ....	103
23. “View All Hints” results. JITS displays all of the hints relating to all of the problems JITS has encountered with the student’s submission.....	104
24. JITS analysis and response to a submission that is identical to the required output. JITS responds in the same manner as a human tutor would.....	106
25. JITS Authoring Tool User Interface. ....	113
26. “View Solution” presenting solutions for the current problem. ....	115
27. Redesigned JITS User Interface incorporating Programming Topic selection panel.....	117
28. JITS abstract internal object representation showing relationships and dependencies between JECA, AI_Module, student, and other components.....	120
29. Redesigned JITS User Interface depicting the list of Programming topics in a drop-down combo list. ....	121
30. Resigned JITS User Interface depicting the “Previous Problem” and “Next Problem” buttons. ....	122

31.	JITS Tutorial window and main JITS User Interface. The tutorial window is launched from the main JITS User Interface by clicking the “View Tutorial” button as indicated by the arrow.....	125
32.	JITS Tutorial window displaying a sample tutorial from the list of Programming Topics.....	126
33.	JITSC versus C performance comparison using pretest and posttest means as data.....	133
34.	Initial design of the output from the “My Performance” button.....	139
35.	Performance of C and JITSC students using mean grades as data. ....	149
36.	“My Performance” button displays performance information for each student. ...	155
37.	Top right section of JITS’ User Interface displaying the “Help Me” button.....	159
38.	JITS Help screen is used to assist new users to get oriented with this ITS. ....	160
39.	Improved JECA demonstrating filtered output from the compiler and JITS presenting the results in a friendly way for the student to make corrections.....	161
40.	A variation of a compiler error due to a student’s submission. Previous versions of JECA would simply return a hint: “Sorry. No hints available.” The improved JECA is intended to be more helpful and presents compiler errors in a more friendly way.....	162
41.	Revised JITS User Interface accommodating a link to the image for the current problem.....	165
42.	JITS Image Viewer depicting the image for the current problem. ....	166

43. Revised “My Performance” button output showing links to previously attempted problems, font, and colour distinctions between solved and unsolved problems.....	169
44. Classifying control groups and experimental groups based on instructors.....	171
45. JITSC versus C1 and C2 performance comparison using pretest and posttest means as data. ....	184
46. JITSC versus C1 performance comparison using pretest and posttest means as data. ....	185
47. JITSC versus C2 performance comparison using pretest and posttest means as data. ....	186
48. Final version of the JITS User Interface.....	192
49. JITS Exit Screen thanking the participant. ....	194

## CHAPTER ONE: THE PROBLEM

Accessibility to computers and computer resources is increasing in our society at a staggering rate. Not only is computer technology changing more rapidly now than at any other time in history, but the price of computers is continually decreasing inversely proportional to the power they deliver. Over 50% of households in Canada and the United States have computers (Vasilevsky, 2003). Internet connections and capabilities are growing at amazing rates due to the number of people who want to be connected to the world of information (Vasilevsky, 2003). Internet Service Providers (ISP) are rapidly upgrading their infrastructure to support real-time video and audio to their clients. Personal Digital Assistants (PDA) such as cellular telephones and palm-pilots, are Internet ready and becoming commonplace in our society. In spite of the advances in computer technology and accessibility, educators have been relatively slow in seizing technology to enhance student learning. There are significant problems in the context of personalized student instruction in current educational systems that can be remedied through the use of appropriate technologies.

Online teaching tools such as WebCT and Blackboard are becoming extremely popular for distance education and mainstream in-class education. In fact, entire colleges and universities have implemented online teaching tools as the central mechanism for delivering all of their courses (Boyd, 2003). The strength of these tools is their ability to provide the teacher and student with a great deal of versatility within the learning environment. Unfortunately, they do not provide any means by which a student may receive ongoing personalized instruction. Teaching students on a one-on-one basis significantly influences the degree of knowledge and skill retained by the student; Bloom

suggests that one-to-one tutoring is the most effective strategy known, generally yielding two standard deviations better performance than traditional instruction. He suggests further that mastery learning approaches one-to-one instruction in terms of measured learner gains (Bloom, 1984).

This raises the following crisis in the educational community. In order for students to reach their potential, they need individual tutoring. However, due to a plethora of factors such as the limitations of online teaching tools, financial considerations, and sheer logistics, each student cannot be granted access to a personalized human tutor for a consistent duration of time. After all, traditionally there is only one teacher in a classroom of students. So, what can be done to solve this problem? One solution is to design and implement Intelligent Tutoring Systems (ITS). A generally accepted definition for an ITS is a system that employs artificial intelligence methods to assist trainees to improve their problem solving skills by monitoring their reasoning, tracking errors to their source, and, based on the diagnosis, providing advice and assistance to strengthen problem solving skills (Tracey, 2003). ITS allows for more open-ended programs (Tracey, 2003).

### **Problem Statement**

The nature of learning and teaching has not changed significantly since the days of Socrates. Experienced teachers understand the value of Socratic dialogue to lead students to deeper levels of understanding and higher levels of performance. Effective instruction involves establishing the right level of difficulty coupled with engaging and realistic coached-practice simulations (de Koning & Bredeweg, 2001). A significant goal

for developers of Intelligent Tutoring Systems is to embed the Socratic dialogue method within a media-rich environment. Developers of ITS attempt to establish an interactive environment that leads to the suspension of disbelief by students, uses a spiral instructional pattern, and accounts for how humans process information and learn.

Intelligent Tutoring Systems represent advanced forms of cognitive technologies that use computational intelligence. Like human tutors, ITS are useful in reducing the time required by students to acquire knowledge and expert skills. For example, Koedinger (2001) found that in traditional approaches, student tasks take two to three times longer than in ITS environments.

The core of all ITS is an Artificial Intelligence (AI) module that is responsible for many tasks including capturing the student's knowledge state, delivering an appropriate lesson, assessing and evaluating student performance, and providing valuable feedback to the student. Thus, ITS are explicitly designed to assist in resolving the crisis in education regarding personalized student education by providing a means through which students may reach their potential.

The "Java Intelligent Tutoring System" (JITS) research project focused on designing, constructing, and determining the effectiveness of an Intelligent Tutoring System for beginner Java programming students at the postsecondary level. First and foremost, this research involved consistently gathering input from students and instructors using JITS as it developed. The cyclic process involving designing, developing, and testing, and back to redesigning was used for the construction of JITS to ensure that it adequately met the needs of students and instructors. The second objective in this dissertation determined the effectiveness of learning within this environment by

comparing students exposed to JITS with those taught Java in a traditional classroom environment.

### **Rationale**

Currently, Java is one of the most popular programming languages for Internet programming (Chen, 2004). Due to the rapid growth of Java, virtually every university and college in North America now offers a Java course (Martinez, 2002). This study is in tune with the current trend and demands of the science and technology and education sectors of our society. A third justification for this study is the fact that there is no Intelligent Tutoring System for the Java programming language at this time. A fourth reason supporting this study is based on the international support and interest the scientific community has had during the development of JITS (Sykes & Franek, 2003, 2004a, 2004c).

Additional justification for this study is drawn from a provincial level. The double-cohort students of Ontario secondary schools may cause significant problems when they enter postsecondary institutions. For instance, Lakehead University expected a 35% increase of students entering first-year programs in 2003 compared to the previous year (Gilbert, 2003). Despite the accommodations that were performed, class sizes rose significantly. Intelligent Tutoring Systems may be an answer to provide individualized attention for students who would otherwise be disadvantaged. In light of these and other recent statistics, the Intelligent Tutoring Systems like the one proposed in this dissertation may prove to be extremely beneficial to the current set of crisis in education.

### *Context*

Recent advancements in multimedia, high-speed Internet connections, and computer-mediated communication and communities for individual and distance learning all have the potential for revolutionary improvements in education. Numerous countries are striving to support this revolution and are learning how best to adopt these new technologies (Alevan & Ashley, 1997).

An even larger revolution is approaching. This revolution will be based on the widespread use of Artificial Intelligence in educational technology (Koedinger, 2001). The two fundamental bodies of research that support this paradigm shift come from cognitive science and Artificial Intelligence (Alevan & Ashley, 1997). One reason for the increase in Artificial Intelligence in education is due to the fact that powerful computers are becoming very affordable.

Another reason for this revolution has been scientific progress which comes from two sources. First, progress in Artificial Intelligence has led to a deeper understanding of how to represent knowledge, how to reason, and how to describe procedural knowledge (i.e., “how to” knowledge). Second, research in cognitive science has led to a deeper understanding of how people think, solve problems, and learn. There is a powerful synergy here. Cognitive scientists often use Artificial Intelligence techniques to build simulation models of cognitive processes, dependencies, and represent behaviours (Conati & Van Lehn, 1999). Artificial Intelligence scientists use results from cognitive science to guide their explorations and to design software with more human-like characteristics. When applied to education, this synergy leads to combinations of software and activities that can help more students achieve better learning.



Intelligent Tutoring Systems are being used in numerous areas of education including mathematics, physics, cognitive skill development, and workplace simulations (Koedinger, 2001). For instance, an ITS called the Pump Algebra Tutor (PAT), was developed by Kenneth Koedinger of Carnegie Mellon University which had extremely positive results (Anderson, Corbett, Koedinger, & Pelletier, 1995). PAT is currently used in several hundred high schools, middle schools, and colleges around the United States and in Europe. PAT is designed to help students to learn to model real-life problem situations using algebraic representations including tables, graphs, equations, and words. Modern mathematics is less about computing single answers and more about creating models that can provide answers to multiple questions. Thus, PAT's curriculum emphasizes the use of activities that draw on students' common sense and prior informal strategies to help them acquire and make sense of formal mathematical strategies and representations. The goal is to help all students be successful in algebra and see its relevance in both academics and the workplace. Both teachers and students have been enthusiastic about PAT's use as part of the Algebra I curriculum. Field studies have shown dramatic student achievement gains relative to control classes: 15-25% better on standardized tests of basic skills and 50-100% on assessments of problem solving and representation use.

In summary, the purpose of this study is to design and construct an ITS for entry-level college and university students learning Java and to conduct a performance score quantitative study to ascertain the effectiveness of the system. The proposed research will be pioneering and will impact the fields of cognitive science, Artificial Intelligence and education. However, to design and construct an Intelligent Tutoring System can take

several years of work for a team of people such as, Educational Psychologists, Computer Programmers, Knowledge Engineers, and AI experts. Since this is a dissertation, I am responsible for all components of this study. As a result, this project proposes to construct an ITS designed to tutor students in a very specific fashion using a subset of the Java programming language. In this way, a prototype may be constructed which will be sufficient to prove the concept is sound and provide a means by which a quantitative study may be performed. There is little doubt that e-learning and related technologies are becoming important tools in education, and it is imperative that research be conducted in the area of intelligent tutoring systems. By doing so, educators will be enabling students to gain personalized instruction which will help them achieve better learning.

### **Definition of Terms**

#### ***General Definitions***

AI	Artificial Intelligence
ITS	Intelligent Tutoring System
JITS	Java Intelligent Tutoring System

### **Assumption and Limitations**

Instructors selected for this study were drawn from the School of Applied Computing and Engineering Sciences (ACES), Sheridan College, Institute of Technology and Advanced Learning, Ontario, Canada. Students were drawn from the population of computer studies courses in their first year. An overriding assumption is that usual classroom characteristics were in play in the JITS.

## **Feasibility**

### ***The Timeliness Factor***

Currently, Java is one of the most popular programming languages for development of multiplatform applications and enterprise level business solutions (Chen, 2004). Many colleges and universities in North America are offering Java programming courses at various levels in various certificate, diploma, and degree programs (Martinez, 2002). These two factors, among others, signify that the Java Intelligent Tutoring System research project is of merit and an important study at this time.

### ***The Cost Factor***

The cost associated with this study was quite minimal. As expected, there were incidental fees such as travelling between Sheridan's two main campuses and printing questionnaires, forms, and letters throughout the duration of the study.

### ***The Time Factor***

To design and construct an Intelligent Tutoring System takes years of work for a team consisting of researchers and programmers. Clearly, such resources were not available. The researcher was responsible for all aspects of the research project. The initial design and development for the Java Intelligent Tutoring System took nearly 1,000 hours of research and programming work. Over the last 2 years, the researcher worked on JITS approximately 20 to 30 hours per week. During the third year, a quantitative investigation study involving control groups and experimental groups was conducted. The purpose of this component of the study was to determine performance score differences between control and experimental groups.

### ***The Accessibility of Data***

Within the School of Applied Computing and Engineering Sciences at the Sheridan Institute of Technology and Advanced Learning there were many students who were interested in trying out JITS. There were many introductory Java classes offered every semester (i.e., May, September, and January). As a result, there were many opportunities to conduct studies and elicit feedback to improve the Java Intelligent Tutoring System. The motivation for selecting ACES at the Sheridan Institute of Technology and Advanced Learning was due to circumstance and convenience. I have been a professor in the School of ACES for 10 years, am very comfortable with the faculty and staff, and have developed over 20 courses within the curriculum for the school. An additional benefit is that ACES has just recently changed all of the first-year programming courses to Java. These factors contribute to the fact that data in various forms were easily accessible for this study.

### ***Inconvenience to Participants***

The degree of inconvenience for instructors and students was very low. Students in the experimental group were exposed to JITS during classroom break periods or at the end of a classroom period. The amount of time allocated for the students to try out JITS was between 30 minutes and one hour per week. When some students were participating in the research study, nonparticipants were encouraged to work on course-related exercises and other activities as set out by the professor of the class as per usual class activities. At Sheridan, classes are typically 3 hours in duration in a mobile environment where each student has their own laptop computer. A typical instructional period consists of a sequence of a 15-30 minute lecture followed by a 15-30 minute hands-on activity.

This sequence is repeated until the end of class. However, there are usually several 10- to 15-minute breaks during a 3-hour class. I conducted the JITS experiments to coincide with the break periods and at the end of class periods.

The time spent on the research project by participants did not detract from course objectives due to the nature of exposure to the instruments in this research project. One of the goals of this research was to determine if the research project can be of benefit to students to more quickly and effectively reach course objectives. The overall time spent by students in the research study did not detract participants in their academic objectives.

### **Outline of Remainder of the Document**

Chapter Two contains a literature review analysis of Intelligent Tutoring Systems. Within this chapter, a discussion of the psychological framework of ITS is presented. This chapter gives background information to the reader as to what has previously been accomplished in the field of Intelligent Tutoring Systems. In this chapter, the popularly accepted ACT-R theory of cognition for ITS development is discussed.

Chapter Three contains the design framework for JITS. Within this chapter, a discussion involving the initial design for JITS is presented. Design considerations such as user interface considerations, online persistency, number of concurrent users, student tracking, student modeling, and other considerations are reviewed and presented in this chapter.

Chapter Four presents the implementation details of the Java Intelligent Tutoring System. Following from the previous chapter, which specified the design approach, this chapter focuses on details involving the Java Error Correction Algorithm and the initial

construction of JITS. The initial designs for JITS' Human-Computer interaction, Hint Generation, and User Modeling issues are also included in the chapter.

Chapter Five describes the methodology and procedures by which this research was conducted. Included in this chapter is the design and development methodology for JITS. This chapter also includes the manner in which data were gathered regarding student performance in JITS.

Chapter Six discusses the findings of the research in the form of three case studies coupled with the statistical analysis of the data. The case studies summarize the students' comments about JITS and how these suggestions were used in the redesign and redevelopment of JITS. Tables in this chapter contain statistical information including performance data, descriptive statistics, two-way ANOVAs, and charts.

The last chapter of this document summarizes the results and discusses the implications of the analysis. The recommendation section of this chapter offers suggestions for future work on JITS and in the area of online e-learning tutors. Following the chapters are the references and appendix.

## **CHAPTER TWO: LITERATURE REVIEW**

This chapter presents the main foundational areas which support the study by demarking the boundaries of investigation, reviewing existing Intelligent Tutoring Systems, and providing motivation for the proposed direction of investigation. The areas reviewed include:

1. The psychological framework of ITS;
2. Student-Teacher perspectives;
3. Current State of Development for Intelligent Tutoring Systems; and
4. ACT-R Cognitive Theory for Developing Tutors.

### **The Psychological Framework of ITS**

The psychological underpinnings within Intelligent Tutoring Systems are not new. The principles that form the foundation of ITS are thousands of years old. Throughout the years, researchers have probed into the complexity of the human mind to understand and explain its functionality. The psychological framework supporting ITS is based on this corpus of knowledge about human cognition and learning. In this section, the psychological framework and pedagogical strategies involved with Intelligent Tutoring Systems are presented. The infrastructure supporting the development of Intelligent Tutoring System relies on cognitive activities.

#### ***Cognitive Activities***

Cognitive activities include such things as perceiving, thinking, learning, remembering, and problem solving. In order for Intelligent Tutoring Systems to be effective for the student, the learning process must support two distinct purposes. First, the ITS must exhibit cognitive-type activities so that the learner will interact and respond

appropriately to the tutoring process. Second, the ITS must be able to identify these types of activities in the learner and plan the next set of steps in the student's learning process.

The details behind cognitive activities include a range of information processing activities that take sensory data and engage in processes to create meaningful information for some specific purpose. For example, some of these activities involve the following:

- **Attention:** acquiring information by paying attention to what is happening and perceiving the relevant;
- **Encoding:** transforming sensory data into mental propositions and constructs for processing;
- **Associating:** relating new mental propositions and constructs to existing knowledge;
- **Storing:** keeping information and knowledge for future use. This involves short-term, intermediary, and long-term memory stores;
- **Retrieving:** timely access to stored information and knowledge;
- **Communicating:** producing results and desired outcomes; sharing knowledge with others.

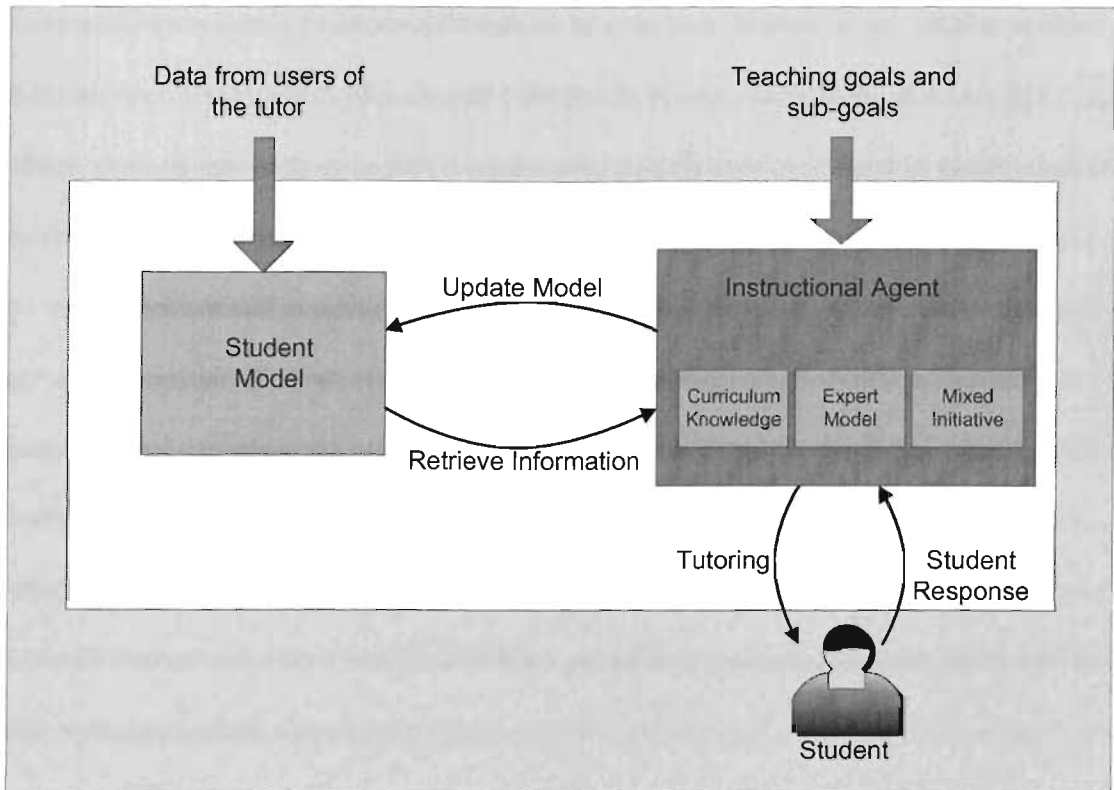
The psychological and pedagogical framework of the proposed Java Intelligent Tutoring System is based on the developments of cognitive science and artificial intelligence. The framework is based on the foundational cognitive and information processing activities. ITS are typically comprised of four modules including: Expert, Student, Instructional Agent, and Interface. These modules collectively represent the following modeling activities required to effectively tutor a student in a specific domain. The modeling activities are: Curriculum Knowledge Modeling, Student Modeling,



Expert Modeling, Mixed-Initiative, and Self-Learning. Figure 1 depicts an architecture of an Intelligent Tutoring System.

The following is an example illustrating the abstract operations of an Intelligent Tutoring System. Suppose the goal is to tutor the student to spell the word “cat.” The Student Model represents the characteristics of similar students and, in particular, specific characteristics of the student being tutored. The Instructional Agent is one part of the Intelligent Tutoring System that takes this goal and, aided with the Curriculum Knowledge, commences the tutoring process. While there are many different types of Intelligent Tutoring Systems, rich interaction is a key component to all current tutoring systems. The interaction between the Student and the ITS is depicted in Figure 1 by the arrows “Tutoring” and “Student Response.” The action of “Tutoring” is based on information retrieved from the Student Model by the Instructional Agent, the Expert Model, the goal, and the Curriculum Knowledge module. When the student responds to the system, the Instructional Agent updates the Student Model appropriately to ensure that the ITS always knows the current state of the learner. This cycle continues until the goal has been achieved.

***Instructional agent.*** As illustrated in the example, the Instructional Agent is designed to guide the student towards a solution during the tutoring process. The Instructional Agent is adaptable in that it makes changes to the instructional strategies based on changes in state in the student model. Depending on the domain, various strategies may be used, such as explanation, guided discovery learning, coaching, and probing. Human teachers and tutors do this regularly within the delivery of a lesson (Bloom, 1984).



*Figure 1.* Architecture of an Intelligent Tutoring System.

How and why human teachers alter their teaching style is an interesting question and is currently being researched (Koedinger, 2001; B. P. Woolf, Beck, Eliot, & Stern, 2001). Research in cognitive science will lead to more effective modules that will better adapt to the needs of the student (B. P. Woolf et al., 2001).

***Curriculum Knowledge Modeling.*** The Curriculum Knowledge (CK) module generates appropriate instructional material based on the context of the student's performance. Components in this module include problems, solutions, exercises, hints, and help. The CK module presents the student with the appropriate instructional information to stimulate student learning while minimizing discourse and frustration. The ITS needs to be able to distinguish from predefined responses and genuine responses that would most help the student.

***Student Modeling (SM).*** This component models each student in the system. There are two main components of the Student Model. The first is a collective student model representing common characteristics that all students in the system share. The second component is the unique model for the student being tutored. The Student Model is used in the assessment of the student's knowledge and in the instructional process by the Instructional Agent. In some Intelligent Tutoring Systems, the assessment need not be formalized as a quiz or test but may include informal approaches such as coaxing or subtle probing questions. For example, Table 1 describes a typical dialogue between a student (S) and the ANDES Physics Tutor (T).

Table 1

*Student Modeling: Assessment and Instructional Adaptation. Excerpt from ANDES*

*Physics Tutor*

---

**Problem:** An elevator slows to stop from an initial downward velocity of 10.0 m/s in 2 seconds. A passenger in the elevator is holding a 3 kg package by a vertical string.



**T:** *What is the definition of acceleration?*

S: I don't know.

**T:** *Let's look at it a different way. If I am driving north and you want to stop me which direction do you have to push?*

S: north

**T:** *But if the acceleration went the same direction as the velocity, then the elevator would be speeding up. Try again, if I am driving north and you want to stop me which direction do you have to push?*

S: south

**T:** *Right! What does that tell you about my acceleration?*

---

***Expert Modeling (EM).*** This component of an Intelligent Tutoring System models an expert's knowledge for a particular domain. The Expert Model is used by the Instructional Agent in determining appropriate feedback for the student. Current systems vary in the type of knowledge they teach. This is because each ITS is extremely domain specific; that is, each ITS provides a means by which each student may construct their knowledge in a very specific domain. Some Intelligent Tutoring Systems teach formal logic and formal knowledge while others teach overall processes in the context of the domain. For example, MathTutor and Algebra Tutor, both associated with mathematics, teach formal logic and knowledge (Anderson & Reiser, 1985; Beal, Beck, Woolf, & Rae-Ramirez, 1998). On the other hand, Intelligent Tutoring Systems such as MeTutor focus on processes involved in firefighting which are more process-oriented in terms of the curriculum tutored (Rowe & Galvin, 1998).

Constructing the expert model requires specifying the relative difficulty of the topics, knowledge of the strategies that can be used by the tutor, a large amount of analogies, examples, and error diagnosis abilities to effectively tutor the student (B. P. Woolf et al., 2001).

***Mixed-Initiative (MI).*** This component of the ITS is based on the development of human-computer interaction (B. P. Woolf et al., 2001). The dynamics involved are based on human-human interaction that have been studied and analyzed from a communication perspective (Graesser & Person, 1994; Graesser, Person, & Harter, 2001). As a result, the mixed-initiative module determines the most effective way for the ITS to communicate with the student while ensuring rich interactivity and productivity. Natural language dialogue is used more frequently in the design of recent Intelligent

Tutoring Systems (Graesser et al., 2001). Another responsibility of the Mixed-Initiative module is its ability to recognize mistakes. *Error diagnosis* will help the ITS to diagnose mistakes, plausible misconceptions, and recognize missing information. The goal is to establish a balance between the student controlling the conversation and the ITS.

***Self-Learning.*** The ITS needs to have the capacity to monitor, evaluate, and improve its own teaching performance as a function of experience. In other words, based on the information in the Student Model and student responses, the Instructional Agent reflects upon what was successful, what was not, and refines the process for the current student and future students (see Figure 1).

In their most sophisticated form, Intelligent Tutoring Systems might reason based on knowledge about how students solve problems and make inferences in the domain. The theoretical focus has shifted from exclusive diagnosis and remediation to recognizing and supporting students in managing their own cognitive processes. In other words, ITS may provide a way for students to develop their metacognitive abilities. A tangential, yet related element to Intelligent Tutoring Systems is a philosophical debate regarding Strong versus Weak Artificial Intelligence. Strong AI is defined as a hypothetical form of AI that can truly reason and solve problems; Strong AI is said to be sapient, or self-aware, but may or may not exhibit human-like thought processes. The term Strong AI was originally proposed by John Searle and was applied to digital computers and other information processing machines (Searle, 1980). Searle defined Strong AI as, "according to Strong AI, the computer is not merely a tool in the study of the mind; rather, the appropriately programmed computer really is a mind" (Searle, 1980, p. 417).

Weak AI, on the other hand, refers to the use of software to study the behavioristic and pragmatic view of intelligence. In Weak AI, there is not the claim for software actually being intelligent, but simply being a tool that can be used to assess hypotheses regarding the nature of intelligence. Formality is a necessity if a mechanistic approach is required. Weak AI also covers probabilistic systems where results are not deterministic (Searle, 1980). If results are based not on "real" cognitive mechanisms with a deep enough complexity as to accommodate intentionality, though the input/output interface layer would appear to the user as hiding an intelligent "mind". This is clearly an illusion; like watching an animation which, to the observer, appears perfectly "real". Since the perspective impression is perfect from the observers point of view, the observer ignores common sense and accepts as fact that the images are real (Searle, 1980). The definition of "Weak AI" accepts this reality and is opposed to Strong AI.

In summary, the framework of an ITS is supported by progress in the fields of philosophy, cognitive science, and Artificial Intelligence—specifically Weak AI. Intelligent Tutoring Systems are constructed based on interdisciplinary studies, a wealth of human tutoring knowledge, and have been met with notable success (Graesser et al., 2001; Koedinger, 2001). At this time however, Intelligent Tutoring Systems do not display the appearance of "true" intelligence from an external perspective. In other words, ITS do not inherently imply that there is a "real" mind hidden in the machine with the same cognitive capabilities as those, or equivalent to, human ones. As a result, ITS, are currently being designed and constructed as Weak AI systems, as the one developed in this dissertation. However, the future looks promising for more sophisticated ITS due to advancements in cognitive science, human-neural modelling, and the rapid progress of

computer technology. The philosopher René Descartes identified thinking as proof of one's existence. "I think, therefore I am." From such simple intuitions, we can generalize in order to say "thinking things exist" (Cottingham, Stoothoff, Murdoch, & Kenny, 1991). Perhaps one day Intelligent Tutoring Systems will be considered thus.

### **Student-Teacher Perspectives**

#### ***The Student's Perspective***

Cognitive studies of instruction have shown that students need to remain active and motivated to succeed (Fletcher, 1995; Regian, 1997; Seidel & Perez, 1994). Students must want to learn and to be involved, active, and challenged to understand and manipulate the material presented. The experience must be authentic and relevant to the learner's world (Woolf, 1992; Woolf & Hall, 1995). Simple presentations of text, images, or multimedia usually result in systems that encourage passiveness. As a result, the learning can be less effective than intended. Interactive exercises are required that involve students in the material (B. P. Woolf et al., 2001). Not surprising, human tutors actively engage students. This often leads to significant improvements in the learner's achievement (Bloom, 1984). In fact, one-on-one tutoring by human tutors can increase the average student's performance by 2.0 standard deviations. That is a two-letter grade improvement (Bloom). Intelligent Tutoring Systems emulate the behaviour of human tutors and have been met with significant success (Koedinger, 2001; B. P. Woolf et al., 2001).

Another benefit ITS have is the speed of knowledge acquisition. For example, an ITS was designed for the LISP programming language with the goal to improve students' programming abilities at the postsecondary level (Anderson, Conrad, & Corbett, 1989).



In this study, comparisons were conducted between a control group in which students solved the same programming problems without the aid of the ITS and an experimental group which used the ITS. Students in the experimental group completed the problems in *one third of the time* with better posttest performance than students in the control group (Anderson et al., 1989).

Intelligent Tutoring Systems have many benefits for education. By adapting curriculum to each student, learning could be customized for normal students, gifted students, learning-disabled students, and minority groups<sup>1</sup>. This allows self-directed or asynchronous learning to occur. Many current ITS are being constructed to support e-learning techniques and pedagogies. Thus, the goal for Intelligent Tutoring System designs is to provide worldwide access to an electronic personalized tutor for students in domain-specific areas. These concepts were important in the design of the Java Intelligent Tutoring System.

### ***The Teacher's Perspective***

Traditional classrooms in Ontario typically range from 20 to 32 students. Postsecondary institutions in this province may have hundreds of students per class. Clearly, a teacher will struggle to be effective under these circumstances. One-on-one tutoring is by far the preferred and most successful instructional model (Bloom, 1984; Koedinger, 2001). The use of Intelligent Tutoring Systems would be extremely beneficial for teachers trying to help each student in a classroom. The individual teacher needs to be empowered to deliver the curriculum as effectively as possible for each individual student. Schofield, Evans-Rhodes, and Huber (1990) found that teachers take

---

<sup>1</sup> The terms “normal,” “gifted,” “learning-disabled,” and “minority group” students used in this context are defined by the Ministry of Education and Training. Please see: <http://mettowas21.edu.gov.on.ca/eng/general/postsec/taskbg1.html> for more information about these terms.

a greater facilitator role in student-centred learning environments. This in turn would result in higher achievement for most students (Bloom, 1984; Schofield, Evans-Rhodes, & Huber, 1990). Since Intelligent Tutoring Systems effectively engage students, teachers are more able to provide a personalized assistance tool to students who most need it (Wertheimer, 1990).

Intelligent Tutoring Systems provide many benefits for teachers. First, ITS help teachers with day-to-day responsibilities such as tracking student performance on quizzes, tests, assignments, exercises, and readings. This in turn assists teachers by providing more time for them to focus on curriculum development and delivery. In this way, teachers are in a better position to more effectively meet the needs of their students. Consequently, the students' interests can be more easily incorporated and formalized by the teacher through the use of authoring tools available for Intelligent Tutoring Systems. This is essentially the same as specialized learning plans quite common in the school setting.

Another benefit of ITS is that they minimize the disruptions in class, since students are focused and engaged in working through the course material. An extreme example provided by Anderson et al., 1995,

a student in a school in another state had the LISP tutor. The student, frustrated by restrictive access to the LISP tutor, deliberately induced a 2-day suspension by swearing at a teacher. He used those 2 days to dial into the school computer from his home and complete the lesson material on the LISP tutor. (p. 22)

Intelligent Tutoring System research is currently focused on three main areas: cognitive modeling, communication modeling, and implementation (B. P. Woolf et al.,

2001). A clear direction ITS researchers have is to fully include teachers and students as equal partners in the research (Heffernan & Koedinger, 2001). There is little doubt that this combined effort will improve the quality of Intelligent Tutoring Systems of the future.

### **Current State of Development for Intelligent Tutoring Systems**

This section presents a review of the current research in Intelligent Tutoring Systems. The review was important as it guided the design and development of the Java Intelligent Tutoring System. The framework for the construction of JITS was based on the widely accepted ACT-R theory of skill acquisition which was developed by a group of computer and cognitive scientists at University of Pittsburgh and Carnegie-Mellon University (Anderson, 1998; Anderson et al., 1995 2). This theory identifies a set of cognitive principles for the development of tutors described below (Anderson, Boyle, Corbett, & Lewis, 1990; Anderson et al., 1995).

Intelligent Tutoring Systems have undergone significant changes over the years and can be classified into three main categories. The first generation of ITS were basic Computer Aided Instruction (CAI) systems. They presented text or graphics and depending on the student's response, different pages would be shown. Model-tracing ITS were second generation tutors that allow the tutor to follow the student's actions as they work through a problem. The current level of research and development for Intelligent Tutoring Systems is the third generation. These tutors engage in dialog with the student to allow students to construct their own knowledge of the domain. For third generation tutors, interaction with the student is the key element in the design since it is essential to keep the student's attention on task and as close as possible to the solution path. This has

the benefit of minimizing student frustration and reducing off-task activities that do not yield in increased learning (Anderson & Pelletier, 1991).

Heffernan and Koedinger, 2001, state:

We think that if you want to build a good [third generation] ITS for a domain you need to:

- i) study what makes that domain difficult, including discovering any hidden skills, as well as determining what types of errors students make;
  - ii) construct a theory of how students solve these problems (We instantiated that theory in a cognitive model); and
  - iii) observe experienced human tutors to find out what pedagogical and content knowledge they have and then build a tutor model that, with the help of the theory of domain skills, can capture and reproduce some of that knowledge.
- (p. 24)

The following section elaborates on Heffernan and Koedinger's (2001) recommendations for the construction of an Intelligent Tutoring System. With reference to item i), there is a need to discover what makes the domain difficult. Understanding the conceptual challenges beginners have when learning to program was an important issue for the researcher to formalize before designing the Java Intelligent Tutoring System. In order for a student to program effectively, s/he needs to (a) understand the syntax and semantics of the language; (b) have the ability to work abstractly on a programming problem; and (c) know how to test and validate the proposed solution.

Item (a) involves the knowledge and skill set to enter code into the computer in a manner that the syntax of the language is satisfied and the semantics of the expressions in

the solution match the grammar of the language. Item (b) involves the knowledge and skills to develop algorithms to solve specific problems. Sometimes small algorithms are used collectively to solve large problems. This ability to decompose a large problem into discrete modules is an additional ability a programmer must have to solve programming problems. In this context, one can envision this characteristic of a programmer by the programmer's ability to effectively "zoom-in" and "zoom-out" on details in the program at various levels. Item (c) involves the ability to objectively critique one's work from a functional perspective. Programmers need to be able to test their programs to determine the validity of their solution. When unexpected results are produced as output of a program (known as a logic error), the programmer needs to be able to "zoom-in" on the related area of the program and ascertain how to correct the problem. Typically, this involves a misunderstanding on the student's part in the manner in which his/her program is executing. In order to solve logic errors, it usually involves a change to the algorithmic design of the program in order to generate the desired output.

Additionally, there is a distinction between overt skills and hidden skills. In the domain of programming, overt skills may include the ability to type and use the mouse effectively to enter code and manipulate text in the computer. An even more difficult issue is the proper identification of hidden skills which needs to take place in the development of an ITS. Hidden skills in the area of beginner Java programmers may include the ability to recognize a syntax error, understand why the error is generated, and know how to remedy the error. This is a hidden skill because there are potentially an infinite number of different scenarios that give rise to a specific syntax error, and the

student needs to have the ability to associate from previous knowledge and experiences with the current unique situation.

The following is another example of a hidden skill. Consider a programming problem to calculate the factorial of a nonnegative integer number provided by the user of the program. There are a number of elements that need to be addressed in the solution for this problem. The solution may be divided into three distinct abstract sections:

(S1) prompt the user for a nonnegative integer number;

(S2) use recursion or a loop to calculate the factorial of this user-entered number;

(S3) display the factorial to the user.

Within each section in this modelled solution exist many hidden skills. The student needs to have the knowledge and skills to program a prompt for the user to enter a number. There are many different datatypes in Java, so the student needs also to understand these differences in order to select the appropriate type for this particular programming problem. For instance, a student may select a “double,” “short,” or “Integer” instead of the desired “int” type. The selection of “double,” “short,” or “Integer” may lead to difficulties in future steps in the solution being constructed. Another hidden skill in (S1) is the proper validation that the user-entered number is an integer and is not negative. The student needs to be able to recognize this requirement and have the knowledge and skills to be able to translate this requirement into appropriate programming code. In this case, it would be to use an “IF—ELSE” construct, such as:

```
IF user_entered_number < 0
THEN issue_invalid_input_to_user
ELSE proceed_with_computation_of_factorial.
```

Within section (S2) the student needs to have knowledge of the various loop constructs in the Java programming language and have the skills to implement a solution using them. The hidden skills involved in this step are numerous. They involve preconditions, identifier declaration and initialization, temporary variables, the specification of the test condition, loop body syntax and semantics (i.e., the test condition must change in the body of the loop, syntactical knowledge including when to use opening and closing braces, etc.), and other such low-level details of implementation in Java.

Last, in section (S3), the student needs to have the knowledge of how to display information within a Java program, the types of problems that may be encountered in outputting information, and the knowledge of how to correct these errors.

The last section of Heffernan and Koedinger's (2001) statement in i) states: "determining what types of errors students make." In programming, this involves recognizing that the student may perform many different variations of the two classes of errors: syntax errors and logic errors. In the factorial example provided above, there are many different errors that the beginner programmer may encounter. In each of the sections (S1, S2, and S3), numerous syntax and logic errors may be generated and, in each unique case, the student needs to be able to understand why the error is occurring and be able to correct it.

As presented above in the example of the factorial problem, programming is a difficult domain to model because of the vast and detailed knowledge and skills required for a student to have, the ability to "zoom-in" and "zoom-out" at various levels of detail in a program, the ability to deal with and manage various levels of abstraction, and the

ability to “troubleshoot” program errors and to remedy them in a reasonable amount of time.

Furthermore, items i) and iii) are well covered by the researcher’s experience in programming for over 20 years and being a Professor of Computer Science for 10 years at the Sheridan Institute of Technology and Advanced Learning. The researcher was the co-ordinator of the Computer Science Technology program for several years and has extensive firsthand knowledge of the curriculum implemented. The researcher has learned and taught over 10 different programming languages at the postsecondary level. The researcher understands Java very well and knows the fundamental skills required by students to solve programming problems. Regarding Heffernan and Koedinger’s (2001) statement i), the researcher has gained years of experience in the types of errors students make. The researcher has determined that when a student encounters a syntax problem and cannot overcome it, it is largely due to a conceptual gap (i.e., lack of knowledge) or misunderstandings in the grammar of the particular programming language. As a comparison to the English language, a student who does not use the period character properly may lack the understanding that the *period* character is a special symbol that denotes the end of a sentence. This is a direct analogy to the Java programming language except instead of a period, it is the *semicolon* character and instead of *sentences*, the collection of grammatically correct symbols is called a *statement*. To remedy these types of cognitive issues (i.e., conceptual gaps or misunderstandings), usually all that is required is a minilesson on a few similar examples to the problem at hand and the student fills in the conceptual gap or corrects the knowledge associations. However, if the student has encountered a logic error and cannot overcome it, then this typically reveals a



difficulty in the student's ability of abstraction. The student does not have the ability to carefully step through his/her solution with a critical eye while examining the developed algorithm. The researcher has discovered that the main reason, particularly for beginner programmers, is their inability to remain patient and focused while analyzing their solution slowly and carefully. This very significant ability is often the one that separates a student from being successful in programming or not. This is a reflection of the student's inability to "zoom-in." In other words, the student fails to be able to "zoom-in" enough to gather all the related attributes of the algorithm and troubleshoot the precise problem.

Referring to item ii), Heffernan and Koedinger (2001) state: "construct a theory of how students solve these problems (We instantiated that theory in a cognitive model)." The theory used in this dissertation is based on the ACT-R theory, which can be summarized by four principles. The ACT-R theory is described in the following section.

### **ACT-R Cognitive Theory for Developing Tutors**

The first principle derived from ACT-R (Architecture of Cognitive Tutors) is that it is essential to define the target cognitive model as a set of production rules (Anderson, 1998; Anderson & Pelletier, 1991). Production rules are a set of IF-THEN-ELSE constructs which outline discrete knowledge components which collectively represent the steps required for a student to reach a solution for a problem. A typical ITS may have several hundred production rules to effectively cover the domain and the various states a student may be in within a realm of feasibility and predictability. Heffernan and Koedinger (2001) reinforce this principle: "Without this [principle] one does not have a well-defined educational goal" (Koedinger, 2001). In other words, in the context of

ACT-R, tutoring is assuring students (a) construct the production rules, (b) practice the production rules, and (c) remediate the errors in the production rules. Additionally, it is a goal of the Intelligent Tutoring System to guide the student towards a solution. However, it is not mandatory that the solution be achieved by the student. In other words, the ITS recognizes that the student may become frustrated and not wish to continue. The ITS records the current state of the student's progress, noting the degree of learning that has taken place even though a solution may not have been achieved.

The second principle concerns how these production rules are to be communicated to the student (Anderson, 1998). According to ACT-R theory, one cannot directly tell students the underlying rules (Anderson, 1998; Graesser et al., 2001). The goal for ITS is to provide a vehicle by which students construct knowledge for themselves as opposed to having the information told to them (B. P. Woolf et al., 2001). ITS need to communicate the production rules to students by providing them with examples that illustrate the rules. As a result, the most effective way for students to construct knowledge is to acquire these rules as a by-product of problem-solving. This form of experiential learning is an effective way for students to construct knowledge and increase their cognitive abilities (O'Reilly & Munakata, 2000).

The third principle of ACT-R theory is that one wants to maximize the rate at which students have opportunities to form and practice these production rules (Anderson, 1998). Based on other research by ITS researchers, it was shown that what predicts students' final achievement is how much practice they have had of these rules and not how that practice occurs (Anderson et al., 1995; Anderson & Pelletier, 1991). Associated with the concept that "practice makes perfect" is the corollary to minimize floundering

which is incorporated into many leading-edge Intelligent Tutoring Systems. The basic idea is to reduce student frustration during the problem-solving session and select problems that offer practice on those production rules where students most need practice (Anderson et al., 1995). A production rule in the ACT-R theory is a statement of a particular contingency that controls behaviour in the Intelligent Tutoring System. The following are two examples of production rules:

```
IF the goal is to classify a person
   AND he is unmarried
THEN classify him as a bachelor
```

```
IF the goal is to add two digits d1 and d2 in a column
   AND  $d1 + d2 = d3$ 
THEN write d3 in the column
```

A production rule is a condition-action pair. The condition specifies a pattern of input symbols that must be present for the production rule to execute. The action section specifies the action that is to take place. A typical ITS may have hundreds of production rules to encapsulate the knowledge of the domain of instruction. For the design of the Java Intelligent Tutoring System, the set of production rules is represented by the grammar of the Java language coupled with custom production rules augmented to the language. Chapters 3, 4, and 5 discuss the JITS production rules in greater depth.

The fourth principle of ACT-R cognitive theory for tutoring deals with how to treat errors in student problem solving (Anderson, 1998). Anderson et al. base this principle on an earlier work in 1990, which states, “people learn best when they generate the answer for themselves rather than are told” (Anderson et al., 1990). However, the consequence of letting people generate their own knowledge is that errors are inevitable. Fortunately, there are four considerations outlined in ACT-R theory that deal with error remediation (Anderson, 1998). First, many errors do not reflect misunderstandings or

lack of knowledge; rather the errors are simply unintentional slips. The second consideration is that people learn best when they construct the knowledge themselves. This is analogous to hands-on training as opposed to lecture-based teaching. The third consideration is that a lot of time can be wasted when the student is floundering while trying to solve a problem. This state is called an *error state* and is not beneficial for learning. The fourth consideration is that when students have problems with their knowledge, it is more effective to provide another opportunity to learn the correct production. Since the student does not need a deep appreciation of their error, it is not effective for the ITS to expound on it (Heffernan & Koedinger, 2001).

The ACT-R Theory for the development of tutors has led to a standard framework for the design and construction of Intelligent Tutoring Systems. The goal of this framework is to ensure that Intelligent Tutoring Systems will provide rich learning environments for students that will support their cognitive development in the specific domain of study in as effective means as possible. Many researchers in the area of ITS support the following steps to design and construct an Intelligent Tutoring System.

1. construct the interface;
2. define the production rules;
3. create the declarative instruction; and
4. set up the Instructional Agent to manage the curriculum and engage the student through rich-interaction (Anderson, 1998; Anderson et al., 1990; Heffernan & Koedinger, 2001).

During the design of the Java Intelligent Tutoring System, these steps were performed but not in the order presented. Due to the complexities involved with the way

in which JITS is designed, a massive amount of effort was spent on step 2, that is, defining the production rules. This is because JITS was designed to recognize any small Java program and offer “intelligent” feedback when there is no authored solution available. In other words, unlike other Intelligent Tutoring Systems, there is no predetermined solution for each problem. As a result, the focus of this step in the project was on compiler error correction strategies which used extensive production rules in the form of Backus-Naur Form (BNF) for the grammar of the Java language. Once this was completed, the next step the researcher pursued was step 3.

Once the production rules were in place and validated, the declarative instruction became the focus of the researcher. Declarative instruction was designed and implemented by a series of tutorial web pages with ease of navigation and quick reference of paramount design consideration. For more information regarding this step in the design of JITS please refer to Chapters 3, 4, and 5.

In step 4 of the ACT-R theory recommendation, a prototype for the Instructional Agent including a hint generation module was designed and developed. Small curriculum modules were also created to test the interaction between user and the ITS prototype. After extensive testing of the prototype system, the last step for design and development was the construction of the User Interface (step 1). Please see Chapters 3, 4, and 5 for a detailed description of the design and implementation of the Java Intelligent Tutoring System User Interface.

## **CHAPTER THREE: DESIGN**

This chapter presents the design framework for the Java Intelligent Tutoring System. The design incorporates leading-edge techniques that are used by current developers of Intelligent Tutoring Systems. Design considerations such as website persistency, concurrency, student tracking, student modeling and other considerations are reviewed and presented in this chapter.

The design for the construction of JITS is based on the popularly accepted ACT-R theory of skill acquisition. The ACT-R theory was developed by a group of cognitive scientists at University of Pittsburgh, and Carnegie-Mellon University (Anderson, 1998; Anderson et al., 1995; Anderson & Pelletier, 1991). This theory identifies a set of cognitive principles for the development of tutors. The proposed Java Intelligent Tutoring System is based on this accepted theory of tutor construction. Four main sections are included in this chapter: “Initial Designs of the Java Intelligent Tutoring System”; “Motivation for the design of the Java Error Correction Algorithm”; “Java Error Correction Algorithm Design”; and “Java Intelligent Tutoring System Design and Architecture”.

### **Initial Designs for the Java Intelligent Tutoring System**

During the design of JITS, the four ACT-R cognitive theory principles for developing tutors were carefully considered. According to the ACT-R theory, the following steps are recommended:

1. construct the interface;
2. define the production rules;
3. create the declarative instruction;

4. set up the pedagogical agent to knowledge trace, manage the curriculum, and engage the student through rich-interaction (Anderson, 1998; Anderson et al., 1990; Heffernan & Koedinger, 2001)

The first step is to define the target cognitive model as a set of production rules (Anderson, 1998; Anderson & Pelletier, 1991). These production rules, although not written in traditional IF—THEN—ELSE constructs are embedded in the Java language specification grammar which is used by the Java Error Correction Algorithm (discussed in the following section). This grammar definition contains hundreds of grammatical production rules that support the Java programming language. As a result, JITS upholds the first principle of this cognitive theory.

The second principle of the ACT-R theory states that these productions need to be communicated to the student. A subcomponent of this principle is that one cannot tell students the underlying rules (Anderson, 1998; Graesser et al., 2001). JITS is designed with rich-interaction throughout the entire tutoring session. Rules are never directly told to students. Rather, JITS communicates with the student in the form of well-defined hints that are customized to each student in the system. JITS is designed to provide various opportunities for students to engage in problem-solving activities for the beginner programmer, which is an effective way for students to construct their skills and cognitive abilities.

The third principle of ACT-R theory is the old adage “practice makes perfect.” JITS maximizes the rate at which students have opportunities to form and practice the production rules. JITS does this by engaging the student continually in problem-solving questions. As a result, JITS is designed to support this principle.

The last principle of ACT-R cognitive theory describes how the tutor should deal with errors in student problem solving. JITS is designed with a Java Error Correction Algorithm which intelligently determines the intent of the student even when s/he makes mistakes. JITS quickly recognizes students in an *error state* and immediately encourages the student to make appropriate changes. This has the benefit of minimizing students getting “stuck” and at the same time minimizes their frustration.

The initial designs for JITS include the Java Error Correction Algorithm, the User Interface, and web-based infrastructure. These components are described in the following sections.

### **Motivation for the Design of the Java Error Correction Algorithm**

JITS is designed to provide extensive hands-on practice for students learning Java in the form of attempting to solve programming problems. All entry-level programming students make syntax mistakes and logic errors. Thus, a module that sophisticatedly determines the intent of the student and can identify various types of errors that students make is a necessary component for an ITS for the Java programming language.

While text correction is commonplace in word processors, mobile phones, etc., it is not commonplace in the area of compiling a computer program. When a person writes a program in any language, it must precisely follow the syntax and grammar rules of that language. Any mistake, even so minute as forgetting a “;” will cause the program to fail compilation. This dissertation research proposes an intriguing new use in teaching programming by autocorrecting typical mistakes that beginner programming students make. From a pedagogic/didactic perspective, support for the beginner programmer when these types of errors occur can be very helpful. Thus an error correction algorithm



would be very helpful for students. Reviews from the learning and teaching sciences yields this to be true (O'Reilly & Munakata, 2000). As a result, the Java Error Correction Algorithm fits in this chosen theory. Furthermore, based on the principles of the ACT-R cognitive theory for developing tutors, the Java Error Correction Algorithm also coincides with this philosophy.

### **Java Error Correction Algorithm Design**

This section describes the design of the Java Error Correction Algorithm (JECA). The design arose from research involving decision trees, expert systems, and compiler tools. It became clear after preliminary research that JavaCC provided the best features for the development of an error correction algorithm. JECA is designed to consider three distinct cases:

CASE 1: student enters perfect code and it compiles and runs;

CASE 2: student enters code that needs modification but with JECA changes will compile and run; and

CASE 3: student enters code that needs modification but will not compile regardless of all corrections employed by JECA; however, suggestions are presented to the student to bring the code to a closer state for compilation.

The Java Intelligent Tutoring System's intelligence is accomplished by this embedded logic module (i.e., the Java Error Correction Algorithm). This module performs a number of operations behind the scenes. It implements a sophisticated scanner and parser that autocorrects the student's code when appropriate as well as generates a number of parse trees that have small permutations. This module then attempts to compile the best trees to ascertain the most likely path the student

“intended” to follow. With this knowledge, JITS can efficiently and effectively tutor the student. The goals JECA are to:

1. analyze the student’s code submission;
2. intelligently recognize the “intent” of the student;
3. “auto-correct” where appropriate (e.g., converting “While” into the keyword “while,” “forn” into “for,” etc.);
4. learn individual student’s misconceptions, and categorize the types of errors s/he makes;
5. produce a “modified code” that will compile (or bring the code closer to a state of successful compilation); and
6. prompt the student programmer for information when necessary via well-defined hint support structures.

JECA, combined with a well-defined student modeling mechanism and dynamic hint generation capabilities, enables JITS to significantly improve the performance of beginner Java programmers. Over the last 2 years, JECA took over 1,500 hours of the researcher’s work.

The algorithm used by JECA is presented below.

1. Create a copy of the student’s submission (i.e., “modified\_source”).
2. The scanner examines the student’s code and attempts to extract a token. Let  $S$  be the stream of characters to be validated as a token.
3. A validation process ensues in which comparisons are done using the reserved words and keywords of Java (Table 2), extended keywords (Table 3), and previously declared identifiers.

Table 2

*Java Reserved Words and Keywords*


---

abstract	else	interface	super
boolean	extends	long	switch
break	false <sup>a</sup>	native	synchronized
byte	final	new	this
case	finally	null <sup>a</sup>	throw
catch	float	package	throws
char	for	private	transient
class	goto <sup>b</sup>	protected	true <sup>a</sup>
const <sup>b</sup>	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp <sup>c</sup>	

**Note.** <sup>a</sup> true, false, and null are reserved words. <sup>b</sup> indicates a keyword that is not currently used. <sup>c</sup> indicates a keyword that was added for Java 2.

---

Table 3

*Extended Java Reserved Words and Keywords*

---

Boolean
Character
Number
Byte
Double
Float
Integer
Long
Short
String
StringBuffer

**Note.** This list is a subset of the objects defined in `java.lang.*`

---

4. For a given identifier, if the scanner discovers, within a certain threshold, that  $S$  can undergo transformations to convert  $S$  into a valid token (i.e., a reserved word or keyword, an extended keyword, or as a previously defined identifier), then it will do so. However, if the scanner determines that  $S$  is sufficiently different from all of the items previously compared to, then it will be left unchanged (i.e., it will remain as a new identifier).
5. Update the modified\_source code to reflect these changes and the newly constructed token is submitted to the parser.
6. Repeat 1 through 4 until all input from the student's source code has been processed and the parser has completed the construction of the parse tree representing the modified\_source code.
7. Try to parse and compile the modified\_source code. If the compilation succeeds, then relay the modifications performed to the student in order for them to correct their code and stop processing.
8. If the previous step fails, then extract information regarding why it failed and set up a competition of permutated parse trees containing insertions, deletions, and replacements at the problem area.
9. Run these permutated trees through the parser. The goal of this stage is to determine if the specific problem where the parse failed has been corrected.
10. Select the "best tree(s)" and compile these. The "best tree" is defined as the tree that allowed the parser to successfully consume the largest number of tokens compared to the other trees in the competition.

11. If one or more of these trees successfully compiles, then present this information to the user, indicating the changes made to the student's source code.
12. If none of the trees successfully compile then present the information to the student regarding the selection of the best tree.
13. Let the student respond/make corrections to the source code.
14. Repeat the process from 1 to 13.

The algorithm employed by JECA is presented in flowchart form in Figure 2 and Figure 3.

### **Java Intelligent Tutoring System Design and Architecture**

The design of the Java Intelligent Tutoring System heavily relied on JECA to provide the necessary information in order to offer suitable feedback to the student programmer. However, there were a number of factors that were considered in the design of JITS beyond what JECA offered. The two main perspectives that were considered in the design of JITS were both the student's and the instructor's perspectives. In order for an ITS to be successful in today's e-learning society, JITS was designed with the following qualities.

#### ***Student Perspective***

The following qualities were deemed important in the design to satisfy students and were part of the desired list of criteria in the design of JITS:

1. provide an easily understood, student-friendly user interface that provides all the necessary features for effective ITS tutoring;

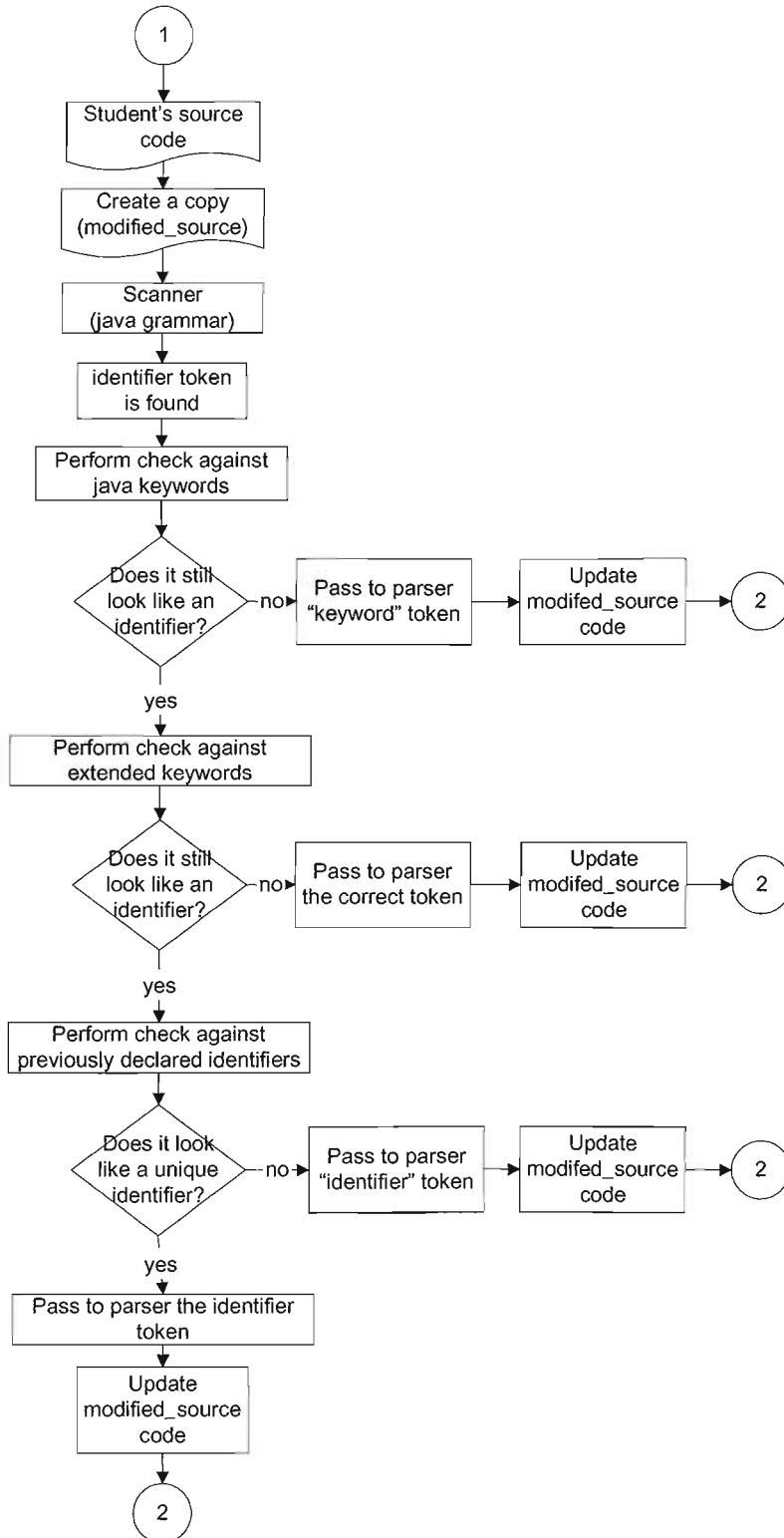


Figure 2. First Component of JECA – Scanner Correction Activities.

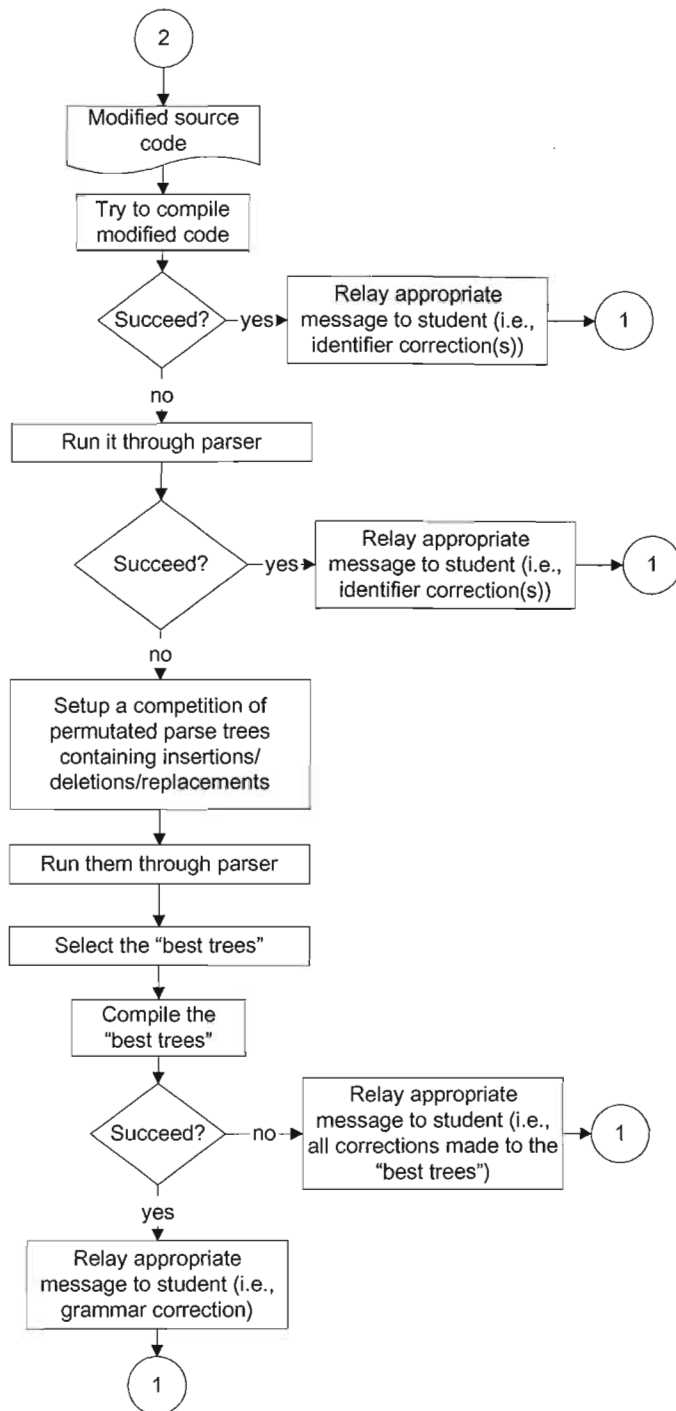


Figure 3. Second Component of JECA – Parser Correction Activities.



2. provide access via an ordinary browser;
3. will not need a high-speed internet connection (i.e., dial-up connection will work fine; thus, students in remote locations have full access to this resource);
4. process student's code submission and respond quickly to the student;
5. support many students concurrently working with the ITS;
6. engage the student by communicating in a clear and concise personalized fashion (e.g., unique hints and error messages for each student);
7. track student performance in a database (e.g., ORACLE); and
8. model the user as s/he works through a problem.

### *Instructor Perspective*

The design of JITS also considered the instructor perspective. The following factors were important in meeting the needs of teachers using this ITS.

1. requires the author of the problem to provide minimal information (e.g., problem statement, program requirements, and required output);
2. the author of the problem does *\*not\** specify any solutions (this is based on the premise that for a given programming problem there may in fact be numerous solutions);
3. JITS must be able to recognize a very large number of possible solutions for a particular programming problem;
4. student performance information should be easily accessible;
5. an instructor-friendly, web-based user interface to author problems (i.e., Authoring Tool).

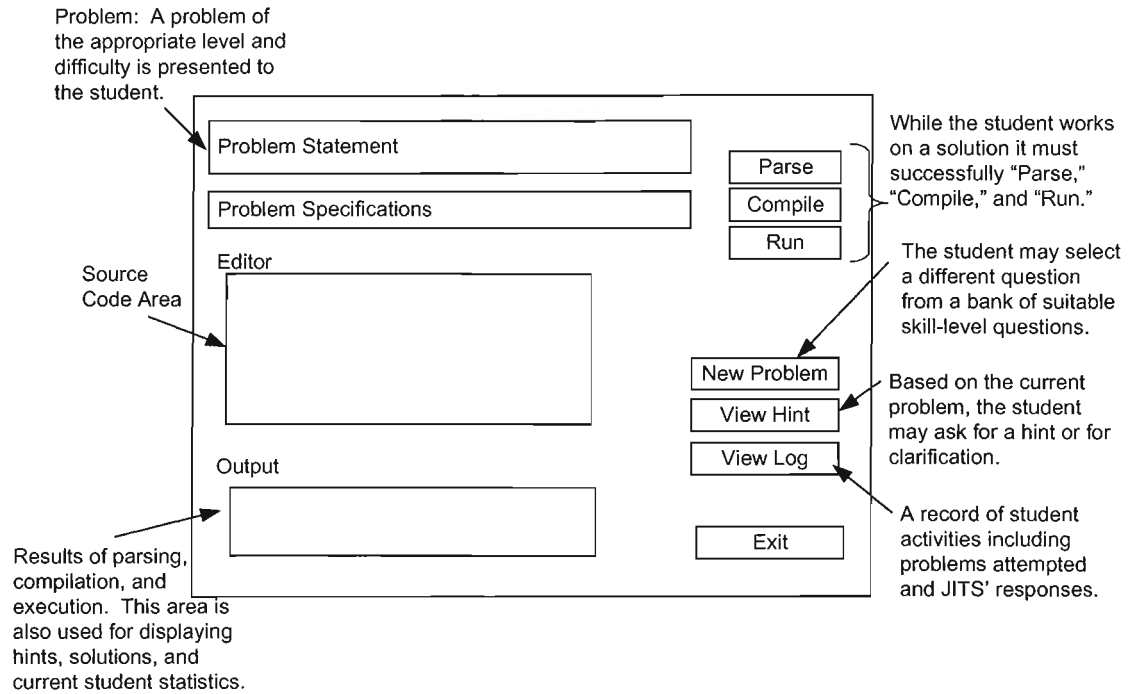
This section includes the initial JITS User Interface and a description of the initial web-based infrastructure architecture.

### ***Initial JITS User Interface***

The initial design of the JITS user interface was representative of fundamental features of a professional programming Integrated Development Environment (IDE). Students are presented with a problem, the problem specification, the skeleton code, the code editor, and a number of buttons with which to interact with the tutor. For example, once the student is ready to submit the code to JITS, the “Submit” button may be pressed. The user interface was designed so that students could view the hints by pressing “View Hint” button, see solutions by pressing the “View Solution” button. Hints were initially designed as “canned responses” to specific states the students would be in based on their code submission. At any time, the student would be able to see a record of their activities by pressing the “View Log” button. This would simply show the student the history of what had transpired in the current session. Over the last two years, the initial interface and supporting infrastructure for JITS has taken approximately 500 hours of the researcher’s work. Figure 4 depicts the initial design of the user interface for JITS.

### ***Infrastructure Design***

The infrastructure design for JITS draws from the area of leading-edge techniques and technologies for multithreaded distributed concurrent e-learning application designs. The Model-View Controller (MVC) design pattern was used to ensure that concurrency and robustness would be provided by JITS. The MVC contains three main tiers: the client’s browser, the middle-tier, and the database-tier.



*Figure 4.* Initial design for the JITS User Interface.

By design, there were no restrictions placed on the browser. In other words, JITS was designed to work with any browser, and no custom installed client software of any sort was required. The middle-tier is a server running a TomCat web server, currently equipped with 4GB RAM and 2 Pentium-IV processors. The database-tier is a separate server running ORACLE. The initial JITS database schema was designed to support the core functionality of JITS consisting of 3 tables: student, problems, and student\_problems. The student table contains information regarding each student in the system such as student name, password, current problem, etc. The problems table contains details regarding programming problems used by JITS such as problem description, specifications, templates, etc. The student\_problems table is an intersection relation representing details regarding each student's attempt at a problem.

The Model-View-Controller design pattern was a core component to the design of JITS. Figure 5 depicts the MVC design pattern. First the student makes a request (via HTTP in the browser). The Controller module receives the request and performs operations that include instantiating JavaBeans. These beans are used to model the student as s/he works with JITS. The collection of these beans represent the modeling of each student in JITS. During specific operations, beans may need to retrieve information from the JITS database schema (e.g., to select a new problem or retrieve solutions to a problem, etc.). These data are stored in the ORACLE JITS database schema represented in the figure as the Enterprise Information System (EIS). The information is gathered up and processed by the bean, which then forwards it to the View component (i.e., the Java ServerPage [JSP]), which then formats it appropriately for the student in the JITS user interface and returns it to the student's browser.

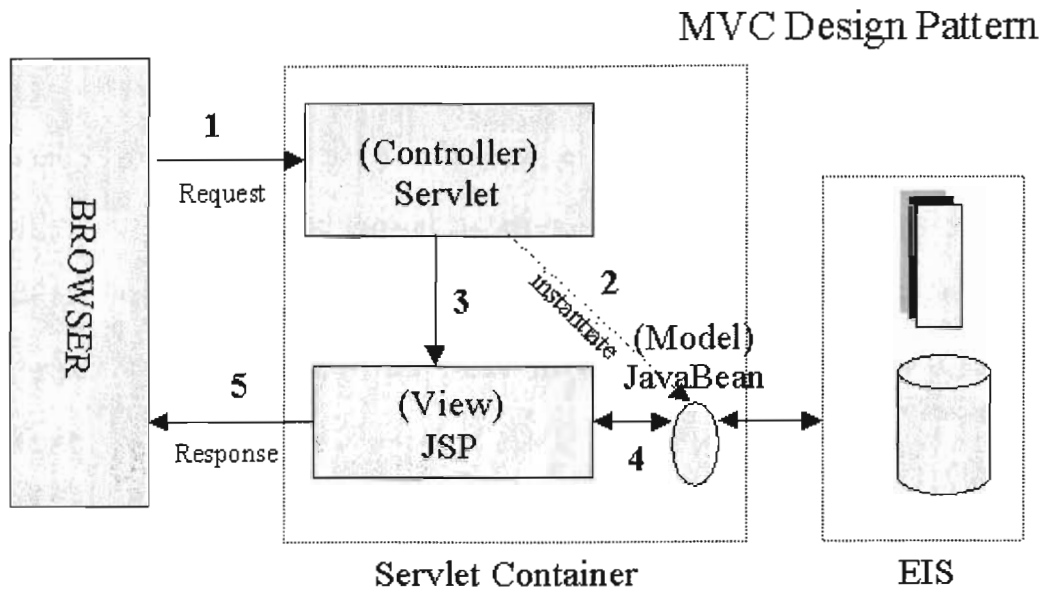


Figure 5. Model View Controller (MVC) design pattern implemented in JITS.

## CHAPTER FOUR: IMPLEMENTATION

This chapter presents the implementation details of the first version of the Java Intelligent Tutoring System. Following the previous chapter, which specified the initial design of JITS, this chapter focuses on details involving the Java Error Correction Algorithm and the initial construction of JITS. The initial designs for JITS' Human-Computer interaction, Hint Generation, and User Modeling issues are also included in the chapter.

### **Initial Java Error Correction Algorithm (JECA) Implementation**

The core module of the Java Intelligent Tutoring System is the Java Error Correction Algorithm (JECA). The first component of JECA involves scrutinizing the identifiers that the scanner has tokenized by comparing them to keywords, reserved words, extended keywords, and to currently validated identifiers. The second component has the parser perform a rigorous deep level error recovery technique implemented by a variation on the Burke-Fisher Error Recovery algorithm (Burke & Fisher, 1987). This algorithm is explained in greater depth in the following sections.

#### ***First Component of JECA: Error Recovery in the Scanner (Lexical Analyzer)***

It is sometimes desirable to change what the scanner has interpreted to a single Java keyword. The reserved words and keywords in the Java programming language are presented in Table 2. As an example, suppose the beginner programmer submitted the following code:

```
public class Test {
    public static void main() {
        Int sum = 0;
        For (iint i=0; i<=10; i++)
            sum = sum + i;
        System.out.println("Sum is:" + sum);
    }
}
```

There are 3 distinct syntax errors. The “Int sum = 0;” statement, the “For,” and the “iint.” It is desirable to present the appropriate information to the student programmer in a way that is both supportive and direct. In this example, the student mistakes the “Int” and “For” for the keywords “int” and “for” respectively. A typical compiler will produce the following:

```
Test.java:5: ')' expected
    For (iint i=0; i <=10;    i++ )
           ^
Test.java:5: not a statement
    For (iint i=0; i <=10;    i++ )
           ^
Test.java:5: ';' expected
    For (iint i=0; i <=10;    i++ )
                               ^
3 errors
```

The proposed error recovery algorithm, JECA, attempts to understand the “intent” behind the student’s program and, by prompting the student, and behind-the-scenes modifies the submitted program as follows:

```
public class Test {
    public static void main(String args []){
        int sum = 0;
        for (int i=0; i <=10;    i++ )
            sum = sum + i;
        System.out.println("Sum is: " + sum);
    }
}
```

generating the anticipated result:  
**Sum is:55**

The student will receive prompts for each “assumption” the JECA intent recognition module is performing. For example, on encountering the “Int” in line 3, a message such as “I found an ‘Int’. Would you like to replace it with ‘int’? (y/n).” In this fashion, the student of the system is fully aware of all changes that are taking place on the submitted code. In other words, all changes are made explicitly known to the user.

This philosophy is different from other compiler designs that make changes to the source program without notifying the user (Fischer & LeBlanc, 1991). For example, an analogy is found in the C programming language. Given a simple program like:

```
main() {  
    return 0;  
}
```

may be interpreted by a compiler as:

```
int main() {  
    return 0;  
}
```

The compiler implicitly puts in the default type 'int' during compilation. Such implicit changes can be misleading to the user (Fischer & LeBlanc, 1991). JECA, on the other hand, does not do any implicit changes to the code. All code changes are overt. A supporting mechanism used to do this is depicted in Figure 6.

A Keyword object houses all attributes and functionality associated with a keyword in the language. It contains the name of the keyword (i.e., `String _name`), the symbol table ID for the keyword (i.e., `int _id`), dynamically learned variations on the keyword (i.e., `String _variation []`), the number of times these corresponding variations have occurred (i.e., `int _count []`), and the total number of variations learned at this time (i.e., `int _variation_count`). The Keyword object contains useful information that can be used for statistical analysis and capturing a representative model of the student of the system. By keeping track of the types of errors the student makes and the number of times these types of errors occur, the system is in a good state to offer meaningful feedback to assist the student to program better.



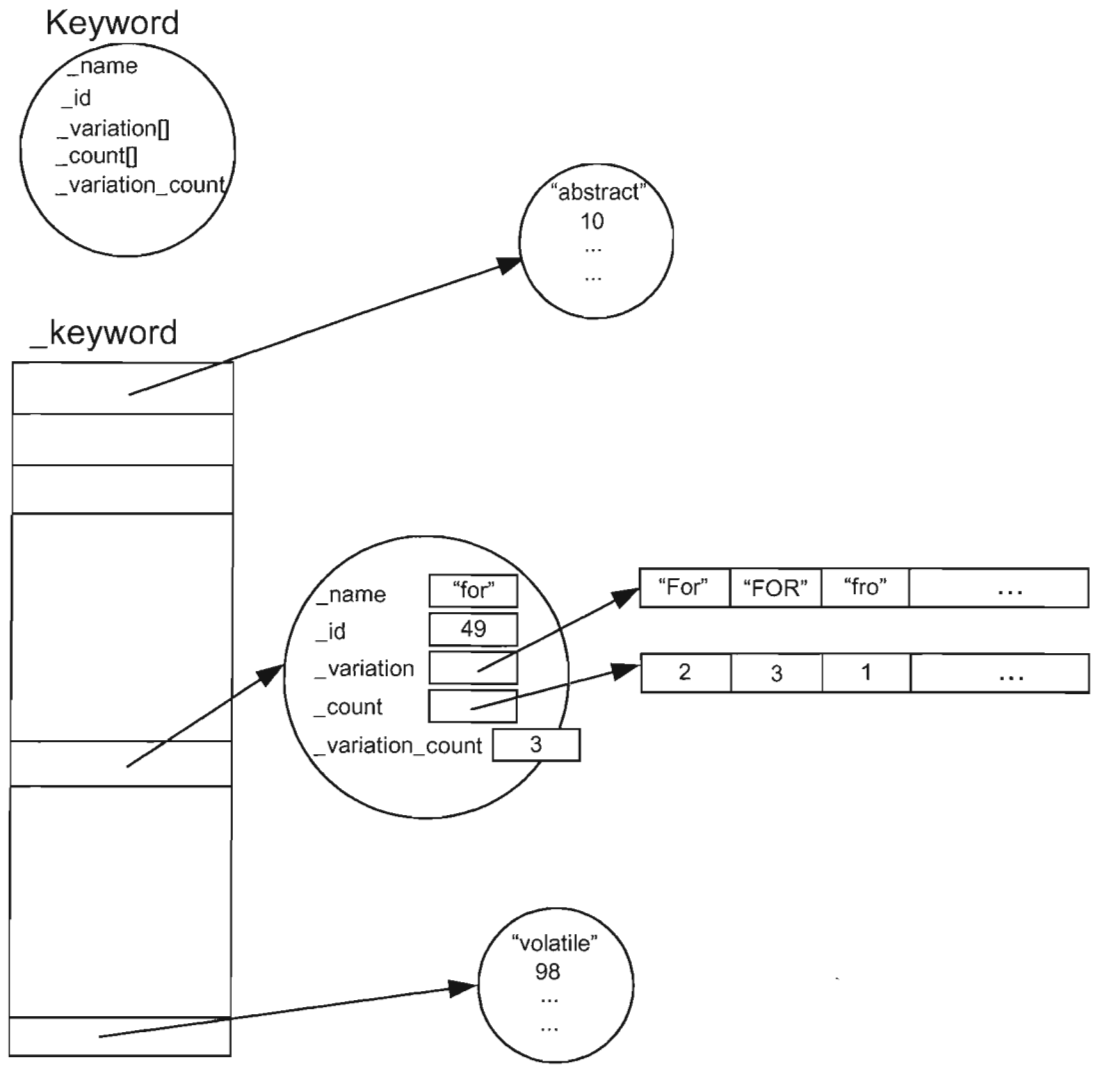


Figure 6. Keyword object and `_keyword` data structure.

Similar data structures are implemented for `Extended_Keywords`, and `Identifiers` in order to record information regarding these types of data. This information is gathered during the lexical analysis phase by JECA.

Given a lexeme that has currently been classified as an identifier token, the objective is to analyze this lexeme and determine if it should remain as an identifier or be classified as a different type of token. The algorithm includes a reference to the `Edit_Distance` object that has a method to determine the edit distance between two strings. For example, given the strings, “while” and “wiles,” the edit distance is 2 (i.e., a count of 1 for the missing character “h” and 1 for the additional character “s”). The algorithm for this identifier-classification process is presented below:

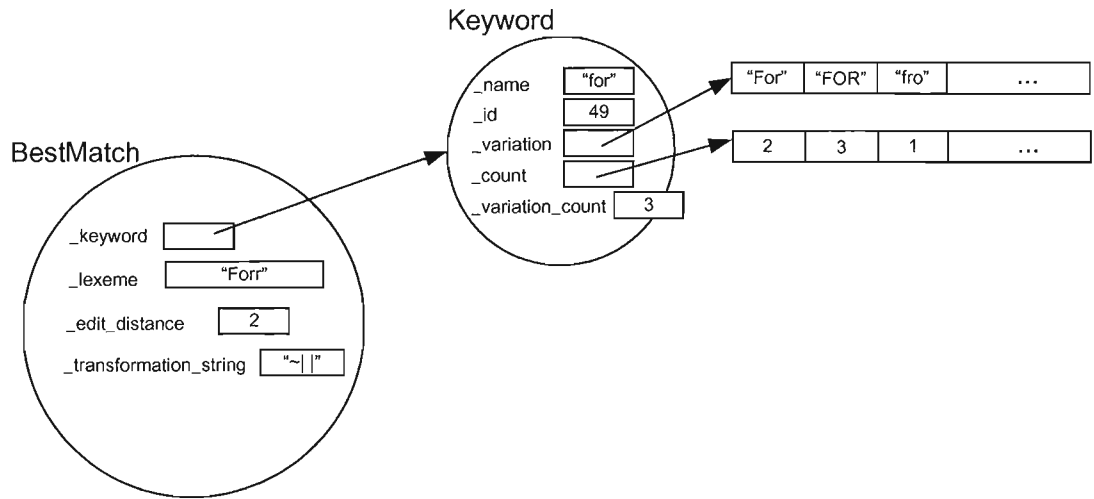
```
loop
  i = 0
  go through the _keyword array
  extract the keyword name at position i
  d = Edit_Distance (lexeme to keyword)
  if (d <= THRESHOLD)
    add it to a refinement collection
  i++
end loop
```

```
perform refinement on the refinement collection and determine if it
should be considered a keyword, extended_keyword, or as a new
identifier
```

JECA uses an additional object called “BestMatch” to assist in refining the search for appropriate potential keyword matches. The refinement collection is a Java `Collection` of `BestMatch` objects which represents the best matches of all the keywords that are similar to the identifier in question. The refinement process proceeds and applies additional rules and constraints to narrow the number of `BestMatches` until it is determined that the identifier is indeed a valid identifier or should be converted into a keyword. Once this is determined, the lexical analyzer (i.e., `TokenManager` in `JavaCC`)

returns the appropriate Token to the parser. A figure of the BestMatch object is presented in Figure 7.

A member of the BestMatch object is `_transformation_string`. This member receives this value from the Edit\_Distance algorithm. The Edit\_Distance algorithm accepts two strings for comparison and determines the closeness of these strings by performing insertions, deletions, and character replacements (Sykes & Franek, 2003). The cost for an insertion, deletion, transposition, or character change is 1. Figure 8 depicts a transformation string given two strings “FORr” and “for.” The algorithm is quite flexible and can be easily modified to accommodate various scenarios. For example, the edit distance in Figure 8 could be 2 (i.e., case-mismatch “F” and an additional “r”). It could also be configured to produce an edit distance of 1.5 (i.e., case-mismatch = 0.5 and 1 for the additional “r”) or any other cost depending on setting some switches. The rationale behind this is based on the premise that the algorithm should draw close relationships between strings that have the correct sequence of characters but may not have the correct case. Researchers in the area of education and psychology believe this concept is pedagogically sound (O'Reilly & Munakata, 2000). A student who uses “For” instead of “for” has a clearer conceptual understanding of the “for loop” construct than a student who uses “Fore” for instance. These different cognitive models are reflected in the algorithm.



*Figure 7.* BestMatch object—used for the refinement process in determining an identifier or a keyword.

```
Forr  
~| |  
fo-r
```

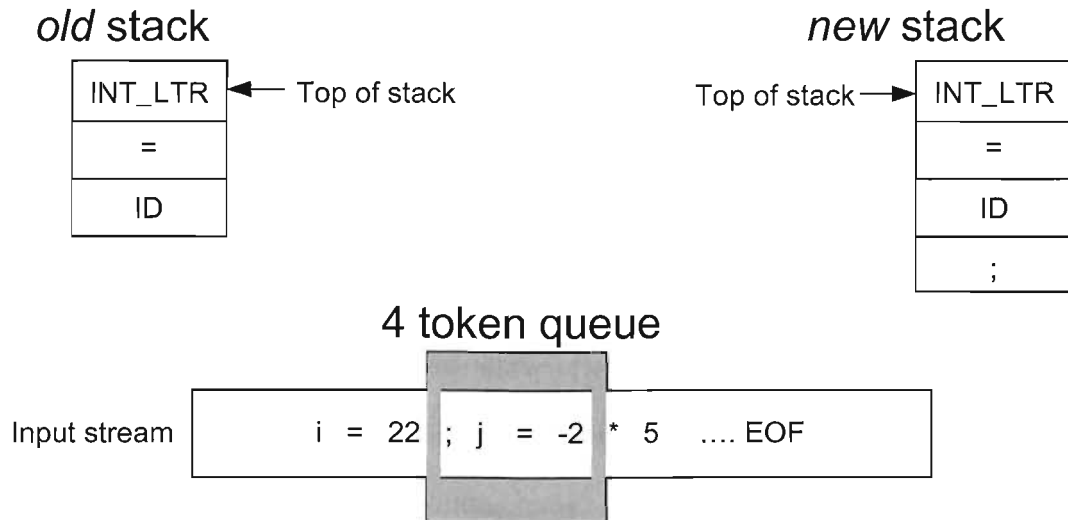
*Figure 8.* BestMatch member contains the Transformation string from Edit\_Distance algorithm.

### ***Second Component of JECA: Error Recovery in the Parser***

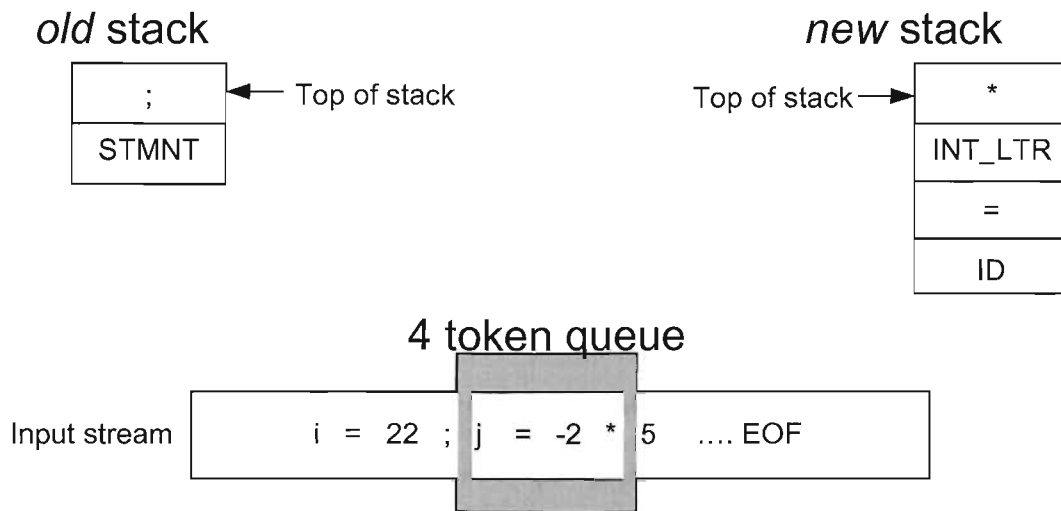
JECA's parser component algorithm implementation is loosely based on the Burke-Fisher Error Recovery algorithm (Burke & Fisher, 1987; Fischer & LeBlanc, 1991). This algorithm exhaustively tries single token insertion, deletion, or replacement at every point within  $k$  tokens before where the error occurs. In other words,  $k$  represents a window of tokens where the problem resides. Given  $N$ , representing the total number of tokens in the language, there are  $k+kN+kN$  possible deletions, insertions, and substitutions within the  $k$  token window (Burke & Fisher, 1987). The  $k$  token window is kept on a queue. In this algorithm, all semantic actions must be delayed to prevent unwanted side effects until parse is validated (Burke & Fisher, 1987).

The Burke-Fisher Error Recovery algorithm uses 2 stacks, *current* and *old*, and a *queue* of  $k$  tokens (Burke & Fisher, 1987). *old* stack contains all successfully parsed tokens so far. *current* stack contains potential tokens covering a window of the next  $k$  tokens. *old* stack and *queue* are used together to reparse string after replacement, deletion or insertion of single token into *queue*. Figure 9 and Figure 10 depict an example using the Burke-Fisher error recovery algorithm.

The proposed parser error recovery algorithm for JECA is similar in nature to the Burke-Fisher algorithm. However, there are some significant differences. First, since JECA is aimed at the beginner Java programmer, the size of the source program will always be very small (i.e., 50 lines of code or less). As a result, a Vector (i.e., `java.lang.Vector`) Abstract Data Type (ADT) is used to store the entire source program in memory. In this fashion, the tokens can be easily traversed and manipulated, thus providing opportunities for greater analysis on the input program.



*Figure 9.* Burke-Fisher error correction algorithm with a 4-token queue in the middle of processing a statement production.



*Figure 10.* Burke-Fisher error correction algorithm with a 4-token queue completing the processing of a statement production and commencing a new production.



Second, the Burke-Fisher algorithm delays semantic actions to prevent unwanted side effects. In JECA there are no semantic actions as would be expected in a typical compiler. In other words, unlike other compilers that generally produce assembler code or intermediate code, the proposed algorithm's goal is to correct errors so that the parse will be as valid as possible. It does not have extensive semantic actions like other compilers. The output of the proposed algorithm is a modified source code that is intended to successfully parse by the standard "javac" executable (i.e., Java compiler). The standard Java compiler will be invoked next to perform the translation from the modified source program to byte code. The third main difference between Burke-Fisher's algorithm and JECA's is that the student programmer will be asked for clarification during the error recovery session. Instead of using Burke-Fisher's approach to exhaustively insert, replace, or delete tokens in a  $k$ -window token list, only the most probable tokens will be presented to the student programmer. As a result, the student has a significant degree of control over the error correction process. This is supported by an inner module which generates parse tree variations which are then tested against the parser and Java compiler. These variations are based on a number of considerations involving token replacements, deletions, insertions, and transpositions. A competition is arranged such that the parse tree(s) that succeed in recognizing the most tokens in the source code are selected for further scrutiny. It then becomes a competition among the best trees to determine the appropriate course of action in terms of determining the specific hints issued for the student. Table 4 depicts this internal JECA functionality. Please note the student does not see any of these computations.

Table 4

*Internal JECA Parse Tree Permutations and Competition for the Selection of the Best Trees*

**Given the following program:**

```

1 public class Test {
2     public static void main(String args []){
3         iint sum = 0 ;
4         FOR ( Int i=0; i<10  i++ )    // missing ';'
5             sum = smu + i;
6     }
7 }
8 }

```

**and submitting it to JECA will yield a ParseException stating:**

Line 4 Column 30

Offending token: kind=>identifier, image=> "i"

Previous to Offending token: kind=>integer\_literal, image ==> "10"

The ParseException contains a list of expected tokens:

Expected ...

```

;
=
>
<
==
<=
>=
etc.

```

JECA takes this "expected" list, creates permutations on the base parse tree involving insertions, deletions, replacements, and transpositions, and then sets up the competition to determine the best tree...

Nothing compiled successfully...but here is the best tree...

```

public class Test {
    public static void main(String args [] ) {
        int sum = 0 ;
        for ( int i = 0; i < 10; i++ )
            sum = smu + i;
    }
}

```

The fourth difference between the Burke-Fisher algorithm and JECA is that the parsing stops when it encounters a situation that it cannot satisfy the current production. The justification for this stems from the philosophy behind teaching beginning programmers (Anderson et al., 1995; Sykes, 2003). It is important that the student programmer does not become overwhelmed by the number of error messages produced by compilers when errors occur (Graesser et al., 2001; Koedinger, 2001). Rather, it is more helpful to:

1. extract detailed information regarding the single error message and stop parsing;
2. provide *one* clear and meaningful error message to the student; and
3. encourage the student to make the correction (O'Reilly & Munakata, 2000).

### **Initial Java Intelligent Tutoring System Implementation**

The JITS infrastructure supports the student via a browser accessing information from the tutor via an HTTP request/response process model. The processing is accomplished by JavaBeans™ within a servlet engine web server. The presentation layer uses JavaServer Pages™ technology which communicates to the bean representing the student and creates an XHTML page for the student's browser. During processing, the bean gathers all the information about the student's code and submits it to JECA for processing. The infrastructure architecture uses a JDBC connection from the JavaBeans™ to an external database which stores and retrieves specific information about the student including student history and performance statistics.

The implemented architecture has numerous benefits (Pawlan, 2004). It is scalable, platform independent, and lightweight (Pawlan, 2004). The student will never need to install software on his/her machine and will not need a high-speed network connection to use JITS. Other benefits include fast execution, as all processing is done on the middle-tier web server, currently equipped with 4GB RAM and 2 Pentium-IV processors. The net result is a product that increases the accessibility for JITS to many students—a vital requirement for an equitable and successful educational product in today’s Internet-ready community. Figure 11 presents a pictorial view of the JITS multithreaded distributed Web-based Infrastructure.

### **Human-Computer Interaction**

The interface for computer-based programming tutors was given careful consideration during the design of the Java Intelligent Tutoring System (JITS). The user interface is based on a presentation format implemented in many popular Integrated Development Environments used by professional programmers (e.g., Visual Café, JDeveloper, JBuilder, etc.). The JITS login screen and user interface are shown in Figure 12 and Figure 13 respectively.

Students are presented with a problem, the problem specification, the skeleton code, the code editor, and a number of buttons with which to interact with the tutor. The student types in his/her solution in the Source Code Area (see Figure 13) and presses “Submit.” This invokes a call to the corresponding JavaBean™ representing the student.

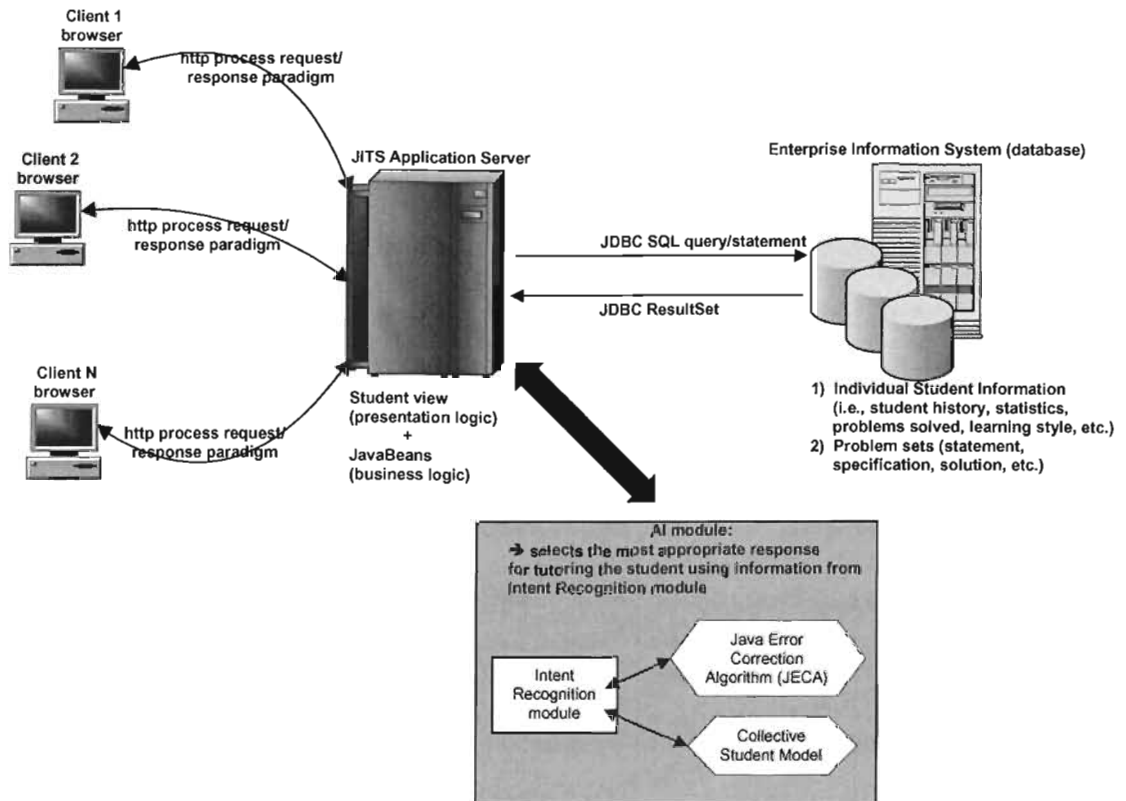


Figure 11. JITS multithreaded distributed web-based infrastructure.

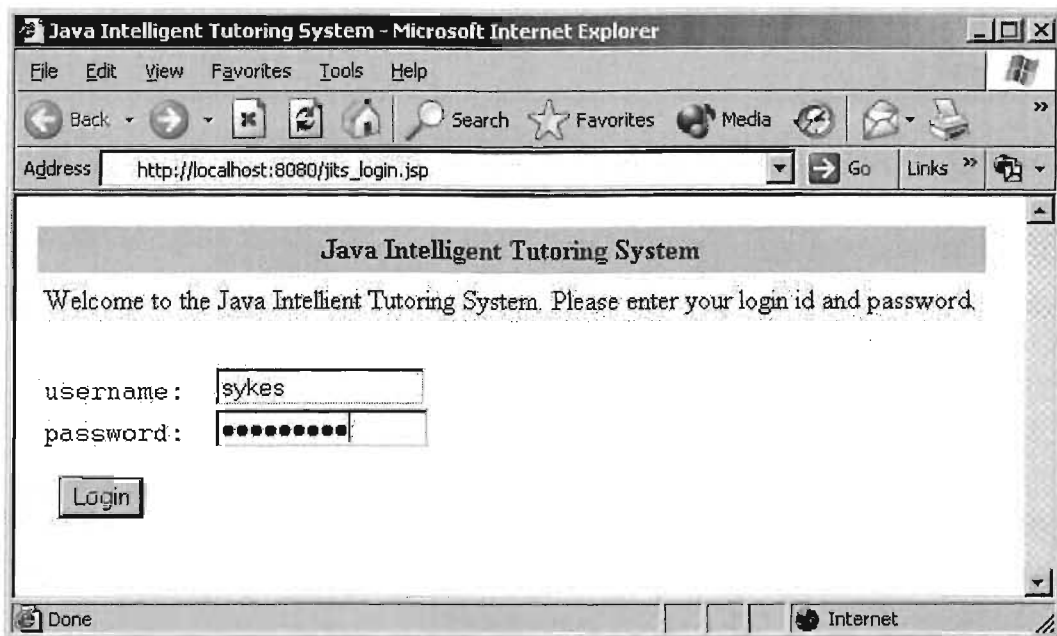


Figure 12. JITS login screen.

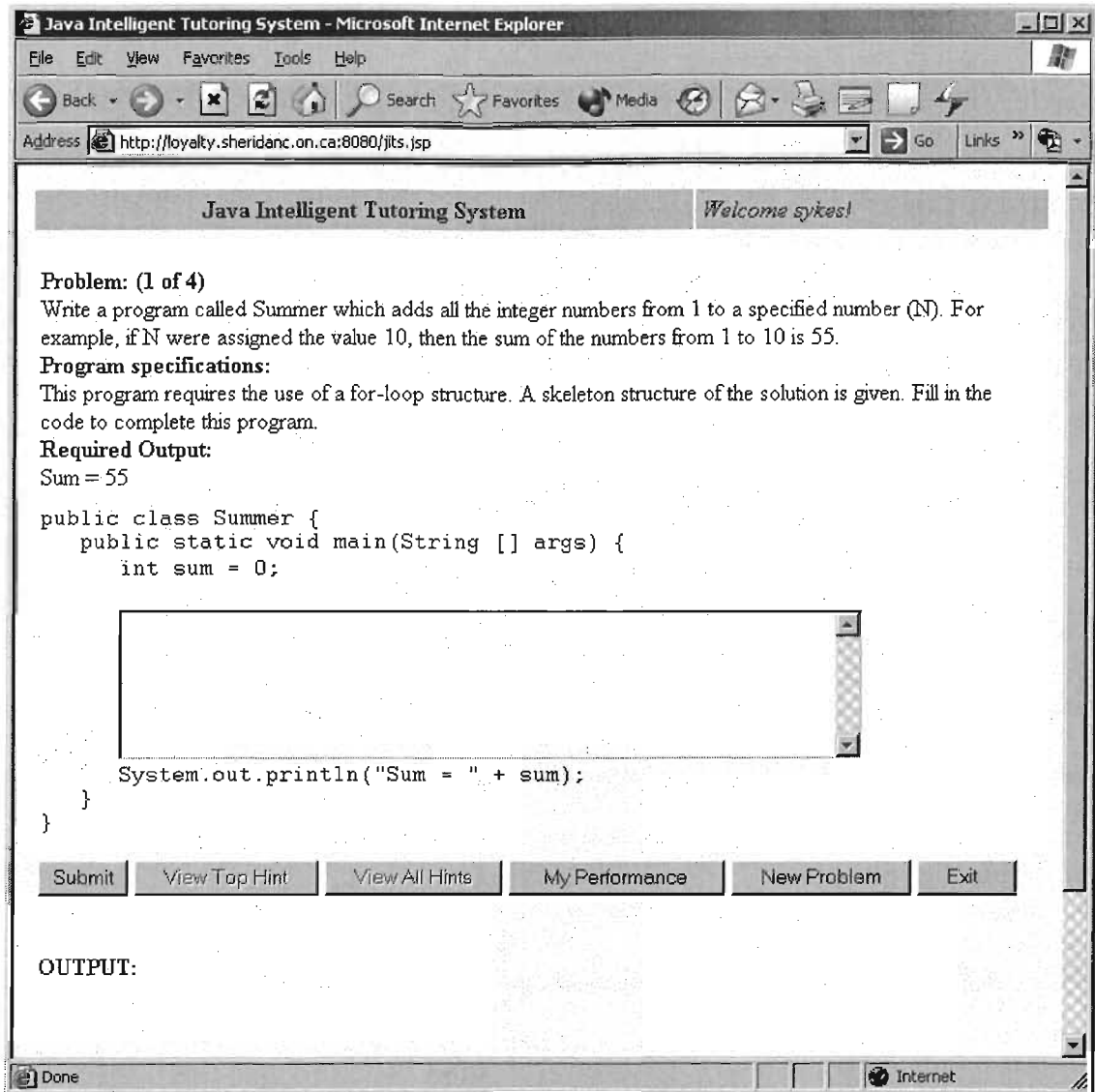


Figure 13. Initial JITS User Interface.

The code is then dispatched to JECA, which processes the submission and generates a set of appropriate hint objects. The student, at any time, may explicitly request a hint from JITS by pressing “View Top Hint” or “View All Hints.” The hints are dynamically generated based on the problem details and the student’s submission.

The initial design of JITS used both static content and dynamic content. In other words, some of the information was loaded into the student’s browser from the database while other information was hard coded. The dynamic content was extracted via JDBC from an ORACLE database schema and embedded into the JITS web page. The initial schema is presented in Table 5.

### **Hint Generation**

An additional design consideration is the categories of hints that are generated by JECA for JITS. There are a number of different categories of hints that may be created as a result of the student’s code submission. They are presented in Figure 14.

A `KEYWORD_REPLACEMENT_HINT` arises from a situation where the student typed in a suitably close representation to a Java keyword. For instance, if the student typed in “Whiles,” this would be interpreted as the keyword “while.” An `EXTENDED_TYPE_REPLACEMENT_HINT` is when the student wrote “Sting” which will be interpreted as “String”—the `java.lang.String` class. An `IDENTIFIER_REPLACEMENT_HINT` is used in the situation where a suitably close match to an existing identifier has been found.



Table 5

*Initial JITS ORACLE Schema Tables*

---

```
CREATE TABLE PROBLEMS (  
    problem_id          NUMBER(3),  
    problem_desc        VARCHAR2(400) NOT NULL,  
    problem_spec        VARCHAR2(400) NOT NULL,  
    problem_output      VARCHAR2(50),  
    template_top_section VARCHAR2(400),  
    template_bottom_section VARCHAR2(400),  
    problem_difficulty  VARCHAR2(20)  
);
```

```
CREATE TABLE STUDENTS (  
    student_name        VARCHAR2(30),  
    student_password    VARCHAR2(15),  
    problem_set_id      NUMBER(3),  
    problem_id          NUMBER(3),  
    skill_level         NUMBER(3),  
    performance_rating  NUMBER(3),  
    performance_history VARCHAR2(2000),  
    times_connected    NUMBER(5),  
    date_last_connection VARCHAR2(30),  
    picture             LONG RAW,  
);
```

```
CREATE TABLE STUDENT_PROBLEMS (  
    student_name        VARCHAR2(30),  
    problem_set_id      NUMBER(3),  
    problem_id          NUMBER(3),  
    number_of_attempts  NUMBER(3),  
    solved              CHAR(1),  
    students_solution   VARCHAR2(500),  
    solution_date       VARCHAR2(30),  
);
```

---

```
KEYWORD_REPLACEMENT_HINT = 1;  
EXTENDED_TYPE_REPLACEMENT_HINT = 2;  
IDENTIFIER_REPLACEMENT_HINT = 3;  
GRAMMATICAL_HINT = 4;  
CLOSE_BUT_LOGIC_ERROR = 5;  
SUCCESSFULLY_SOLVED_PROBLEM = 6;  
GENERAL_HINT = 7;  
OTHER_TYPE_OF_HINT = 8;
```

*Figure 14.* Hint categories.

For example, consider the following snippet of code:

```
int my_int = 0;           // declaration
my_it = my_intt + 1;    // and use
```

There would be two IDENTIFIER\_REPLACEMENT\_HINTs generated for this piece of code:

***Identifier Replacement Hint: Would you like me to replace "my\_it" with "my\_int"?***  
***Identifier Replacement Hint: Would you like me to replace "my\_intt" with "my\_int"?***

A GRAMMATICAL\_HINT is generated when the parser fails on a particular production in the Java grammar. Specific information regarding the error is recorded in the Hint object depicted. The last two types of hints are GENERAL\_HINT and OTHER\_TYPE\_OF\_HINT. GENERAL\_HINT is used in the situation when the student is far from the solution path and needs to be realigned with the program statement and program specifications for the posed problem. If the student's code compiles but produces output that is not the same as the required output, as specified in the problem statement, the CLOSE\_BUT\_LOGIC\_ERROR is used. When the student solves the problem the SUCCESSFULLY\_SOLVED\_PROBLEM hint is used. Last, OTHER\_TYPE\_OF\_HINT is reserved for future research.

There are a number of important pieces of information represented in a Hint object. The Hint object is depicted in Figure 15. The \_type member corresponds with one of the six types of categories of Hints currently supported in JECA. The \_col and \_line members specify where the error occurred. The \_line\_of\_code and \_error\_pointer represent the source code and the exact location of where the error occurred. There are two tokens to assist in identifying where the error occurred in terms of the tokens. \_offending\_token represents the precise token the parser failed on,

and `_previous_to_offending_token` represents the last successfully parsed token during parsing. The `_hint` member is a String summarizing the actual hint relying on the values of other data members in this object. It is intended to be used during the feedback process during student tutoring. The last member of the Hint class is the `_confidence`, which will be assigned an integer from 1 to 10. A confidence value of 1 indicates a high level of certainty, indicating the suggested hint is correct and will bring the student closer to a compiled program. On the other hand, a confidence value of 10, indicates uncertainty on behalf of the hint generated. In these situations, the student will have to use their own judgment based on the detailed information provided to them by the Hint objects, namely the data members, `_type`, `_col`, `_line`, `_line_of_code`, `_error_pointer`, `_offending_Token`, and `_previous_to_offending_Token`.

An example follows to illustrate these design aspects of the proposed error correction algorithm. Given the source program depicted in Figure 16, JECA would modify the program to the program depicted in Figure 17. As a result, the following Hint objects would be created by JECA:

- 1) **Keyword replacement hint: Would you like me to replace "Int" with "int"?**
- 2) **Keyword replacement hint: Would you like me to replace "FOR" with "for"?**
- 3) **Keyword replacement hint: Would you like me to replace "iint" with "int"?**
- 4) **Grammatical hint: Look near line: 8 column: 10. Look between the "++" and the "sum"**

The following section depicts how the Hint objects are used in a typical dialog between JITS (via the supporting JECA module) and the student programmer.

## Hint

\_type GRAMMATICAL\_HINT

\_col 10

\_line 8

\_line\_of\_code for (int i=0; i <=10; i++

\_error\_pointer ^

\_offending\_Token sum

\_previous\_to\_offending\_Token ++

\_corrected\_line\_of\_code for (int i=0; i <=10; i++ )

\_hint Grammatical hint: Look near line: 8 column: 10. Look between the "++" and the "sum"

\_confidence 1

Figure 15. A JECA Hint object representing a grammatical error.

```
public class Test {  
    public static void main() {  
        Int sum = 0;  
        For (iint i=0; i<=10; i++  
            sum = sum + i;  
        System.out.println("Sum is:" + sum);  
    }  
}
```

*Figure 16.* Arithmetic sum Java program with grammatical errors and syntax errors.

```
public class Test {  
    public static void main(String args []) {  
        int sum = 0;  
        for (int i=0; i <=10; i++ )  
            sum = sum + i ;  
        System.out.println("Sum is:" + sum);  
    }  
}
```

*Figure 17.* Internally corrected JECA source program for the arithmetic sum problem.

Using the example presented in Figure 16, focusing only on the area where the student enters code in the “source code area” (see Figure 13), a dialogue occurs between JITS and the student. Table 6 presents a typical dialogue that would occur between JITS and the student for this example.

The tutoring process is dynamic. At any time, the student is able to interject, disagree with JITS’ suggestions, or modify the source code. JECA is designed to be invoked many times to support the JITS tutoring process.

JECA is significantly different from other standard Java compilers. Given the source program in Figure 16, an ordinary java compiler would produce the following:

```
Test.java:5: ')' expected
    Forr (Int i=0; i <=10;    i++
        ^
Test.java:5: not a statement
    Forr (Int i=0; i <=10;    i++
        ^
Test.java:5: ';' expected
    Forr (Int i=0; i <=10;    i++
                                ^
3 errors
```

The embedded JECA system in JITS is much clearer and more helpful than standard Java error systems. JECA has been designed for the beginner Java programmer and intelligently recognizes the intent behind the student’s code submissions.



Table 6

*Hint Objects Utilization and Typical Dialogue Between JITS and the Student*

Student's submission:

```
For (intt i = 1; i <= 10; i++ {
    sum = smu + i;
}
```

**JITS:** Would you like to replace "For" with "for"? (*Keyword replacement hint*)

**Student:** Clicks Yes, or changes the code manually.

**Resulting code:**

```
for (intt i = 1; i <= 10; i++ {
    sum = smu + i;
}
```

**JITS:** Would you like to replace "Int" with "int"? (*Keyword replacement hint*)

**Student:** Clicks Yes, or changes the code manually.

**Resulting code:**

```
for (int i = 1; i <= 10; i++ {
    sum = smu + i;
}
```

**JITS:** Look near line: 4 column: 37.

Look between the "++" and the "{" (*Grammatical hint*)

**JITS elaborates:**

**HINT STRING :**

```
for ( int i=0; i<10; i++    {
```

← Missing ")"

**CORRECTED CODE:**

```
for ( int i=0; i<10; i++) {
```

**Confidence... : 1** (*high certainty*)

**Student:** Makes the appropriate changes to the code.

**Resulting code:**

```
for ( int i = 1; i <= 10; i++) {
    sum = smu + i;
}
```

**JITS:** Would you like to replace "smu" with "sum"? (*Identifier replacement hint*)

**Student:** Clicks Yes, or changes the code manually.

**Resulting code:**

```
for ( int i = 1; i <= 10; i++) {
    sum = sum + i;
}
```

### **Initial User Modeling Design**

The initial JITS user modeling component tracked some information, but once the students finished a session this information was lost. For instance, the information that was capturing included: the number of attempts for each problem the student has tried; the number and type of misconceptions involving keywords, extended keywords, and identifiers are recorded (e.g., “For,” “fro” instead of “for,” etc.); whether the student has solved the problem or not; and the difficulty level of the problems that have been solved by the student.

## **CHAPTER FIVE: METHODOLOGY AND PROCEDURES**

The methodology employed in this dissertation is supported by two distinct research components. The first component is related to the manner in which JITS was designed and constructed. In this research section, students and professors using the prototype JITS offered suggestions and comments for the improvement of JITS. The new knowledge was fed back into the redesign and construction of JITS. Beyond the initial development of JITS, a cyclic process was used: design, develop, test, modify, redesign, redevelop, retest, etc. This research methodology involved qualitative instrumentation including observation, surveys, and personal interviews. The goal of this methodology was to improve JITS.

The second component of the methodology is related to the manner in which JITS was evaluated. In order to determine the degree and quality of learning that took place by students using the Java Intelligent Tutoring System, a quantitative investigation on performance scores was conducted. The research methodology for this section involved an experimental design with repeated measures. As a result, the researcher was able to compare pretest and posttest performance differences as well group differences (i.e., Control versus JITSC). One advantage of this type of analysis is that interaction effects were able to be calculated and analyzed.

### **Subjects**

The population of this study were students across the province taking a comparable course in programming. The sample in this study were the students in their first year of college taking a beginner Java programming course at the Sheridan Institute of Technology and Advanced Learning. During the summer of June to August 2004,

there were two such classes taking this course. One class was located at the Davis campus. This class was the experimental group (i.e., JITSC). The other class was located at the Trafalgar Road campus. This class was the control group, which consisted of 23 students. One professor taught both classes for the first 7 weeks. After a midterm break for week 8 in the term, another professor took over and taught both classes for the remainder of the term (i.e., for the last 7 weeks). Fourteen students consented to try the Java Intelligent Tutoring System (i.e., JITSC). Approximately every week, ½ to 1 hour long sessions were conducted by the researcher to elicit specific information about their experience with the Java Intelligent Tutoring System.

A similar study was conducted during the fall of September to December 2004. During this period there were two instructors teaching a first year Java programming course. Instructor “A” had two classes; the JITSC group consisted of fourteen students, and the C group consisted of 25 students. Instructor “B” had three classes; the JITSC group consisted of fourteen students, the C1 group consisted of eighteen students, and the C2 group consisted of 23 students. Both instructors taught for the entire semester (i.e., 14 consecutive weeks). Every week, ½ to 1 hour long sessions were conducted by the researcher to elicit specific information about their experience with the JITS.

During both time periods (Summer and Fall 2004) the JITSC group were talked to and observed during the ½ to 1 hour long sessions. Additionally, many JITSC students emailed the researcher with comments and suggestions for improvement. The manner in which students were interviewed was primarily individually based; however, there were some occasions when an issue was raised that were a shared concern among several students. The total number of students involved in this entire research project (i.e., all

JITSC students) was  $14 \times 3 = 42$ . The kind of note taking procedures were observations recorded in a researcher's log book. Such observations included information regarding individual student's progress through a specific programming problem in JITS. For example, the programming topic, the problem number, types of mistakes and errors, and JITS' response to the student were all recorded in the researcher's log book.

Professors were also selected to participate in this study. The selection of professors was based on a number of factors including their knowledge of the Java programming language, level of course offerings, and interest in offering critical opinions on the Java Intelligent Tutoring System. A total of 4 professors were selected for this study.

### **Statement of Procedures**

Two global procedures were required:

**PART A** Design and develop the Java Intelligent Tutoring System; and

**PART B** Data gathering for the quantitative investigation.

#### ***PART A: Design and Development Procedure for the JITS***

Although the initial JITS was already designed and developed, the system had never been tested by students. Once students and instructors started working with JITS, there were many features to be included. The cyclic process involving designing, developing, and testing, and back to redesigning was used for the ongoing refinement of JITS to ensure that it adequately met the needs of students and instructors.

An interview-style survey sheet was constructed to aid in gathering input from students in the JITCS groups and professors. The survey included six open-ended questions to facilitate a great number of perspectives and opinions. One of the

measurement instruments for this component of the study was this survey, depicted in Table 7. By presenting the survey to students and teachers who have used JITS, feedback representative of these two perspectives was gathered. Additionally, the researcher often visited the classroom to informally assess JITS. Between ½ hour and 1 hour per week was spent with students and professors, who offered important suggestions for improving JITS. This information was recorded in the researcher's logbook. This form of data gathering proved to be the most effective way of receiving feedback from students and instructors for the refinement and improvement of JITS.

***PART B: Quantitative Investigation Procedures***

A series of programming problems were developed for both the Java Intelligent Tutoring System Class (JITSC) and the Control group. Students in the control group were taught in a traditional format such as instructor-led instruction, group-work, demonstration, etc. The JITSC group received this same instruction as well. This investigation involved both intragroup and intergroup comparison of student achievement by using pre- and posttest performance tests. Performance tests are small quizzes containing two to four programming problems and space for the student to write their solutions.

Table 7

*Interview Sheet*

<b>Project Interview</b>				
<p>I am conducting a survey of those participants who were taught using the Java Intelligent Tutoring System at Sheridan. The information gathered from our interview will be used for my research. This involves determining the effectiveness of learning in this environment. For each question, select the most appropriate response based on the following scale:            1 = strongly favourable to the concept, 2 = somewhat favourable to the concept, 3 = undecided, 4 = somewhat unfavourable to the concept, 5 = strongly unfavourable to the concept. The following questions will be asked during the interview.</p>				
1. How do you rate the Java Intelligent Tutoring System's usefulness?				
<b>Very Useful</b>				<b>Not Useful</b>
1	2	3	4	5
Comments: _____				
2. Do you feel the Java Intelligent Tutoring System is beneficial to your studies? List and explain the advantages/disadvantages of this learning environment.				
<b>Very Beneficial</b>				<b>No Benefits</b>
1	2	3	4	5
Comments: _____				
3. Compare JITS with a traditional classroom. Do you feel JITS is better or worse than an ordinary classroom teaching environment? Identify any similarities and differences between a traditional classroom experience and the JITS learning experience.				
<b>JITS is much better than traditional classroom</b>				<b>JITS is much worse than traditional classroom</b>
1	2	3	4	5
Comments: _____				
4. How do you rate the ease with which you use and understand the tutoring style of the JITS?				
<b>Very easy to use &amp; understand</b>				<b>Very difficult to use &amp; understand</b>
1	2	3	4	5
Comments: _____				
5. Have you enjoyed JITS? Explain why or why not.				
<b>Very Enjoyable</b>				<b>Not enjoyable</b>
1	2	3	4	5
Comments: _____				
6. Do you feel you learn more detailed information or about the same as a regular classroom when using JITS? Explain why or why not.				
<b>Learn Better</b>				<b>Learn the same</b>
1	2	3	4	5
Comments: _____				

The performance tests were administered at the beginning of the term and at midterm (i.e., week 7 in the term). As a result, there were statistical analysis opportunities. Figure 18 presents a sample performance test.

These nonsubjective measurements quantify the performance level of students prior to exposure to JITS and allow comparison to the level after exposure to JITS. In addition, comparisons were made between the JITSC and the Control group. Regardless of the measurement category, there were three possible outcomes: (a) there was no difference in performance between the JITS group and the Control group; (b) students in the JITS group performed higher than the Control group; and (c) students in the JITS group performed lower than the Control group. This is summarized as follows:

- | JITSC | = | C |      (no difference), or
- | JITSC | > | C |      (Java ITS resulted in higher performance than C), or
- | JITSC | < | C |      (Java ITS resulted in lower performance than C).

The following section describes the details of the way in which this quantitative investigation procedure was performed. Prepare a series of programming problems for the Control group:

1. Select a series of topics that are routinely taught to students when learning the Java programming language in one semester, for example, datatypes, identifiers, scope, methods, looping constructs, and arrays;
2. develop a series of programming problems that are based on those selected topics; and



### For-loop quiz

- Q1. Write a program that computes the sum of all the odd numbers from 1 to 5000. Use a for-loop in your solution. Complete the following program in the space provided.

```
public class Odd {
    public static void main(String[] args)
    {
        int total = 0;

        _____

        _____

        _____

        System.out.println("Total of all odd numbers is " + total);
    }
}
```

- Q2. The *Fibonacci* sequence is described by  $u(n+2) = u(n+1) + u(n)$ ; where  $u(0)=1$ ,  $u(1)=1$ . In other words it looks like: 1, 1, 2, 3, 5, 8, 13, ... etc. So,  $u(2) = u(1) + u(0) = 1+1=2$ ,  $u(3) = u(2) + u(1) = 2+1=3$ , etc.

Complete the following program using a for-loop that computes the Fibonacci number for  $u(20)$ . Use the space provided below.

```
public class Fibonacci {
    public static void main(String[] args)
    {
        int total = 0;
        int start = 0;
        int end = 20;
        int last_fib_number = 1;

        _____

        _____

        _____

        System.out.println("The Fibonacci number for 20 is " + total);
    }
}
```

Figure 18. Sample performance test for quantitative investigation.

3. ensure that they meet the requirements of the unit or subunit of study by encouraging several teachers with expertise in this area to review the series of lessons developed.

Prepare a series of programming problems for the Java Intelligent Tutoring System:

1. Select the same topical area corresponding to the Control group's lessons;
2. develop a series of problems for the Java Intelligent Tutoring System; and
3. ensure that they meet the requirements of the unit or subunit of study of the Java programming language by encouraging several teachers with expertise in this area to test the series of lessons developed in the JITS.

Collect data to determine the effectiveness of the learning experience by using JITS by:

1. conducting the pretest for baseline data on students in the JITSC and Control groups prior to exposure to the experiment.
2. determining the mean and standard deviation for the JITSC and Control groups.
3. conducting tutoring sessions using the Java Intelligent Tutoring System to the experimental group.
4. conducting traditional-form lessons for the Control group.
5. conducting the posttest given to both JITSC and Control group.<sup>2</sup>
6. computing standard statistical measures between pre- and postexposure to JITS lesson and traditional-form lesson for the two groups respectively (i.e., JITSC and Control groups).

---

<sup>2</sup> All tests for this study were knowledge-based and skill-set-based programming problems corresponding to the material covered in the classes.

7. Computing additional statistical information such as two-way ANOVA with repeated measures. The template of the organizational layout of the results is presented in the form of tables as shown in Table 8.
8. A cyclic process involving designing, developing, testing, and back to redesigning was used for the development and refinement of JITS to ensure student and instructor satisfaction. For the quantitative component, the methodologies for data-gathering of student performance were presented. This involved conducting pre- and posttests to the Control and JITSC groups. The posttests were administered after the JITSC group used the Java Intelligent Tutoring System for a period of time.

### **Methodology and Procedures: A Summary**

This chapter described the methodology by which this study was conducted. It contains the specifications of the procedures used for which data were gathered for the refinement of JITS and the qualitative aspects of this study.

A cyclic process involving designing, developing, and testing, and back to redesigning was used for the development and refinement of JITS to ensure student and instructor satisfaction. For the quantitative component, the methodologies for data-gathering of student performance were presented. This involved conducting pre- and posttests to the Control and JITSC groups. The posttests were administered after the JITSC group used the Java Intelligent Tutoring System for a period of time. The last component of the methodology was to compute standard descriptive statistical measures and to compute ANOVAs to determine degree of difference between JITSC and the Control groups. In the first study a three-way analysis of variance (ANOVA) was computed with Group (C, JITS), Semester-Point (first half, second half) and Test –Time

(pretest, posttest) as the independent variables, and test score as the dependent variable.

In this analysis, Semester-Point and Test-Time were treated as repeated measures.” For

the second study a two-way ANOVA was computed with Group (C, JITS) and Test –

Time (pretest, posttest) as the independent variables, and test score as the dependent

variable.

Table 8

*Performance of Students in JITSC and Control Prior Exposure to JITS and After Exposure to JITS*

**JITSC**

<i>Student</i>	<i>Pre-Test</i>	<i>Post-Test</i>
S <sub>1</sub>		
S <sub>2</sub>		
...		
mean		
standard deviation		

**Control**

<i>Student</i>	<i>Pre-Test</i>	<i>Post-Test</i>
C <sub>1</sub>		
C <sub>2</sub>		
...		
mean		
standard deviation		

It should be noted that this experiment grew in the real world—no preassigned classes were formed specifically for this study. Rather, the formation of classes under study resulted in the natural selection process of the course offerings in the institute. As a result, the data have a high degree of natural validity. On the other hand, it is a limitation, since the researcher did not have random assignment to groups. Thus, it is a quasi-experimental study.

## CHAPTER SIX: FINDINGS (ANALYSIS AND EVALUATION)

Due to the nature of this dissertation involving the extensive details relating to the design and construction of JITS and the formal evaluation of the tutor, this chapter first presents a summary of the JITS developmental research, which includes the final version of JITS as of completion of this research project. The second summary presents the results of the student performance assessment, which includes a discussion regarding the effectiveness of the JITS.

The process employed in the design and refinement of JITS was a student-centred approach which elicited students' comments for the improvement of JITS. Additionally, instructors using JITS offered comments which helped shape JITS. The methodology used was a cyclic process reflecting the students' comments: design, develop, test, modify, design, develop, test, etc. The second objective set out in this research was to determine the effectiveness of learning within this environment by comparing students exposed to JITS with those taught Java in a traditional classroom environment.

Beyond the summary sections, this chapter provides many details about both components of this research, namely, the design and refinement of JITS (including qualitative analysis) and the quantitative analysis of students using the system as it was being developed. The results are presented by way of a collection of three *Program Development Sessions*, with qualitative and quantitative findings for each. Each Program Development Session presents a number of *sections* entitled: "JITS Developmental Research," "JITS Performance Score Analysis," and "Summary and Recommendations for further JITS Development." Each *section* is further divided into *parts* typically representing research work conducted between 1 and 3 weeks in duration. Each *part* is

identified by a start date and an end date representing the scope during which the research was performed.

### **Summary of JITS Development Research**

The following section describes the completed Java Intelligent Tutoring System in terms of the user interface only. The final version of JITS includes many other features beyond the user interface; however, for brevity of this summary, these details may be found after the summary sections in this chapter.

#### ***JITS User Interface***

The user interface for the Java Intelligent Tutoring System underwent a number of significant changes throughout the duration of the research study. During some of the experiments, major changes were conducted within very short timelines to ensure the student's suggestions were taken seriously and that significant changes were done to the user interface. Figure 19 depicts the completed JITS user interface. The first section (i.e., label 1) presents a personalized welcome to the student logged in. Label 2 presents a note relative to the current state of solving the problem at hand. In this section, notes are dynamically created by JITS that are personalized to each student. Label 3 presents the problem template structure including the problem statement, the problem specifications, and the required output. This section also draws reference to the problem number out of the total number of problems available in this programming topic. At the end of Section 3, a link (i.e., label 4) is provided to a picture if the problem has a visual component (i.e., an equation or relevant drawing) to assist the student in more clearly understanding the problem. If the student clicks the link, the picture is shown in a separate window to allow the student to refer to the picture while at the same time work



Java Intelligent Tutoring System - Microsoft Internet Explorer

Address: <http://loyalty.sheridanc.on.ca/jits.jsp>

Java Intelligent Tutoring System Welcome sykes! Help Me

**NOTE:** You have attempted this problem 4 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

**Problem: (3 of 5) in Problem Set # 2 (Topic: Java Statements)**  
Write a program called Power Generator which calculates the result of a number multiplied by itself.

**Program Specifications:**  
This program requires the use of a function. A skeleton structure of the solution is given. You need to declare the variable: result [View the image for this problem.](#)

**Required Output:**  
Result = 10000

```
public class Power {
    public int powergen(int num) {
        return num * num;
    }
    public static void main(String [] args) {
        Power p = new Power();
        double result
        p.powergen(19);
        System.out.println("Result = " + result);
    }
}
```

**Programming Topics**

- Java Basics
- Java Statements
- If statement
- for loops
- do while loops
- while loops
- Arrays

Take me there

View the Tutorial

Submit View Top Hint View All Hints View Solution

Previous Problem Next Problem My Performance Exit

OUTPUT:

Figure 19. Completed version of the JITS User Interface.

with the main JITS user interface. Label 5 shows the template provided by JITS for each problem in the system. Label 6 presents the editing region where the student types his/her solution. Label 7 depicts the various buttons which the students use to interact with JITS. Buttons include “Submit” to submit a solution to a problem and to receive feedback. The two buttons, “View Top Hint” and “View All Hints” provide the means by which students can see the hints that JITS provides. The “View Solution” button provides potentially various solutions to the current problem based on the `Collective_Student_Model`. The “Previous Problem” and “Next Problem” buttons are used for navigating within a problem set. The “My Performance” button yields detailed information about the student’s performance including problems solved, problems attempted, the number of attempts for each problem, and comparison information to the “average” JITS student. Links are provided in the “My Performance” output for rapid access to any problem the student wishes to retry. Label 8 shows where the majority of the responses from JITS are presented. Information such as hints, solutions, performance scores, and errors are all shown in this area of JITS. Label 9 presents the choices of the various programming topics that the student may choose. The “Take Me There” button is used to bring the student to the selected programming topic. Label 10 presents the “View the Tutorial” button which launches the JITS Tutorial window. The tutorial window may be viewed at the same time as the student is working with the main JITS user interface (i.e., the tutorial may be referenced while working on a problem in JITS). Label 11 shows the “Help Me” button which opens a separate window displaying the screenshot of JITS with labels to all of the components in JITS.

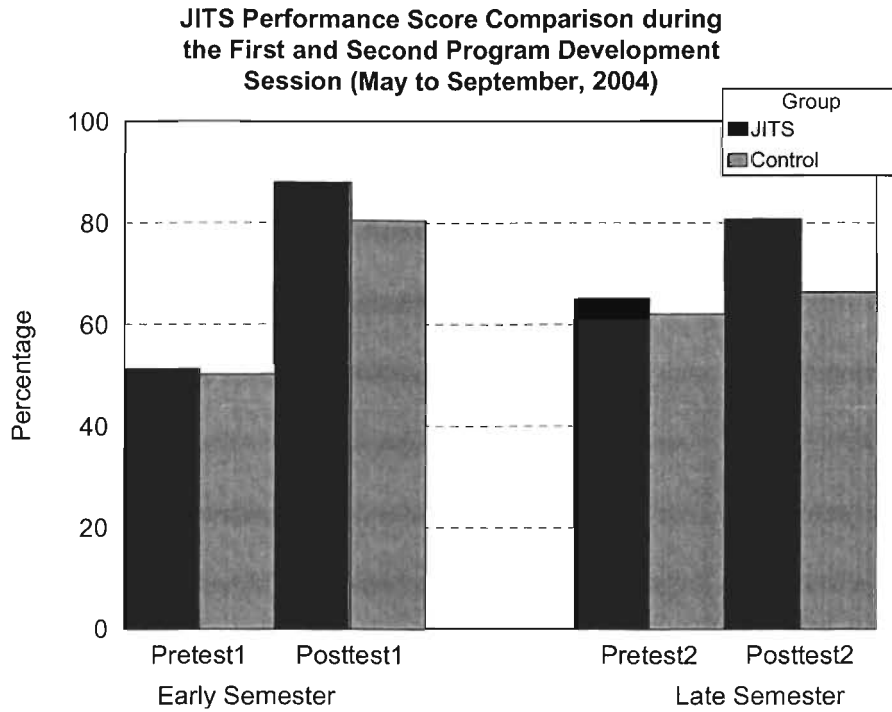
The purpose of this window is to orient new users of JITS so that they feel supported and can more quickly become productive in this ITS. The last label (i.e., 12) is the “Exit” button. This button brings up a screen which thanks the student for trying out the system and performs some system-wide cleanup procedures behind the scene.

### **Summary of Student Performance Score Analysis**

This summary presents the main findings regarding student performance scores involving students that were exposed to JITS (experimental groups) and those taught in a traditional classroom environment (control groups). Two main testing periods are presented in this section consisting of equal periods of time (i.e., one semester). The first semester was from May 2004 to September 2004 which entailed the *First Program Development Session* (7 weeks: May to July) and the *Second Program Development Session* (7 weeks: July to August). The second term was from September 2004 to December 2004 (14 weeks in total). This term entailed the *Third Program Development Session*.

#### ***First and Second Program Development Sessions (May 2004 to September 2004)***

A 2 x 2 x 2, three-way Analysis of Variance (ANOVA) was computed with Group (i.e., JITS, Control), Time (i.e., Early Semester [May to July], Late Semester [July to August]), and Test (i.e., pretest, posttest) as the independent variables, with the last two variables treated as repeated measures. The dependent variable was performance on the competency tests. The main effect for Test,  $F(1, 35) = 119.43, p < .001$ , was qualified by a Test by Group interaction,  $F(1, 35) = 4.98, p < .05$ , and a Test by Time interaction,  $F(1, 35) = 43.82, p < .001$ . As may be seen in Figure 20, the Test by Time interaction is due to a larger gap between pretest and posttest early in the semester as compared to the



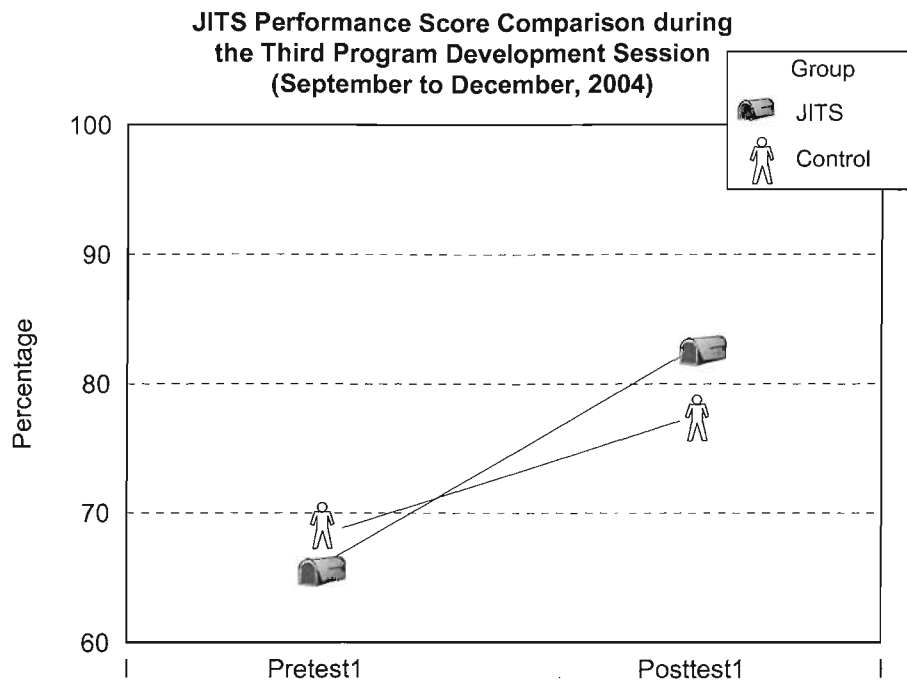
*Figure 20.* Showing (a) the two-way Semester by Test interaction due to the smaller gap between pretest and posttest later in the semester, and (b) the two-way Group by test interaction due to the superior performance of the JITS group at posttest.

difference late in the semester. This would seem to indicate, and logically so, a more dramatic learning curve early in the semester. More interesting, and more to the point of this study, was the Test by Group interaction, which showed the superior performance by the JITS group. Since there was no three-way interaction we can infer that the JITS group was performing better than the comparison group at posttest for both points of time (i.e., early semester, and late semester). (Means and standard deviations are reported later in this chapter, in Table 15).

### ***Third Program Development Session (September 2004 to December 2004)***

A 2 x 2, two-way ANOVA was computed with Group (i.e., JITS, Control) and Test (pretest, posttest) as the independent variables, with the second variable treated as a repeated measure. The dependent variable was performance on the competency tests. The main effect for Test,  $F(1, 92) = 61.12, p < .001$ , was qualified by a Test by Group interaction,  $F(1, 92) = 5.36, p < .025$ . As may be seen in Figure 21, the Test by Group interaction is due to the superior performance of the JITS group at posttest. (Means and standard deviations are reported in Table 20 which is found later in this chapter).

The remainder of this chapter presents various details associated with the two components of this dissertation, namely, the refinement of JITS (including the qualitative analysis) and the quantitative analysis of students using the system as it was being refined. The findings are presented as a collection of *Program Development Sessions*, with qualitative and quantitative findings for each.



*Figure 21.* Showing the two-way Group by Test interaction for responses indicating superior performance for the JITS group at post-test.

Within each Program Development Session there are a three numbered *sections* entitled: “JITS Developmental Research,” “JITS Performance Score Analysis,” and “Summary and Recommendations for further JITS Development.” Each section is further divided into *parts* which represent work conducted between 1 and 3 weeks in length.

The process employed in the design and refinement of JITS was a student-centred approach which elicited students’ comments for the improvement of JITS. Additionally, instructors using JITS offered comments which helped shape JITS. The methodology used was a cyclic process reflecting the students’ comments: design, develop, test, modify, design, develop, test, etc.

The second objective set out in this dissertation was to determine the effectiveness of learning within this environment by comparing students taught Java in a traditional classroom environment with those exposed to JITS. The results from this quantitative component of the study are presented below. Each of the three Program Development Sessions contains three sections entitled: “JITS Developmental Research,” “JITS Performance Score Analysis,” and “Summary and Recommendations for further JITS Development.” Each section consists of one or more *parts* representing work conducted over several weeks.

#### **First Program Development Session: Section #1: JITS Developmental Research**

This session presents the findings from the first “live” test of JITS with students. It includes student and teacher comments and the researcher’s observations as students tried JITS. A significant amount of redesign and program refinement took place during this session which spanned from May through June, 2004.

***Session #1, Section #1, Part #1: May 3 to May 17, 2004***

This study was the first field test of JITS for students and instructors. During this study, a number of issues were raised. A number of students found that the hints generated by JITS were confusing. The researcher addressed this issue by introducing a number of new features to JITS.

The first feature added was a pointer (i.e., the caret character: “^”). This “pointer” was introduced to indicate the exact location of where an error has occurred in the student’s submission. For example, suppose the student submitted the following snippet as part of a solution:

```
for (int i=0; i<10 i++)
```

After the student clicks “Submit” button to send the code to JITS for analysis, JITS replies to the student with the following response:

**Suggestion:**

Look near line: 4 column: 37. Look between the "10" and the "i"

**Change:**

```
for (int i=0; i<10 i++)
```

**to:**

```
for (int i=0; i<10; i++)
```

The use of this pointer makes it easier for students to see exactly where the error is occurring. The researcher also added a “Suggestion” section to the reply that describes the exact location of where the error is taking place, the symbols used, and the corrected code.

Another issue raised during this part of the research was associated with the hints JITS generates. One teacher suggested more information should be provided in the hints that JITS offers. The researcher addressed this suggestion by creating and designing the



infrastructure for two buttons: “View Top Hint” and “View All Hints.” These two buttons provide different detail of information regarding the student’s submission. Depending on the student’s comfort level, s/he may select one over the other. The “View All Hints” is significant in the context of professional programmer’s Integrated Development Environments (IDE) in which such features are common. This is because in a professional environment, the compiler purposely flushes out all errors in the programmer’s code. It is not unreasonable for dozens of errors to be listed during compilation while a programmer is working on developing a solution to a problem. However, working with dozens of errors would be overwhelming for a beginner programmer. As discussed in Chapter 3 and 4, the philosophy behind JECA supports beginner programmers by focusing students on one specific error in the solution being developed. The “View Top Hint” is a human-computer interaction design feature aimed to support the student. Behind the scenes, JECA is performing the support for the “View Top Hint.”

The “View All Hints” is designed to act as an intermediary step to a more advanced level of competence as would be instilled in professional programmers. Figure 22 and Figure 23 depict examples of the functionality of these two buttons.

Another interesting issue was raised during this part of the research. One student discovered that they could simply type in the text stated in the “Required Output” section into the code section, submit it, and JITS would happily say, “Congratulations—you have solved the problem.” It was not long before all of the students learned this trick to outsmart JITS!

Java Intelligent Tutoring System - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://24.141.104.219:8968/JITS-Project1-context-root/jts.jsp> Go Links

Java Intelligent Tutoring System Welcome tra\_sem4\_student22!

NOTE: *You have tried this problem only once. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.*

**Problem: (1 of 4) in Problem Set # 4**  
 Write a program called Summer which adds all the integer numbers from 1 to a specified number (N). For example, if N were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

**Program Specifications:**  
 This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

**Required Output:**  
 Sum = 55

```
public class Summer {
    public static void main(String [] args) {
        int sum = 0;

        For (int i=0; i<10 i++)
            sum = smu + i;

        System.out.println("Sum = " + sum);
    }
}
```

Submit View Top Hint View All Hints View Solution My Performance New Problem Exit

OUTPUT:

Suggestion: Replace "For" with "for"

Figure 22. "View Top Hint" results. JITS selects the most significant hint to offer the student.

Java Intelligent Tutoring System

Welcome tra\_sam4\_student22!

NOTE: You have attempted this problem 2 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

Problem: (1 of 4) in Problem Set # 4  
Write a program called Summer which adds all the integer numbers from 1 to a specified number (N). For example, if N were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

Program Specifications:  
This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

Required Output:  
Sum = 55

```
public class Summer {
    public static void main(String [] args) {
        int sum = 0;

        For (Int i=0; i<10 i++)
            sum = smu + i;

        System.out.println("Sum = " + sum);
    }
}
```

Submit View Top Hint View All Hints View Solution My Performance New Problem Exit

OUTPUT:

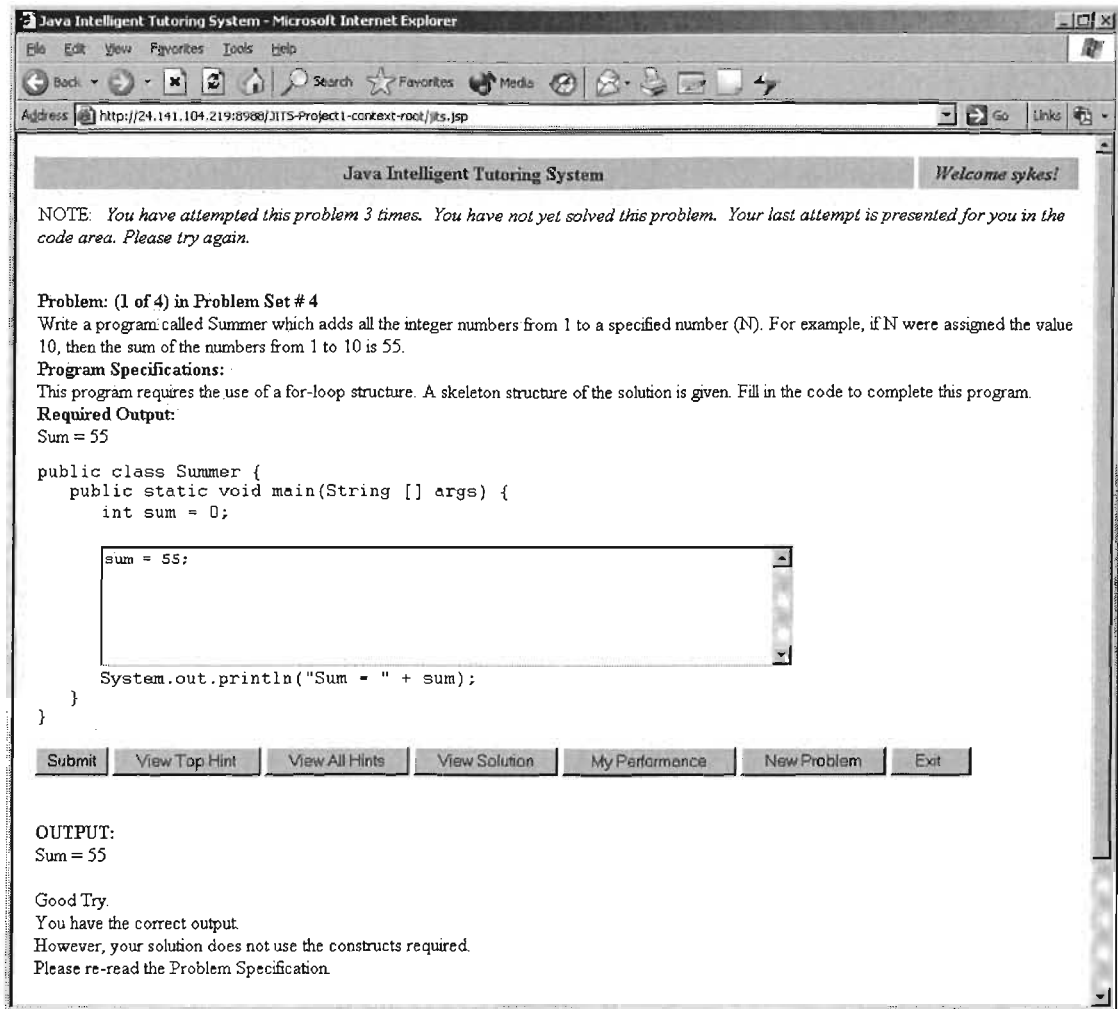
1. Keyword replacement hint:  
Suggestion: Replace "For" with "for"
2. Keyword replacement hint:  
Suggestion: Replace "Int" with "int"
3. Grammatical hint:  
Look near line: 4 column: 36. Look between the "10" and the "i"  
Suggestion:  
Change:  
for (int i=0; i<10 i++)  
to:  
for (int i=0; i<10; i++)

Figure 23. "View All Hints" results. JITS displays all of the hints relating to all of the problems JITS has encountered with the student's submission.

The problem with this is that a student could simply type in the final answer to a programming problem without using the required structures as specified in the outline of the problem description and specification. In order to address this problem, the researcher designed and implemented a solution that prohibits this from being admissible as a suitable solution. JITS recognizes that the student has entered the correct solution but has not solved the problem using the constructs as requested. A separate Artificial Intelligence (AI) module was introduced to solve this problem. Please see Figure 24 for a pictorial explanation of how JITS responds to such requests.

***Session #1, Section #1, Part #2: May 17 to May 31, 2004***

One teacher stated that he wanted a tracking system that would show various statistics about student performance and student activity. The teacher suggested that the information could be used for assessment purposes and later for instructional purposes. As a result, the researcher developed a solution entailing a redesign of the JITS ORACLE database schema in order to address the teacher's request. A report generation script was written in Structured Query Language (SQL) that tracks the students' activities in JITS and records them into the ORACLE database. The student tracking information currently includes the number of questions attempted, the number of times an attempt was made to answer a specific question, the current state of all student submissions for all of the questions, and many other facts about the current student and all other students using JITS. Table 9 and Table 10 depict some of the student tracking information.



Java Intelligent Tutoring System Welcome sykes!

NOTE: *You have attempted this problem 3 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.*

**Problem: (1 of 4) in Problem Set # 4**  
 Write a program called Summer which adds all the integer numbers from 1 to a specified number (N). For example, if N were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

**Program Specifications:**  
 This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

**Required Output:**  
 Sum = 55

```
public class Summer {
    public static void main(String [] args) {
        int sum = 0;

        sum = 55;

        System.out.println("Sum = " + sum);
    }
}
```

[Submit](#) [View Top Hint](#) [View All Hints](#) [View Solution](#) [My Performance](#) [New Problem](#) [Exit](#)

**OUTPUT:**  
 Sum = 55

Good Try.  
 You have the correct output.  
 However, your solution does not use the constructs required.  
 Please re-read the Problem Specification.

Figure 24. JITS analysis and response to a submission that is identical to the required output. JITS responds in the same manner as a human tutor would.

Table 9

*Sample Database Student Tracking Information Indicating Number of Attempts, Solved (true/false), and Student's Solutions*

STUDENT_NAME	PROBLEM_SET_ID	PROBLEM_ID	NUMBER_OF_ATTEMPTS	SOLVED	STUDENTS_SOLUTION
dav_sem3_student10	4	1	6	F	
dav_sem3_student16	4	1	8	T	for (int i = 1; i <= 10; i++) sum = sum + i;
dav_sem3_student16	4	2	0	F	
dav_sem3_student20	4	1	2	F	fdsf
dav_sem3_student3	4	1	3	T	for (int i = 0; i <= 10; i++) sum = sum + i;
dav_sem3_student3	4	2	1	T	for (int i = 4; i > 1; i--) fact = fact * i;
dav_sem3_student3	4	3	16	F	for (int i = 1; i = 500; i = i + 2) total = total + i;
dav_sem3_student5	4	1	0	F	
dav_sem3_student6	4	1	1	T	for (int i=1; i<=10; i++) sum += i;
dav_sem3_student6	4	2	1	T	for (int i=1; i<=4; i++) fact *=i;
dav_sem3_student6	4	3	10	F	for (int i=0; i<=500; i=i+1) total +=i-1;
dav_sem3_student7	4	1	1	T	for (int i=1; i<=10; i++) sum +=i;
dav_sem3_student7	4	2	1	T	for(int i=1; i<=4; i++) fact *=i;
e	4	1	4	T	for ( int i=0; i<=10; i++ ) sum = sum +i;

*(table continues)*

STUDENT_NAME	PROBLEM_SET_ID	PROBLEM_ID	NUMBER_OF_ATTEMPTS	SOLVED	STUDENTS_SOLUTION
e	4	2	5	T	for (int i=1; i<5; i++) fact = fact * i;
e	4	3	12	F	for (int i=0; i<10; i++) total = total + 0
e	4	4	6	F	for (int i=0; i<=1; i++) { total = total + start + end + last_fib_number; }
dav_sem3_student7	4	3	16	F	for (int i=1; i<=500; i +=2) total +=i;
dav_sem3_student7	4	4	12	F	for (int i = 1; i<=82; i++) total +=i + (i-1)+;
dav_sem3_student8	4	1	1	T	for (int i = 1; i <= 10; i++) sum = sum + i;
dav_sem3_student8	4	2	14	T	for (int i= 1; i <= 4; i++) fact = fact * i;
dav_sem3_student9	4	1	6	F	for (int i = 1, i < 11, i++) { sum = sum + i; }
dav_sem3_student9	4	2	5	T	for (int i = 4; i > 0; i--) { fact = fact * i; }
dav_sem3_student9	4	3	4	F	for (int i = 1; i < 500; i = i + 2) { total = total + i; }

---

Table 10

*Sample Database Student Tracking Information Indicating Current Problem Set, Problem\_id, Performance Rating, Skill level, Number of Times Connected to JITS, and the Date of Last Connection*

STUDENT_NAME	PROBLEM_SET_ID	PROBLEM_ID	SKILL_LEVEL	PERFORMANCE_RATING	PERFORMANCE	TIMES_CONNECTED	DATE_LAST_CONNECTION
e	4	1	1	81	null	12	Fri Jun 04 15:56:56
EDT 2004							
dav_sem3_student1	4	1	1	1		0	
dav_sem3_student10	4	1	1	1	null	2	Wed Jun 02 12:54:26
EDT 2004							
dav_sem3_student11	4	1	1	1		0	
dav_sem3_student12	4	1	1	1			0
dav_sem3_student13	4	1	1	1			0
dav_sem3_student14	4	1	1	1			0
dav_sem3_student15	4	1	1	1			0
dav_sem3_student16	4	1	1	1	null	1	Wed Jun 02 13:03:10
EDT 2004							
dav_sem3_student17	4	1	1	1			0
dav_sem3_student18	4	1	1	1			0
dav_sem3_student19	4	1	1	1			0
dav_sem3_student2	4	1	1	1			0
dav_sem3_student20	4	1	1	1	null	2	Fri Jun 04 15:54:40
EDT 2004							
dav_sem3_student21	4	1	1	1			0
dav_sem3_student22	4	1	1	1			0
dav_sem3_student23	4	1	1	1			0
dav_sem3_student24	4	1	1	1			0
dav_sem3_student25	4	1	1	1			0
dav_sem3_student26	4	1	1	1			0
dav_sem3_student27	4	1	1	1			0
dav_sem3_student28	4	1	1	1			0
dav_sem3_student29	4	1	1	1			0
dav_sem3_student3	4	1	1	83	null	1	Wed Jun 02 12:56:44
EDT 2004							
dav_sem3_student30	4	1	1	1			0
dav_sem3_student31	4	1	1	1			0
dav_sem3_student32	4	1	1	1			0
dav_sem3_student33	4	1	1	1			0

*(table continues)*



STUDENT_NAME	PROBLEM_SET_ID	PROBLEM_ID	SKILL_LEVEL	PERFORMANCE_RATING	PERFORMANCE	TIMES_CONNECTED	DATE_LAST_CONNECTION
dav_sem3_student34	4	1	1	1			0
dav_sem3_student35	4	1	1	1			0
dav_sem3_student36	4	1	1	1			0
dav_sem3_student37	4	1	1	1			0
dav_sem3_student38	4	1	1	1			0
dav_sem3_student39	4	1	1	1			0
dav_sem3_student4	4	1	1	1			0
dav_sem3_student40	4	1	1	1			0
dav_sem3_student5 EDT 2004	4	1	1	1 null		1	Wed Jun 02 12:53:27
dav_sem3_student6 EDT 2004	4	1	1	81 null		2	Wed Jun 02 12:52:45
dav_sem3_student7 EDT 2004	4	1	1	81 null		3	Wed Jun 02 12:52:31
dav_sem3_student8 EDT 2004	4	1	1	81 null		2	Wed Jun 02 13:02:53
dav_sem3_student9 EDT 2004	4	1	1	81 null		1	Wed Jun 02 12:51:58

Some of the teachers involved in the JITS research project wanted a web-based means to create problems for JITS to use for students. An authoring system is a tool that allows authorized people (i.e., a teacher) to be able to create, modify, and delete problems that are used in an Intelligent Tutoring System. The authoring tool the researcher is currently developing will enable teachers to work with problems easily and quickly within JITS. Teachers want these problems to be immediately available for JITS to use for students. Most of the teachers stated that they wanted access to JITS from anywhere and everywhere. Some teachers stated that they wanted to enter only the minimum amount of information required for JITS to do its job. In other words, they did not want to spend too much time in the development and typing of new problems in JITS.

The design and development of the JITS Authoring Tool was initiated during this period of time in the project. Approximately 150 hours of programming has already been done on the authoring tool, and work is still in progress. The goal of this tool is to provide the teacher a convenient means to add problems to the database for JITS to use. This will enable teachers to easily manipulate JITS programming problems because the teacher needs to provide only the following information:

1. the problem statement;
2. the problem description;
3. the required output; and
4. the skeleton structure of the program.

As a result, the JITS Authoring Tool is intended to be extremely user friendly in order to add many problems of various levels of difficulty. Once the teacher has

submitted the problems, they are immediately available to JITS and thus to students of the system. The JITS Authoring Tool User Interface is shown in Figure 25.

The JITS Authoring Tool provides a means for the instructor to view all the problems in the lesson set and edit selected problems. In the Java Intelligent Tutoring System, the author of problems does not provide a solution. JITS carefully scrutinizes the student's submission based on the problem description, specification, required output, and template code and determines the appropriate feedback for the student. This ensures the greatest degree of independent knowledge creation for each student.

***Session #1, Section #1, Part #3: Week of May 31, 2004***

In this part of the JITS Developmental Research section of the First Program Development Session, a number of students stated that they wanted to be able to see the solution after a certain number of attempts at a problem or if they got frustrated. Essentially, they said, "It would be nice for solution button to be available." As a result, the researcher designed and developed a "View Solution" button with supporting infrastructure. In the Java Intelligent Tutoring System, teachers are not required to submit solutions during problem authoring. This is based on the premise that given virtually all programming problems, there are potentially limitless solutions. Supplying only one solution for a given programming problem is not an acceptable approach. As a result, a `Collective_Student_Model` representing the sum knowledge of all students was designed and developed. This `Collective_Student_Model` analyzes all the students' submissions and extracts those that are solutions to the particular problem the student is currently working on.

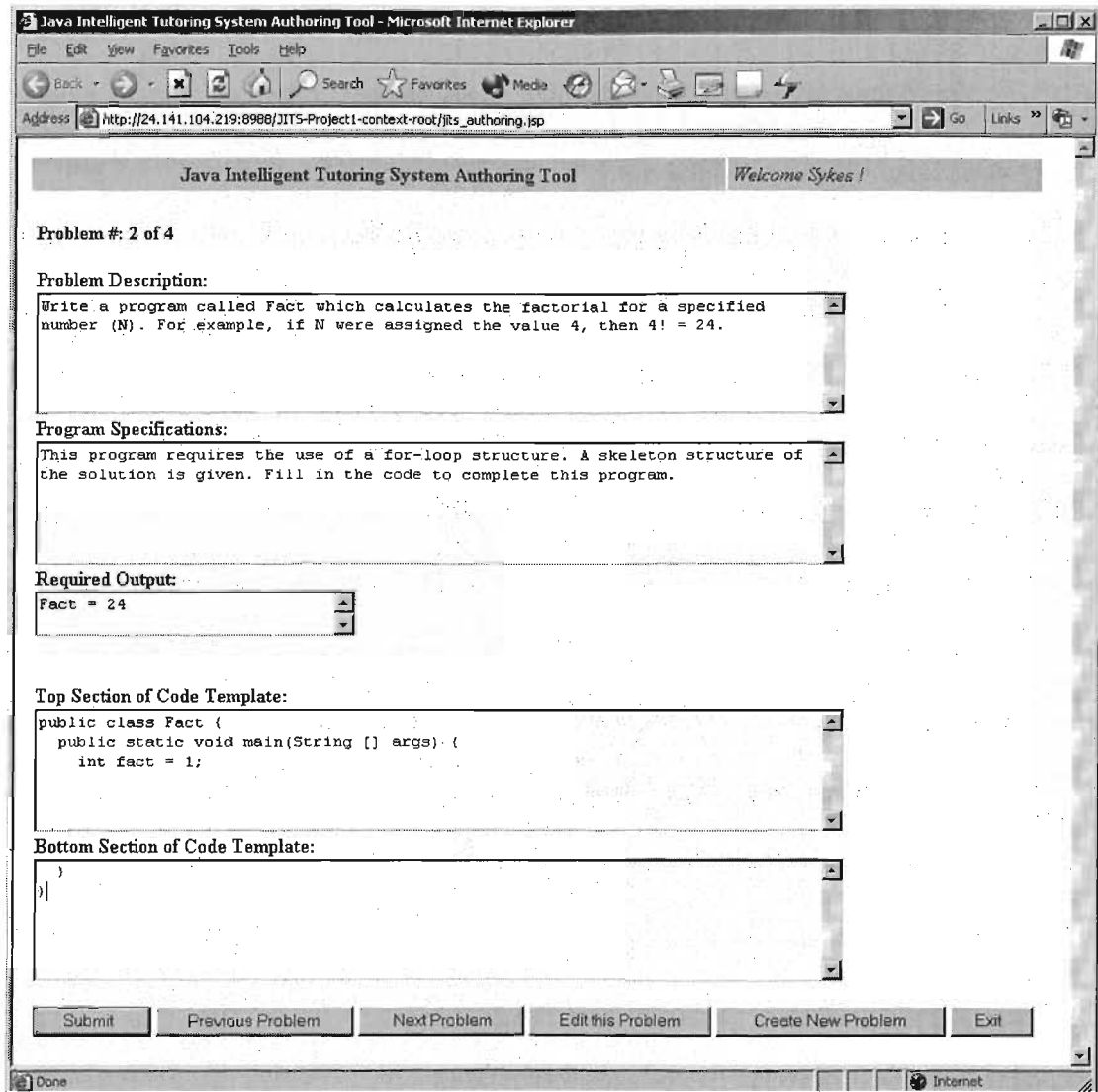


Figure 25. JITS Authoring Tool User Interface.

The AI\_Module uses the information in Collective\_Student\_Model to determine appropriate feedback. The revised JITS user interface is shown in Figure 26, with a small example illustrating this additional functionality due to student suggestions.

***Session #1, Section #1, Part #4: June 7 to June 17, 2004***

One student suggested a list of different programming topics should be listed on the side of JITS User Interface page. The student suggested that it would make JITS more useful for students with varying levels of Java expertise and/or interest and add a significant degree of professional “look” to JITS. The student suggested that “students want to have a choice over their own learning—that is, students want to be able to select the area of study they are interested in (e.g., Java arrays, loops, etc.).” Addressing this problem required a great deal of work. In order to fulfill the request, a complete redesign of the following JITS components was necessary: the ORACLE database schema, the User Interface, and a number of JITS’ internal infrastructure components. In total, the researcher spent several hundred hours on these tasks. The redesigned JITS now contains the following programming topics:

1. Java Basics;
2. Java Statements;
3. The “if” statement;
4. The “for” loop;
5. The “do–while” loop;
6. The “while” loop; and
7. Arrays in Java.

Java Intelligent Tutoring System - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://24.141.104.219:8968/JITS-Project1-context-root/js.jsp> Go Links

Java Intelligent Tutoring System Welcome  
ira\_sem3\_student20!

NOTE: You have tried this problem only once. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

**Problem: (1 of 4) in Problem Set # 4**  
Write a program called Summer which adds all the integer numbers from 1 to a specified number (N). For example, if N were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

**Program Specifications:**  
This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

**Required Output:**  
Sum = 55

```
public class Summer {
    public static void main(String [] args) {
        int sum = 0;

```

```
        System.out.println("Sum = " + sum);
    }
}
```

**OUTPUT:**

Below are possible solutions to this problem.

**Solution 1:**  

```
for ( int i=0; i<=10; i++ )
    sum = sum +i;
```

**Solution 2:**  

```
for (int i=1; i<=10; i++)
sum +=i;
```

**Solution 3:**  

```
for (int i=1; i<=10; i++)
sum += i;
```

**Solution 4:**  

```
for (int i = 1; i <= 10; i++)
    sum = sum + i;
```

Done Internet

Figure 26. "View Solution" presenting solutions for the current problem.

Each Problem Set was at least 2 problems. The redesigned JITS User Interface is presented in Figure 27. With the addition of programming topics, new fields were needed in the tables for the JITS ORACLE schema to track student performance more accurately. Two new tables were added to provide separate learning topics. Table 11 and Table 12 depict the schema structure of the tables that are currently used by JITS. An extensive degree of redesigning and redeveloping was necessary to create a list of “Programming Topics” within JITS. Additional objects were redesigned and rebuilt including: the Student\_Model, the Problem, the JITS User Interface. Figure 28 depicts JITS’ abstract internal object representation. Further modifications were necessary on the User Interface due to technical problems. As a result, the screen shot depicted in Figure 29 shows the revised User Interface developed during this time. Notice the difference in the “Programming Topics” list.

***Session #1, Section #1, Part #5: Week of June 17, 2004***

During this part of the JITS Developmental Research for the First Program Development Session, a number of interesting issues were raised. One student suggested that she would like to be able to navigate both forwards and backwards through the problem sets. The current JITS system allowed only forward movement through the problems to encourage incremental skill development by presenting increasingly more difficult problems to the student. However, the researcher decided that allowing the student full control over the movement through the problem sets has merits. The researcher designed the infrastructure for the “Previous Problem” and tested it out. Figure 30 depicts the newly designed JITS User Interface including the “Previous Problem” and “Next Problem” buttons.

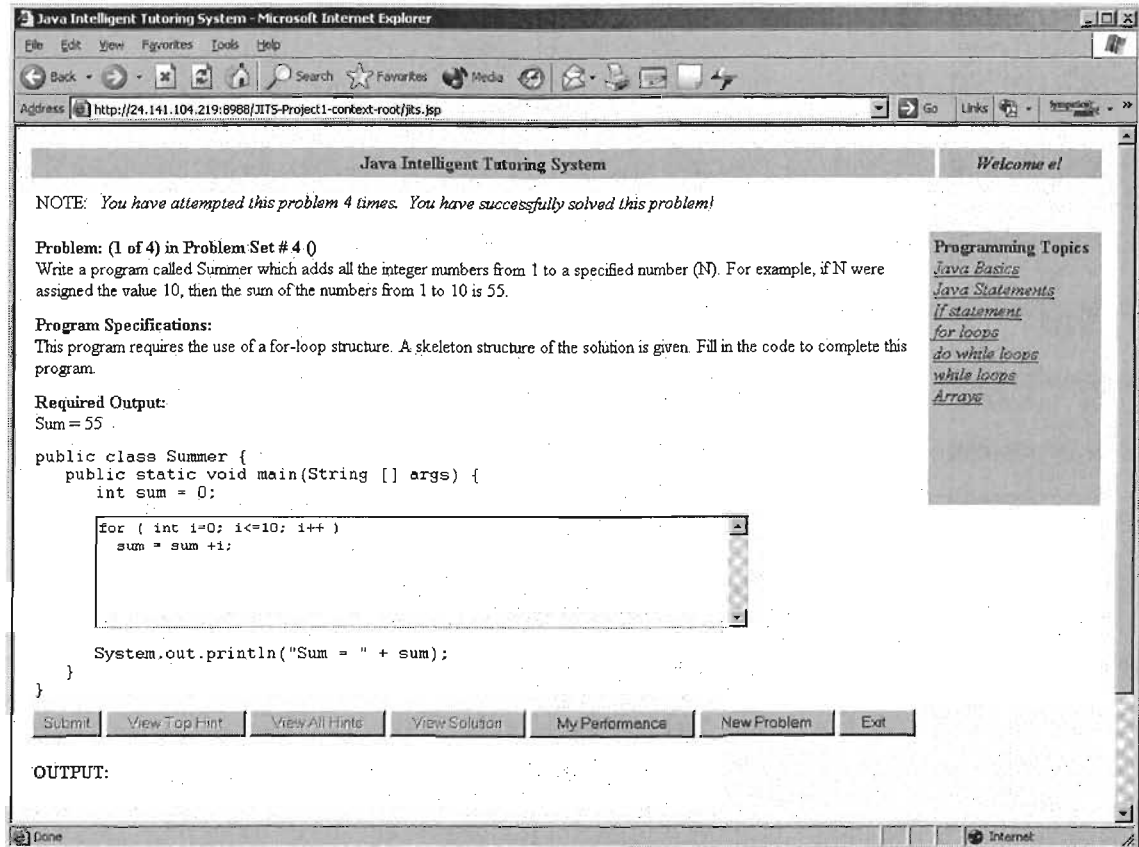


Figure 27. Redesigned JITS User Interface incorporating Programming Topic selection panel.



Table 11

*Redesigned JITS ORACLE Schema Tables*


---

```

CREATE TABLE PROBLEM_SETS (
    problem_set_id    NUMBER(3),
    problem_set_title VARCHAR2(30),
    problem_set_desc  VARCHAR2(400),
);

CREATE TABLE PROBLEMS (
    problem_set_id    NUMBER(3),
    problem_id        NUMBER(3),
    problem_desc      VARCHAR2(400) NOT NULL,
    problem_spec      VARCHAR2(400) NOT NULL,
    problem_output    VARCHAR2(50),
    template_top_section VARCHAR2(400),
    template_bottom_section VARCHAR2(400),
    problem_difficulty VARCHAR2(20),
    problem_keywords  VARCHAR2(200),
);

CREATE TABLE STUDENTS (
    student_name      VARCHAR2(30),
    student_password  VARCHAR2(15),
    problem_set_id    NUMBER(3),
    problem_id        NUMBER(3),
    skill_level       NUMBER(3),
    performance_rating NUMBER(3),
    performance_history VARCHAR2(2000),
    times_connected   NUMBER(5),
    date_last_connection VARCHAR2(30),
    picture           LONG RAW,
);

CREATE TABLE STUDENT_PROBLEMS (
    student_name      VARCHAR2(30),
    problem_set_id    NUMBER(3),
    problem_id        NUMBER(3),
    number_of_attempts NUMBER(3),
    solved            CHAR(1),
    students_solution VARCHAR2(500),
    solution_date     VARCHAR2(30),
);

```

---

Table 12

*Redesigned JITS ORACLE Schema Showing the Newly Created Programming Topics and Corresponding Descriptions*

---

PROB_ID	PROBLEM_SET_TITLE	PROBLEM_SET_DESC
1	Java Basics	variables, variable names, numeric variable types, String type, variable declarations
2	Java Statements	simple and complex expressions, compound statements, operators, precedence
3	If statement	simple and complex if statement problems
4	for loops	problems requiring the use of for loops
5	do while loops	problems requiring the use of the do while loop
6	while loops	problems requiring the use of while loops
7	Arrays	single dimension array problems

---

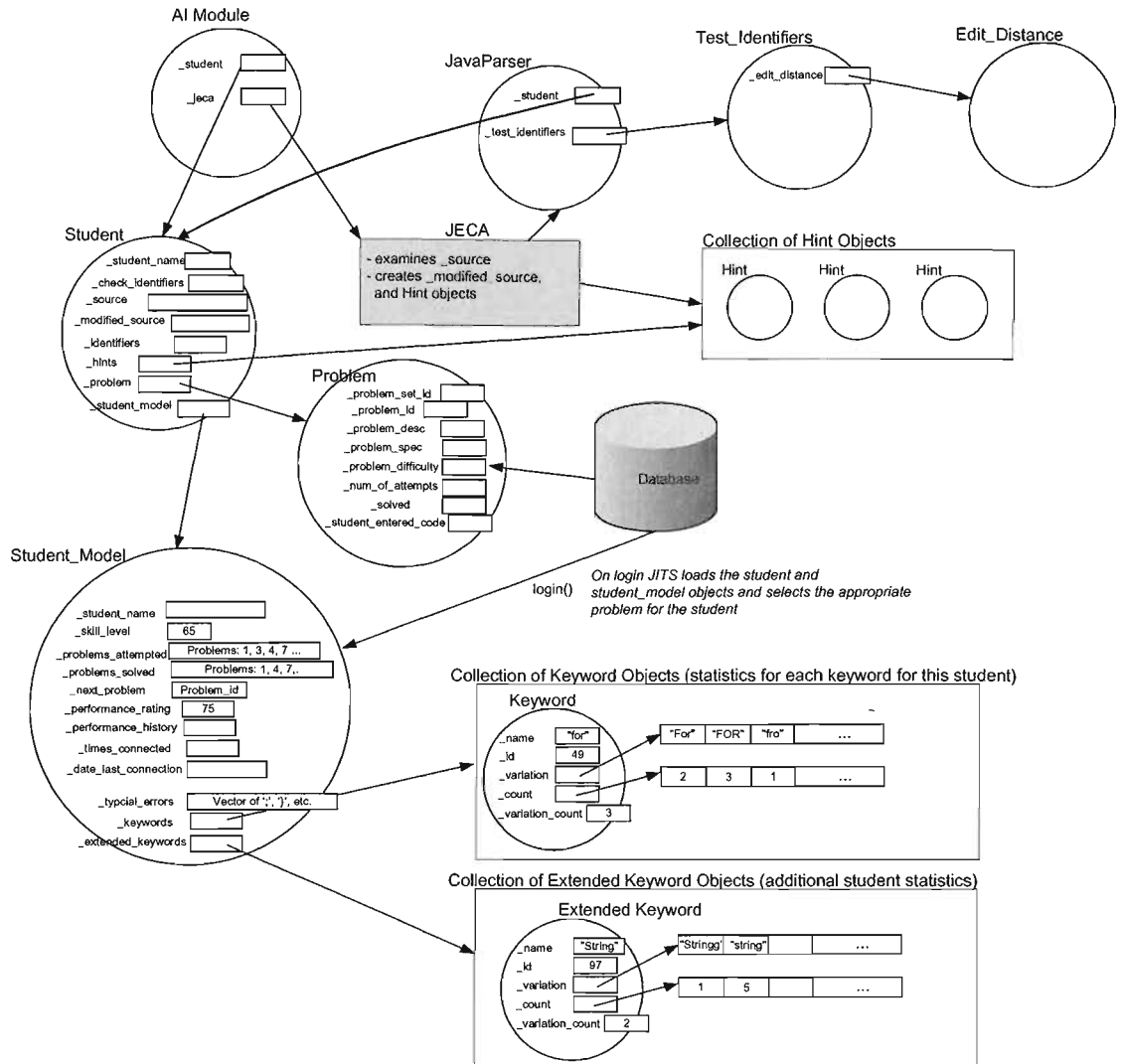


Figure 28. JITS abstract internal object representation showing relationships and dependencies between JECA, AI\_Module, student, and other components.

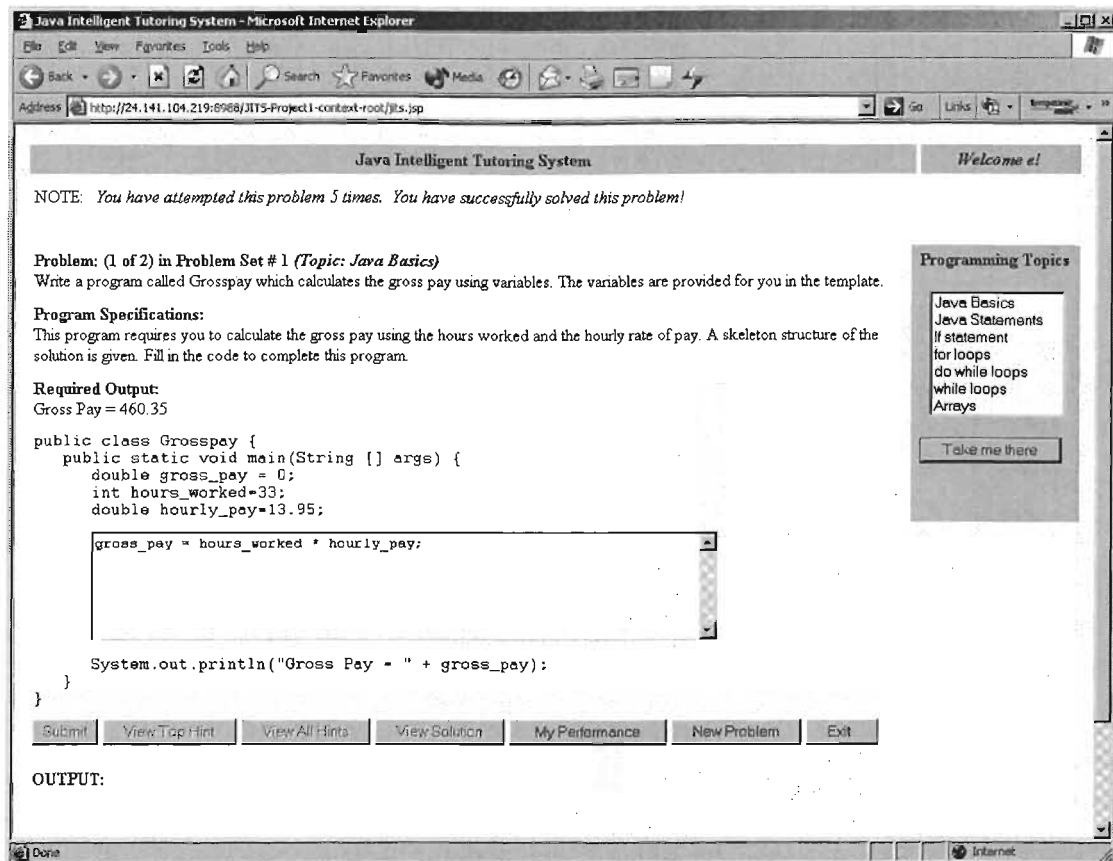


Figure 29. Redesigned JITS User Interface depicting the list of Programming topics in a drop-down combo list.

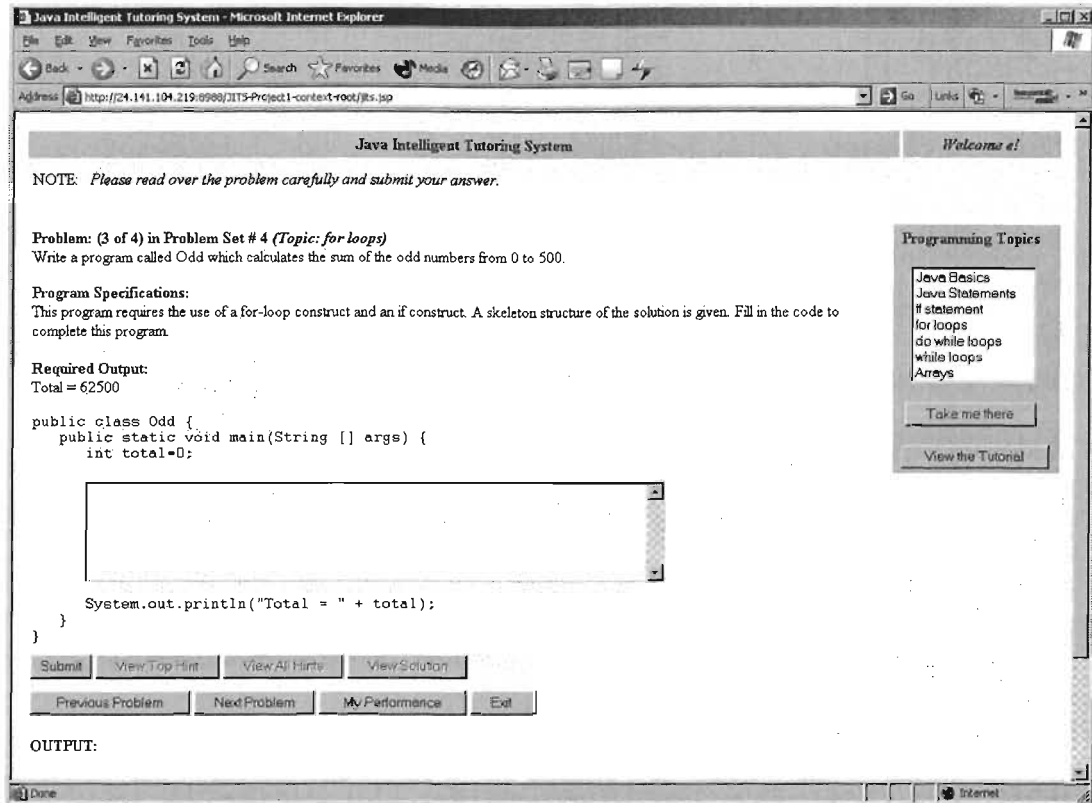


Figure 30. Resigned JITS User Interface depicting the “Previous Problem” and “Next Problem” buttons.

An additional interesting issue was the fact that JITS did not retain information about the exact state a student was in between login sessions. In other words, if a student logged into JITS, attempted to solve a problem and then logoff, JITS would not retain the partial solution. As a result, students had to retype the code that was lost and get back into the same mental state again to attempt to solve the problem that they were last working on. If a student changed from the “For loop” topic to the “While loop” topic, attempted to solve problem 3 of the “While loop” topic, and then logoff, JITS should bring the student back to the exact location where s/he left off.

The researcher designed and developed a solution to this problem. JITS now maintains the exact student state between logins and between all transitions a student may make in JITS. The researcher felt it was important to ensure that the student continues developing skills and knowledge from the point where the student was last working to minimize the loss of cognitive gap between JITS sessions and to reduce the amount of frustration a student may experience when learning to program.

Another issue raised during this study was associated with the “View” buttons (i.e., “View Top Hint,” “View All Hints,” and “View Solution”). There were some problems with the user interface design. For instance, at certain times these buttons should be disabled, for example, at the beginning of a new problem. When the student is in a different state, JITS should enable some or all of these buttons. As a result, the researcher investigated the various states that JITS offered and redesigned and developed more suitable human-computer interface properties for these buttons.

The last issue raised during this part of the research showed how finicky JITS could be in some situations. The researcher became aware of a problem in JITS

from a student who demonstrated the problem to the researcher. JITS encountered a rounding error and did not correctly identify a student's correct response unless the program output was accurate to an extreme level of decimal precision. For example, if a student's submission produced an answer such as, "Gross Pay = 460.349999999" and the required output, as described in the problem statement section stated: "Gross Pay = 460.35," then JITS would not accept it as a correct answer. The researcher identified that JITS was being too particular in the degree of accuracy in the required output of a student's solution. The researcher designed and constructed a solution to the problem that involved a number of renovations to the internal design of JITS. The researcher worked on the AI\_Module object and defined a "suitably\_close" function which checks these sorts of situations and returns "true" indicating that it is close enough to be considered a correct solution to the problem.

***Session #1, Section #1, Part #6: June 21 to June 29, 2004***

During this study, students mentioned they would be interested in a small tutorial section to be incorporated into JITS. This tutorial section would explain each of the different programming topics designed and developed earlier in JITS. The purpose of the JITS tutorial would be to remind students about basic problem-solving strategies, basic syntax associated with specific Java constructs (e.g., if statements, for-loops, etc.). This issue required approximately 100 hours of work by the researcher. The researcher needed to do a significant amount of research on browser pop-ups and JavaScript. However, after much labour, the embedded tutorial was completed. Figure 31 shows the activation of the JITS Tutoring pop-up window from within the JITS User Interface. Figure 32 depicts contents of the tutorial window for the "for" programming statement.

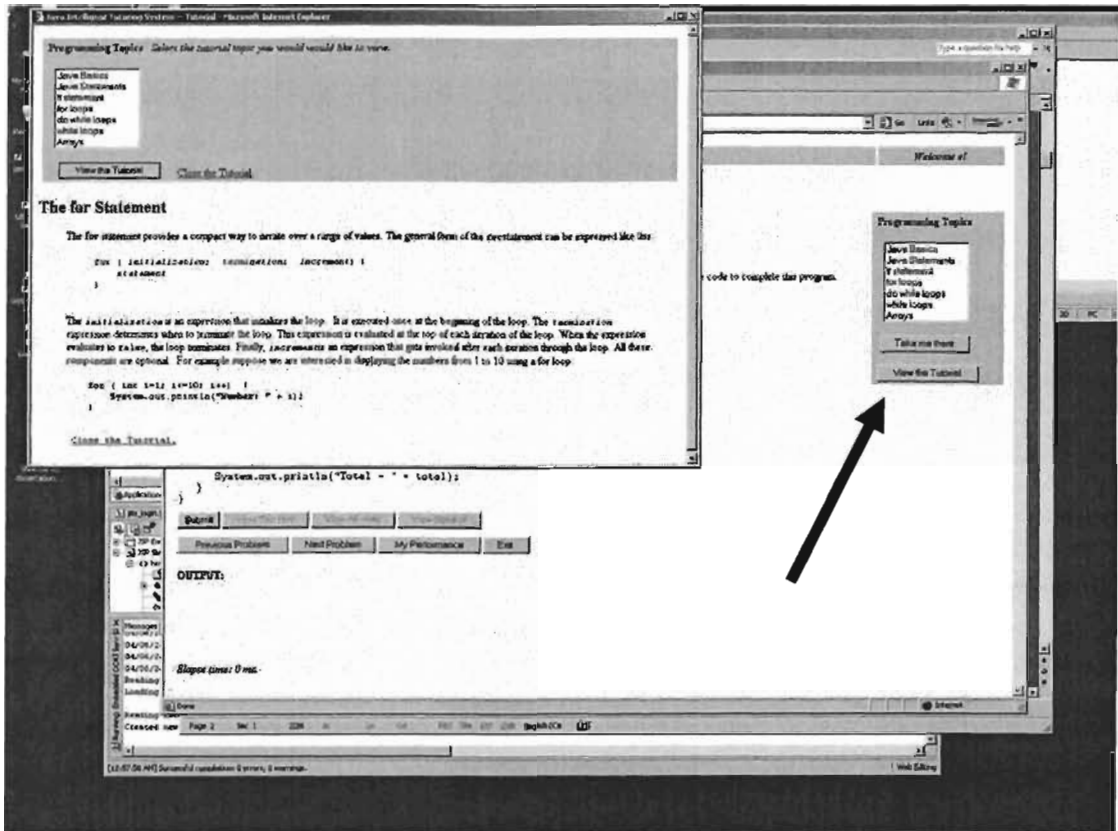


Figure 31. JITS Tutorial window and main JITS User Interface. The tutorial window is launched from the main JITS User Interface by clicking the “View Tutorial” button as indicated by the arrow.



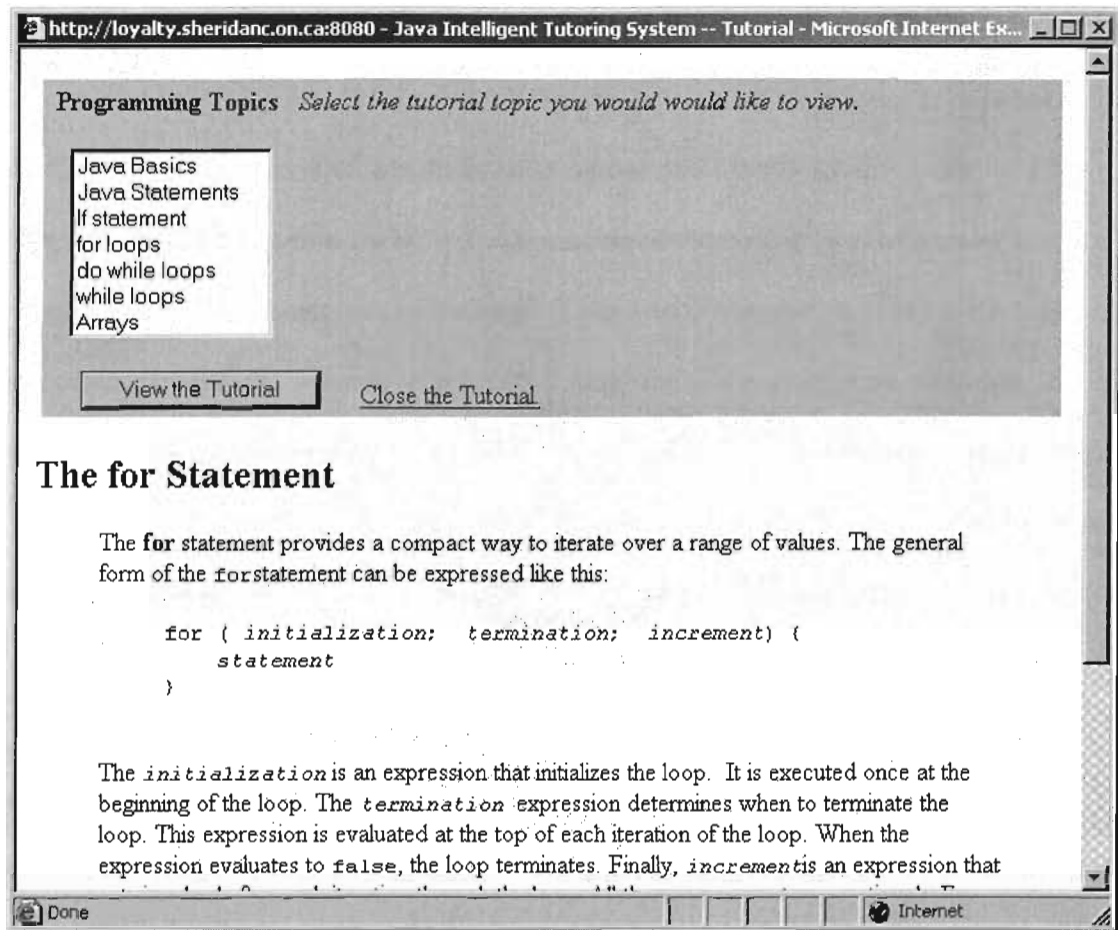


Figure 32. JITS Tutorial window displaying a sample tutorial from the list of Programming Topics.

### **First Program Development Session: Section #2: JITS Performance Score Analysis**

The following presents the First Program Development Session results of the Performance Scores for students in the JITSC group and Control groups. The dates for this section of the research started at the beginning of the term in May and ended 7 weeks later in July. Table 13 and Table 14 display the performance scores of students in classes C and JITSC. Table 15 presents a summary of the descriptive statistical findings on the performance scores in Table 13 and Table 14. In order to determine the relationship between the performance scores in C and JITSC for the first 7 weeks of this course, a two-way ANOVA with repeated measures was conducted. Table 16 show these results.

A two-way ANOVA with repeated measures was conducted, producing the following results,  $F(1,35) = 541.645, p = .459$ , indicating there was no significant difference between the two groups (i.e., C and JITSC). Table 16 shows the results from the ANOVA.

### **First Program Development Session: Section #3: Summary and Recommendations for Further JITS Development**

This section presents a summary and recommendations for the refinement of JITS for the first program development session. The summary presents the results of the performance score analysis, and student and professor perspectives. The recommendations section discusses areas for improvement in the Java Intelligent Tutoring System.

Table 13

*Performance of Students in Class JITSC*

---

**JITSC Class**

<b>JITSC Class</b>		
<b>Student</b>	<b>Pretest (%)</b>	<b>Posttest (%)</b>
S <sub>1</sub>	35.00	100.00
S <sub>2</sub>	35.00	92.86
S <sub>3</sub>	55.00	61.43
S <sub>4</sub>	47.50	80.00
S <sub>5</sub>	25.00	54.29
S <sub>6</sub>	80.00	100.00
S <sub>7</sub>	60.00	97.14
S <sub>8</sub>	30.00	88.57
S <sub>9</sub>	57.50	100.00
S <sub>10</sub>	65.00	100.00
S <sub>11</sub>	85.00	100.00
S <sub>12</sub>	25.00	54.29
S <sub>13</sub>	55.00	100.00
S <sub>14</sub>	62.50	100.00

---

Table 14

*Performance of Students in Class C*

Class C

---

<b>Student</b>	<b>Pretest (%)</b>	<b>Posttest (%)</b>
C <sub>1</sub>	50.00	97.14
C <sub>2</sub>	50.00	57.14
C <sub>3</sub>	60.00	100.00
C <sub>4</sub>	50.00	95.71
C <sub>5</sub>	55.00	41.43
C <sub>6</sub>	40.00	72.86
C <sub>7</sub>	30.00	85.71
C <sub>8</sub>	35.00	87.14
C <sub>9</sub>	67.50	88.57
C <sub>10</sub>	45.00	68.57
C <sub>11</sub>	35.00	91.43
C <sub>12</sub>	75.00	87.14
C <sub>13</sub>	72.50	87.14
C <sub>14</sub>	35.00	28.57
C <sub>15</sub>	30.00	78.57
C <sub>16</sub>	65.00	94.29
C <sub>17</sub>	30.00	98.57
C <sub>18</sub>	77.50	100.00
C <sub>19</sub>	25.00	37.14
C <sub>20</sub>	25.00	68.57
C <sub>21</sub>	75.00	94.29
C <sub>22</sub>	65.00	92.86
C <sub>23</sub>	62.50	97.14

---

Table 15

*Standard Statistical Measures for C and JITSC*

---

<b>Group</b>	<b>Pretest</b> mean and ( <i>standard deviation</i> )	<b>Posttest</b> mean and ( <i>standard deviation</i> )
C	50.217 (17.579)	80.433 (21.046)
JITSC	51.250 (19.209)	87.756 (17.882)

---

Table 16

*Two-way ANOVA with Repeated Measure: Between-Subjects Effects for C and JITSC*

---

Source	Type IV sum of squares	<i>df</i>	Mean square	<i>F</i>	Sig.
Intercept	316408.112	1	316408.112	584.161	.000
Group	303.708	1	303.708	.561	.459
Error	18957.576	35	541.645		

---

### *Summary*

Figure 33 shows a pictorial summary of performance scores between C and JITCS using the means as the data. Although there appears to be a visual difference between the two groups, the standard deviation was so large that there was no statistical differentiation at any level of significance. (See Table 13, Table 14, and Table 15 for specific results.)

However, from a qualitative perspective, the results from the survey administered to the JITSC group show generally positive feelings towards the Java Intelligent Tutoring System (Please see Table 7 for the interview survey). Overall, students appeared to enjoy the Java Intelligent Tutoring System and found it beneficial and friendly to use. The researcher took the raw data from the interview surveys and computed basic statistics as shown in Table 17. Please see Table 7 for the interview survey. The survey consisted of a Likert 5-point scale for each of the six questions. It can be seen that JITS performed above “average” in all categories and scored the highest in two categories: “Enjoyable” and “Ease of JITS Tutoring Style” (The term “average” in this context refers to the third item on the 5-point scale on the survey).

The findings also revealed important issues regarding how to improve JITS. The comments are discussed below and will be reviewed for potential inclusion in subsequent versions of JITS. The comments presented were gathered from two perspectives: student’s and professor’s.

### JITSC versus C Performance Score Comparison Using Pretest and Posttest Means as Data

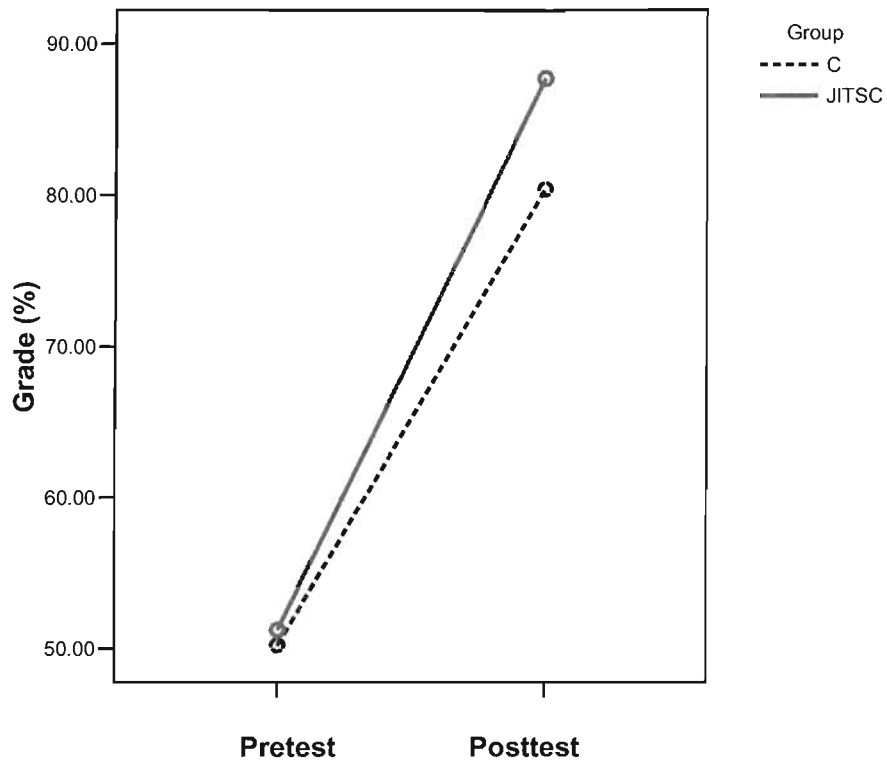


Figure 33. JITSC versus C performance comparison using pretest and posttest means as data.



Table 17

*JITS Qualitative Summary Results for JITSC Students*

---

1. Usefulness.....	71%
2. Beneficial .....	64%
3. JITS is better than a traditional classroom.....	36%
4. Ease of JITS Tutoring Style.....	79%
5. Enjoyable.....	79%
6. Learn Better.....	71%

---

***The Students' Perspective.*** Most of the students enjoyed working with JITS, and a number of students identified the following features as definite strengths of JITS:

1. Feedback mechanism—It provides hints quickly and they are to the point.  
The hints are also not overwhelmingly complicated, and this is quite unlike traditional programming environments and compilers.
2. One student stated, “[JITS] tells me the exact spot in the code where I need to make my correction—I like that. I wish other systems would do that.”
3. JITS helps students solve syntax and logic errors while developing a solution to a problem. One student stated, “I like learning in this environment better than having the instructor demonstrate how to do something.”
4. Several students stated that they like the User Interface of JITS. They said that it is similar to other Integrated Development Environments which are designed for professional programmers.
5. Many students stated that they felt JITS was very useful since it is available at all times and students need only a browser to use JITS.
6. One student said, “Can we have this system in our course from now on?”

Regardless of the apparent success of JITS, this study showed that the suggestions from students helped in making JITS more beneficial for their education. Students felt that the hints were extremely good when the programming error was a syntax error. The Java Error Correction Algorithm (JECA), for the most part, was able to determine the intent of the student and offer meaningful and helpful corrective feedback. However, some students suggested that it would be even more beneficial if JITS could offer help in situations where there was logical mistake in a student's

solution. For instance, given the following submission to calculate the sum of the numbers from 1 to 10:

```
for (int i=0; i<10; i++)
    sum = sum + i;
```

The submission is syntactically correct; however, there is a logic error. JITS would respond as follows:

Sum = 45

Nice Try.

However, there may be a logic error in your program. Take a look at your formula. If you are using a loop check the range of values for the beginning and ending of your loop.

As a result, the student needs to reexamine the submission and try to determine the logic error. In this example, the problem is that the loop is not incrementing far enough. Two correct solutions are provided below:

**Solution #1:**

```
for (int i=0; i<=10; i++)
    sum = sum + i;
```

**Solution #2:**

```
for (int i=0; i<11; i++)
    sum = sum + i;
```

The researcher made note of the need to support students making logic errors, and in later versions of JITS, more detailed hints are generated. Overall, however, the students seemed quite happy with the prototype of JITS. They all seemed eager to see and try out future versions of the Java Intelligent Tutoring System.

*The Professors' Perspective.* The section summarizes the views of professors involved in this study. Two of the professors said they were pleased with JITS in the following ways:

1. One professor stated, "The embedded logic unit called JECA is a sound tool—it picks out the most significant error the student needs to focus on. I feel the student is developing core programming debugging skills with JITS."
2. One professor said the idea behind JITS' User Interface is similar to popular Integrated Development Environments, which should make the transition from JITS to a professional programming environment a little easier.
3. Both of the professors said that they would like to use JITS to augment their existing Java courses. They felt that JITS provided a means for students to receive extra tutoring when the professor is not available.
4. One professor said, "The quality of tutoring that JITS performs is comparable to a human tutor."
5. Both of the professors said that they liked the fact that there was no client installation required for them or their students (i.e., there is no software to install on the client's computer).
6. Many professors were happy that JITS was available at all times. This made it easier for students to work on problems at their own time and at their own pace.
7. One professor said, "JITS provides additional programming practice that can only benefit students."

Although there was no significant level of differentiation between the Control group and JITSC group for performance scores, there are many explanations for this. First, JITS was undergoing extensive redesign and redevelopment during the first 7 weeks of the course in which JITS was tested. Second, there may not have been a suitable number of problems embedded in JITS at the time of testing. For instance, initially, there were only 4 programming problems in JITS. Third, students may not have used JITS enough to reach a higher level of performance. In other words, learning a new software tool requires some initial loss in productivity from the way in which one customarily does things. It will be interesting to note students' results in future field studies now that most of the "bugs" have been worked out of JITS. However, getting over this small hurdle gives way to certain potentials. In this case it may give rise to the student performing better by increasing their knowledge and skill set in programming.

### ***Recommendations***

This section presents recommendations for the refinement of JITS for the first program development session. The recommendations include student and professor comments for improvement in the Java Intelligent Tutoring System.

***Student Perspective.*** A number of the students raised a concern about the "My Performance" button's output. A sample output after pressing the "My Performance" button is shown in Figure 34. Students felt that a more detailed representation of their performance could be helpful. Currently, JITS takes a number of factors into account when computing the student's performance. JITS correlates the student's skill level with the problem difficulty and ranks the points accordingly. JITS determines the number of problems solved against the number of problems attempted in combination.

**OUTPUT:**  
**My Performance:**

Problems Attempted: 4  
Problems Solved: 2  
Overall Performance: 72

*Figure 34.* Initial design of the output from the “My Performance” button.

Furthermore, JITS combines this information with other gathered facts about the student as represented in the established student model. See Figure 28 for a depiction of the student model and related modules. As can be seen from the diagram, there is a tremendous amount of information available for analysis regarding the student's performance. Future versions of JITS will provide much more detail regarding student's performance.

***Professor Perspective.*** One professor suggested that JITS could produce a report representing the student's performance over a period of time. This would also be helpful in identifying students who need additional assistance. It could also be used to identify those students who are doing extremely well and may be interested in more challenging problems.

All of the professors enjoyed the prototype of the JITS Authoring tool. Although still under development, the prototype made professors aware that they can easily create, edit, and review problems. Once the problems have been added, they are immediately available to the students. Currently, the researcher is busy working on designing and developing the JITS Authoring Tool.

### **Second Program Development Session: Section #1: JITS Developmental Research**

During this session, JITS underwent a number of design changes to both internal infrastructure and the User Interface. The changes were based primarily on the suggestions of students from the First Program Development Session. This section presents the findings of experiments that were conducted during the last half of the summer term (i.e., July 1 to Aug 20, 2004). The control group and the experimental group consisted of the same students as the First Program Development Session.

***Session #2, Section #1, Part #1: July 1 to July 14, 2004***

Two students completed all of the programming problems in JITS and suggested that more problems be created. As a result, the researcher spent this time reviewing suitable material that would be fitting to the curriculum and the level of students in the JITS.

At the completion of this study JITS offered at least four problems for each of the seven Problem Sets: *Java Basics, Java Statements, If statement, for loops, do while loops, while loops, and Arrays*. During this period of time, the researcher also tested each of the new problems as rigorously as possible by simulating various errors. The students seemed to enjoy having more choice. With the full navigation in JITS fully functional, students could navigate from problem set to problem set and problem to problem to find areas that were of interest to them.

***Session #2, Section #1, Part #2: July 14 to Aug 11, 2004***

One student mentioned that the “Solutions” button did not display information correctly. The problem was with the display of solutions in the student’s browser. For example, solutions like:

```
for (int i=0; i<arr.length; i++)  
    sum = sum + arr[i];
```

were showing up as:

```
for (int i=0; i sum = sum + arr[i];
```



which is clearly incorrect. A number of important elements were not being displayed (i.e., the “`<arr.length; i++`”) was being omitted)

This problem took approximately 4 hours for the researcher to determine the root cause of problem. The researcher checked the database and confirmed that the correct solution was being recorded in the tables of the database. The researcher then investigated the “Student” class and its method: `Store_State_to_database()`, and the “Collective Student\_Model” class and its `find_solutions()` method. Apparently, the problem is exclusive to the browser’s interpretation in HTML of the “<” sign. Browsers were confusing it as a markup symbol as opposed to simply leaving it as an ordinary character as desired. The researcher corrected the problem by using the appropriate HTML tags instead of using “<”. A reference that assisted me was:

<http://www.w3.org/MarkUp/html3/latin1.html>

which provided the following information:

<i>lt</i>	<code>&amp;lt;</code>	<i>Less than sign</i>
<i>gt</i>	<code>&amp;gt;</code>	<i>Greater than sign</i>
<i>amp</i>	<code>&amp;amp;</code>	<i>Ampersand</i>
<i>quot</i>	<code>&amp;quot;</code>	<i>Double quote sign</i>

The researcher tested and confirmed the correct behaviour using various scenarios and test accounts. The “Solution” was now producing correct results even when specific symbols were used in the solution.

## **Second Program Development Session: Section #2: JITS Performance Score**

### **Analysis**

Table 18 and Table 19 display the performance scores of students in the control group (C) and the experimental group (JITSC). Table 20 presents a summary of the descriptive statistical findings on the performance scores in Table 18 and Table 19. In order to determine the relationship between the performance scores in C and JITSC for the last 7 weeks of this course, a two-way ANOVA with repeated measures were conducted.

### **Second Program Development Session: Section #3: Summary and Recommendations for Further JITS Development**

This section presents a summary and recommendations for the refinement of JITS for the second program development session. The summary presents the results of the performance score analysis, and student and professor perspectives. The recommendations section discusses areas for improvement in the Java Intelligent Tutoring System.

#### ***Summary***

In this Program Development Session, the students who used JITS outperformed the students in the traditional classroom. A two-way ANOVA with repeated measures was conducted and confirms these results, indicating there was a significant statistical difference in performance scores between the two groups.

From a more personal perspective, students during the last 7 weeks of this course appeared to have gained more and more interest in JITS and offered many suggestions for the improvement of this ITS. Table 21 shows these results.

Table 18

*Performance of Students in Class JITSC*

JITSC Class

---

Student	Pretest (%)	Posttest (%)
S <sub>1</sub>	62.00	81.00
S <sub>2</sub>	73.30	83.10
S <sub>3</sub>	64.40	75.80
S <sub>4</sub>	58.90	69.40
S <sub>5</sub>	22.20	78.30
S <sub>6</sub>	78.90	89.40
S <sub>7</sub>	73.30	85.20
S <sub>8</sub>	65.60	77.10
S <sub>9</sub>	72.20	86.10
S <sub>10</sub>	80.00	90.00
S <sub>11</sub>	66.70	83.30
S <sub>12</sub>	45.70	70.00
S <sub>13</sub>	71.40	74.30
S <sub>14</sub>	74.40	87.20

---

Table 19

*Performance of Students in Class C*

## Control Class

---

<b>Student</b>	<b>Pretest (%)</b>	<b>Posttest (%)</b>
C <sub>1</sub>	64.40	80.80
C <sub>2</sub>	46.70	51.90
C <sub>3</sub>	70.00	85.00
C <sub>4</sub>	57.80	76.70
C <sub>5</sub>	52.10	39.60
C <sub>6</sub>	66.50	62.50
C <sub>7</sub>	74.10	65.60
C <sub>8</sub>	71.90	75.20
C <sub>9</sub>	63.30	76.00
C <sub>10</sub>	59.90	79.90
C <sub>11</sub>	56.50	59.70
C <sub>12</sub>	68.90	78.00
C <sub>13</sub>	68.30	69.10
C <sub>14</sub>	22.22	25.40
C <sub>15</sub>	53.30	66.00
C <sub>16</sub>	54.80	68.80
C <sub>17</sub>	69.40	75.40
C <sub>18</sub>	78.90	89.40
C <sub>19</sub>	52.10	37.50
C <sub>20</sub>	53.20	53.70
C <sub>21</sub>	71.90	68.80
C <sub>22</sub>	70.00	81.40
C <sub>23</sub>	77.80	87.50

---

Table 20

*Standard Statistical Measures for C and JITSC*

---

<b>Study</b>	<b>Statistical instrument</b>	<b>C</b>	<b>JITSC</b>
Pretest	Mean:	61.91%	64.93%
	Standard deviation:	12.52	15.17
Posttest	Mean:	66.25%	80.73%
	Standard deviation:	16.79	6.75

---

Table 21

*Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C and JITSC*

---

<b>Source</b>	<b>Type IV sum of squares</b>	<b>df</b>	<b>Mean square</b>	<b>F</b>	<b>Sig.</b>
Intercept	326268.963	1	326268.963	1020.396	.000
Group	1330.710	1	1330.710	4.162	.049
Error	11191.161	35	319.747		

---

There was a significant level of differentiation between C and JITSC in performance scores for the last 7 weeks of the course investigated. Additionally, a two-way ANOVA with repeated measures was conducted confirming these results,  $F(1,35) = 4.162, p = .049$ , indicating there was a significant difference between the two groups (i.e., C and JITSC).

To account for the increased performance, it is conceivable that due to increased exposure to JITS and working through many programming problems, students' cognitive and skill development in core Java curriculum topics increased. Another perspective is that JITS may have been sufficiently sophisticated by this point in time that it was becoming an effective programming tutor. It is also reasonable to conclude, based on the performance scores, that even though there were technical problems and JITS may not have performed correctly (from a pedagogical perspective) at all times, the increased exposure to programming problems for students and the cognitive efforts to overcome these technical problems resulted in the students in the JITSC developing better skills than the students in the traditional classroom environment. As one instructor stated, "Any additional programming practice is good practice" (Karolyi, 2004). This concept is also reinforced by the Carnegie-Mellon ITS researchers philosophy: "practice make perfect" (Anderson et al., 1995; Anderson & Pelletier, 1991). Figure 35 presents the performance of C and JITSC students using mean grades as data in the form of a graph.

### ***Recommendations***

At the completion of the second Program Development Session, there were no major suggestions that either students or instructors offered. The students seemed

### JITSC versus C Performance Score Comparison Using Pretest and Posttest Means as Data

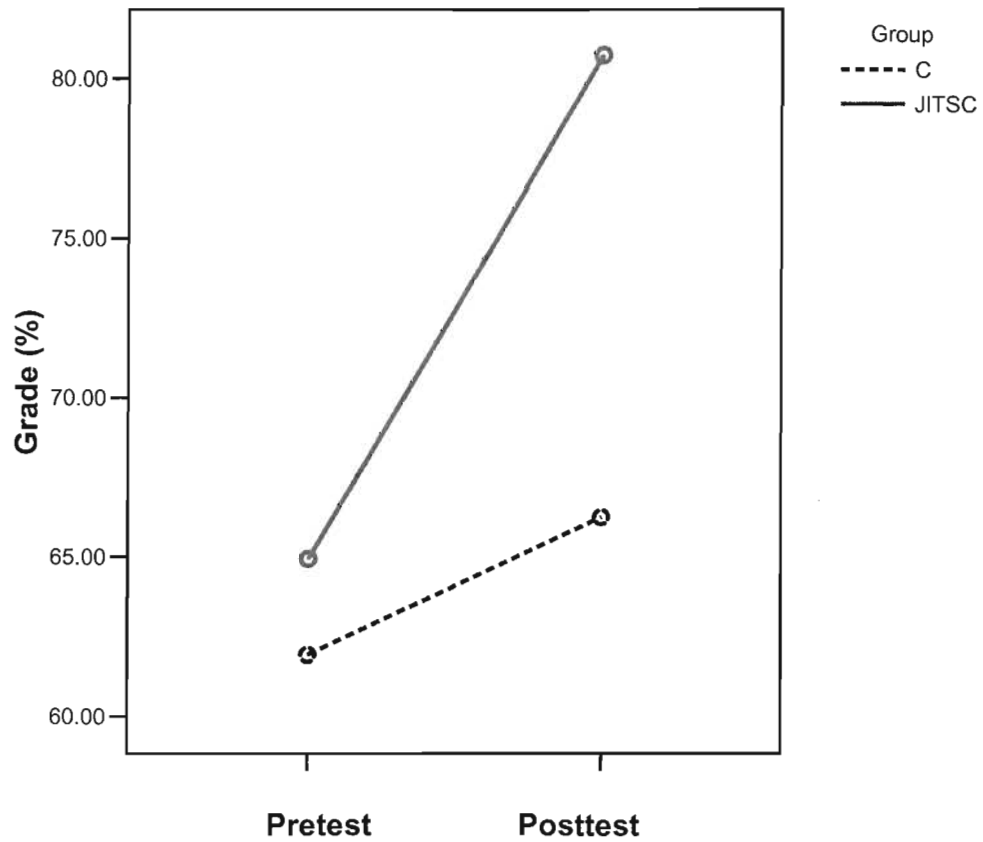


Figure 35. Performance of C and JITSC students using mean grades as data.



content with the performance and manner in which JITS tutored. Instructors were still patiently waiting for the JITS Authoring Tool to be completed.

### **Third Program Development Session: Section #1: JITS Developmental Research**

The third Program Development Session was the last research session conducted for the dissertation. During this session, JITS underwent a number of refinements to both the User Interface and the infrastructure. The information gathered during this session was based primarily on the suggestions by students and partly from the researcher's notes during the First and Second Program Development Sessions that would be beneficial to implement.

This section presents the findings of experiments that were conducted during the fall term of 2004 (i.e., September to December, 2004). There were three different control groups and two experimental groups.

#### ***Session #3, Section #1, Part #1: September 10 to September 28, 2004***

The first issue raised during this third Program Development Session was associated with the "Solutions" button. When the student pressed this button, duplicate solutions were presented. The researcher's solution to this problem was a modification to the way in which JITS extracts information from the database. Now only unique solutions for a specific problem are displayed.

Another issue that was suggested by some students was that additional problems be created that are more difficult and involve instantiation of objects and invoke functions. Unfortunately, a subproblem needed to be solved before the researcher could accommodate the students' request. The researcher integrated new strategies for the name of the classes for the problems. In Java, the classname must match the filename.

With some infrastructure changes, the researcher resolved the problem. The classname for a problem is now used in the structure of the filename to be compiled (i.e., Problem\_<problem\_set\_id>\_<problem\_id>\_<students\_name>.java is the full unique name used). JITS now works with more advanced problems, for example:

```
public class Power {
    public double powergen(int num) {
        return num * num;
    }

    public static void main (String args [] ) {
        Power p = new Power();
        double result = 0;

        result = p.powergen(10);
    }
}
```

The last issue that was raised during this study was associated with logic errors. The researcher observed a scenario where a student's submission did not have any syntax errors, but it had logic errors. In this type of situation, students stated that they would like assistance when these types of errors occur.

The solution to support students' that are encountering logic errors is a very difficult problem. The researcher worked on developing a sophisticated component of JITS AI\_Module that extracts information from the "Student" object, the "Problem" object. The Problem's keywords are used to provide assistance to students that are encountering logic errors. (Problem keywords are also included in the Problem Specification section for all problems.) The goal was to develop a logic-error feedback mechanism so that the student would feel more supported. Two examples are presented that illustrated the development of JITS in the area of responding to student's submissions containing logic errors.

**Example #1: Logic error but submission has correct constructs****Problem: (1 of 5) in Problem Set # 2 (Topic: Java Statements)**

Write a program called Together which concatenates (i.e., combine) several Strings together.

**Program Specifications:**

This program requires you to write a simple expression that stores a concatenated String into a variable. Fill in the code to complete this program.

**Required Output:**

Combined = Hello There Bub !! :)

```
public class Together {
    public static void main(String [] args) {
        String combined = "";
        String first = "Hello";
        String second = "There";
        String third = "Bub !! :)";

        << ===== >>
        << student enters code here >>
        << ===== >>

        System.out.println("Combined = " + combined);
    }
}
```

Suppose the student enters code that produces the following output:

Combined = Hello ThereBub !! :)

JITS previously would respond: "Sorry. No hints available."

Now, JITS responds as follows:

Good attempt. Now let's examine your code.  
 Good news... You are using the correct constructs.  
 However, it looks like there is a **logic** error in your program.  
 When working with arithmetic operators (e.g., \*, /) **logic** errors may occur. They usually happen because of a mistake in your **formula**.  
 Look carefully at this section in your code.

**View the Tutorial** for additional help.

### Example #2: Logic error but submission has incorrect constructs

The following example presents another situation where a student's submission does not have any syntax errors; however, there is one or more logic errors present. If the student submits a solution that does not use the correct constructs (as defined in the Problem Specification and the Problems' keywords), then JITS can guide the student to use the appropriate constructs for the problem.

#### **Problem: (1 of 4) in Problem Set # 4 (Topic: for loops)**

Write a program called Summer which adds all the integer numbers from 1 to a specified number ( $N$ ). For example, if  $N$  were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

#### **Program Specifications:**

This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

#### **Required Output:**

Sum = 55

```
public class Summer {
    public static void main(String [] args) {
        int sum = 0;

        << ===== >>
        << student enters code here >>
        << ===== >>

        System.out.println("Sum = " + sum);
    }
}
```

Suppose the student enters code that is syntactically correct yet has logic errors and does not use the correct constructs as specified in the “Program Specifications” section of this problem. Previously, JITS would respond: “Sorry. No hints available.” Now JITS responds more appropriately:

Nice Try!  
It looks like there is a **logic** error in your program.  
Please re-read the Problem Specification.  
You need to use specific constructs in your solution.  
In this problem you need to use: **for**  
Make corrections to your program and submit it again.

***Session #3, Section #1, Part #2: October 3 to October 8, 2004***

In this session, students suggested that the “My Performance” button should provide more substantial information regarding individual performance in JITS. During this study, the researcher spent 40 hours in redesigning and refining the “My Performance” infrastructure. Part of this restructuring required the alteration of the Student\_Problems table in the JITS ORACLE schema. Another field entitled “viewed\_solution” was added to this table, which was used to record whether or not the student pressed the “View Solutions” button while working on a problem. This in turn proved to be helpful in the redevelopment of the “My Performance” button to more effectively evaluate the student’s work. The researcher wanted to provide clear, meaningful results of how the student is performing and include relative information to the “average student” (other Students in JITS). A sample output of a student pressing the “My Performance” button is presented in Figure 36. Notice the organization of the output.

**My Performance:**

<b>Problem Set</b>	<b>Problem #</b>	<b>Solved</b>	<b>Solution Viewed</b>	<b>Average Student</b>
1	1	No. 4 attempts so far.	<i>Yes.</i>	2 attempts to solve.
1	2	No. 2 attempts so far.	<i>Yes.</i>	2 attempts to solve.
1	3	No. 4 attempts so far.	<i>Yes.</i>	2 attempts to solve.
3	1	No. 1 attempt so far.	<b>No.</b>	1 attempt to solve.
3	2	No. 1 attempt so far.	<b>No.</b>	1 attempt to solve.
4	1	<i>Yes.</i> It took 5 attempts.	<b>No.</b>	2 attempts to solve.
4	2	No. 3 attempts so far.	<b>No.</b>	3 attempts to solve.
4	3	No. 2 attempts so far.	<b>No.</b>	6 attempts to solve.
6	1	No. 1 attempt so far.	<b>No.</b>	1 attempt to solve.
6	3	No. 1 attempt so far.	<b>No.</b>	1 attempt to solve.
7	1	<i>Yes.</i> It took 2 attempts.	<b>No.</b>	1 attempt to solve.
7	2	No. 1 attempt so far.	<b>No.</b>	1 attempt to solve.

Figure 36. "My Performance" button displays performance information for each student.

During the on-site visits, the researcher asked each student in the JITCS for their opinion on the “My Performance” button and asked if there was any other information they would like to have included in the report. All of the students said they were content with the performance report.

***Session #3, Section #1, Part #3: October 8 to October 18, 2004***

During this part of the session, there were mostly technical problems that needed attention. Two students identified technical problems with Problem Set #2 Problem #2. These were corrected within 1 hour of my being notified. There was a mistake with the Average Student section of “My Performance” button. In the “Average Student” section, it was displaying “0 attempts to solve” in the case where a problem had never been solved (i.e., no solution available). It should have said “Problem not yet solved.” The researcher corrected this section of the “My Performance” infrastructure within 2 days from the problem being demonstrated to the researcher.

During this week, many students attempted a lot of problems in JITS. Some students found a few typographical errors in the problem descriptions. Some students found some minor technical problems. As a result, the researcher clarified the problem statements in the problem database. The researcher also corrected the naming of the Java class files which were causing the minor technical problems.

There was one additional issue that was very upsetting for me as the designer and developer. The network connection from Milton (the researcher’s development location) and Sheridan College was proving to be very temperamental. Sometimes connections would be unbearably slow, while other times the connections would simply fail. The problem was Sheridan’s hardware device called a “PacketShaper,” which was

misbehaving. The PacketShaper is responsible for classifying network traffic that travels into Sheridan and out of the college. In other words, access from outside the college into the Sheridan network connection is not working properly. The researcher discussed the situation with the Chief IT Director, who explained that it was a very complicated problem and would probably not get resolved until the end of December. Fortunately, it did not affect students trying out JITS located within Sheridan or students trying JITS outside of typical business hours. It also did not appear to cause any loss of service for students located close to either of Sheridan's main campuses (i.e., Oakville and Brampton regions).

***Session #3, Section #1, Part #4: October 22 to November 2, 2004***

Several students stated that the "tutorial" could be more useful. For instance, they suggested that one of the problems in problem set #2 (i.e., Java Statements programming topic) uses `public static final` for the constant  $\pi$ . The tutorial for this section should include information about `public static final`. In other words, there should be a closer match between the problems in the problem sets and the tutorial topics section. This was a very good point, and the researcher spent the remainder of the week carefully going over the tutorial information and made sure that the programming problems related to the information in the tutorials.

A number of students found that the JITS User Interface required a bit of getting used to. For instance, many students took a few tries to discover that they needed to scroll down in the window to see the program output, hints, solutions, etc. These students suggested that a "Help" button be included that would display basic features of JITS and how to get around in the Java Intelligent Tutoring System. During these few



weeks, the researcher added a “Help Me” button (located beside the student’s name in JITS) and completed the infrastructure and the Java ServerPage to support it. Figure 37 and Figure 38 depict the “Help Me” button location on JITS’ User Interface and JITS’ Help screen respectively.

***Session #3, Section #1, Part #5: November 3 to November 5, 2004***

Some students stated they would like more information from the “hints.” In some situations, JITS would respond: “Sorry. No hints available.” The researcher observed that these types of situations arose because JECA could not determine the intent of the student’s submission to a program. The researcher modified JECA to filter the compiler output (i.e., output from running “javac” (Sun’s Java compiler) and made it more “friendly” for students to read. Two examples are presented as follows. In both of the examples, the student did not create a variable as required for the solution. This results in a compilation error. Consequently, JECA now formats the output from the compiler in a friendly fashion for JITS to present to the student. Figure 39 and Figure 40 depict these scenarios.

During this study, outside of visits with students, the researcher continued to work with Sheridan IT staff on the network problem. The PacketShaper device was still not working correctly. This hardware device is used to packet shape traffic in and out of the college. Over the last month, the researcher had many problems working from home to connect to various servers in order to work on JITS. The programs just take extremely long to process, or they simply fail. Unfortunately, the problem was beyond my control to solve. Fortunately, students did not have any difficulties working with JITS. Within the college, access to JITS remained fast and stable.

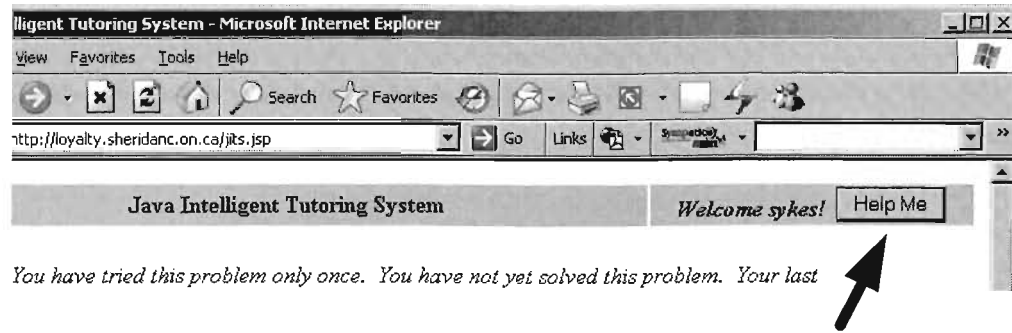


Figure 37. Top right section of JITS' User Interface displaying the "Help Me" button.

http://loyalty.sheridanc.on.ca - Java Intelligent Tutoring System -- Help - Microsoft Internet Explorer

## Help with the Java Intelligent Tutoring System (JITS)

The Java Intelligent Tutoring System is intended to be used by beginner Java programming students. The User Interface is divided into a number of sections as seen below.

This message area informs you of your status in solving the current problem.

Problem Sets contain Problems. Here the student is working on problem 1 in the while-loops problem set.

Keep in mind the required output. This is what your program needs to produce.

Type in your solution here.

Click the "Submit" button when you are ready to submit your solution to JITS.

This Output Area displays the output of your program, hints, and other messages.

Navigate through the problem set by "Previous" and "Next" Problem buttons. Make sure to Click "View Hints" when you encounter problems. If you get really stuck you can click the "View Solution" button.

You can see your individual performance by clicking: "My Performance"

The tutorial explains the Programming Topics and provides code examples that will help you in solving the problems.

To change to a different Problem Set, select it, & click "Take me there"

Java Intelligent Tutoring System

NOTE: You have tried this problem only once. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

Problem: (1 of 4) in Problem Set #6 (Topic: while loops)  
Write a program called `Summer` which adds all the integer numbers from 1 to a specified number (N). For example, if N were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

Program Specifications:  
This program requires the use of a while loop structure. A skeleton structure of the solution is given. Fill in the code to complete the program.

Required Output:  
Sum=45

```
public class Summer {
    public static void main(String [] args) {
        int sum = 0;
        System.out.println("Sum = " + sum);
    }
}
```

OUTPUT:

Java Programming Topics

- Java Basics
- Java Statements & statements
- For loops
- do while loops
- while loops
- Arrays

Take me there

View the Tutorial

Submit View Hint View Solution View Solution

Previous Problem Next Problem My Performance Exit

Done Internet

Figure 38. JITS Help screen is used to assist new users to get oriented with this ITS.

Java Intelligent Tutoring System

Welcome  
tra\_sam1\_student40!

NOTE: You have attempted this problem 4 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

Problem: (3 of 4) in Problem Set # 2 (Topic: Java Statements)  
Write a program called Power Generator which calculates the result of a number multiplied by itself.

Program Specifications:  
This program requires the use of a function. A skeleton structure of the solution is given.

Required Output:  
Result = 10000

```
public class Power {
    public int powergen(int num) {
        return num * num;
    }
    public static void main(String [] args) {
        Power p = new Power();
        p.powergen(10);
        System.out.println("Result = " + result);
    }
}
```

Submit View Top Hint View All Hints View Solution

Previous Problem Next Problem My Performance Exit

OUTPUT:

Your program did not compile. Below are some hints:

```
cannot resolve symbol
symbol : variable result
```

Programming Topics

- Java Basics
- Java Statements
- If statement
- for loops
- do while loops
- while loops
- Arrays

Take me there

View the Tutorial

Figure 39. Improved JECA demonstrating filtered output from the compiler and JITS presenting the results in a friendly way for the student to make corrections.

Java Intelligent Tutoring System - Microsoft Internet Explorer

Address: <http://loyalty.sheridanc.on.ca/jts.jsp>

Java Intelligent Tutoring System

Welcome tra\_sem1\_student40!

NOTE: You have attempted this problem 5 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

Problem: (3 of 4) in Problem Set #2 (Topic: Java Statements)  
Write a program called Power Generator which calculates the result of a number multiplied by itself.

Program Specifications:  
This program requires the use of a function. A skeleton structure of the solution is given.

Required Output:  
Result = 10000

```
public class Power {
    public int powergen(int num) {
        return num * num;
    }
    public static void main(String [] args) {
        Power p = new Power();
        double result;
        p.powergen(10);
        System.out.println("Result = " + result);
    }
}
```

System.out.println("Result = " + result);

Submit View Top Hint View All Hints View Solution

Previous Problem Next Problem My Performance Exit

OUTPUT:

Your program did not compile. Below are some hints:

variable result might not have been initialized

System.out.println("Result = " + result);

Programming Topics

- Java Basics
- Java Statements
- If statement
- for loops
- do while loops
- while loops
- Arrays

Take me there

View the Tutorial

Figure 40. A variation of a compiler error due to a student's submission. Previous versions of JECA would simply return a hint: "Sorry. No hints available." The improved JECA is intended to be more helpful and presents compiler errors in a more friendly way.

*Session #3, Section #1, Part #6: November 5 to November 8, 2004*

Some students suggested that actual equations be presented in problems that refer to mathematical expressions. For example, in one of the problems, the information was presented only as text: “the volume of a cone is 1/3 times PI times the radius squared times the height.” Students suggested it would be clearer to present this information as:

$\frac{1}{3} \pi r^2 h$ . For this problem, the researcher first added another column to the Problems

table of the JITS database schema. The revised schema is presented in Table 22. The researcher then developed a Windows Popup which streams a binary image from the ORACLE database. The researcher updated the main JITS webpage (i.e., jits.jsp JavaServer Page) to include a link “View the image for this problem.” Figure 41 depicts the revised JITS User Interface. If the student clicks on the link, a popup is created and the image for the problem is streamed to the popup. For example, the image for the volume of a cone problem is depicted in Figure 42. Some students suggested that some of the problems should be altered or new ones created to more closely reflect real-world scenarios. While programming, there are certain situations where it is more desirable to use one construct over another. As a result, the researcher created a number of new problems that more closely reflect real-world situations. These new problems employ the use of specific programming constructs.

The researcher then developed a Windows Popup which streams a binary image from the ORACLE database. The researcher updated the main JITS webpage (i.e., jits.jsp JavaServer Page) to include a link “View the image for this problem.” Figure 41 depicts the revised JITS User Interface.

Table 22

*Redesigned JITS ORACLE Schema Tables to Accommodate Pictures*


---

```

CREATE TABLE PROBLEM_SETS (
    problem_set_id      NUMBER(3),
    problem_set_title   VARCHAR2(30),
    problem_set_desc    VARCHAR2(400),
);

CREATE TABLE PROBLEMS (
    problem_set_id      NUMBER(3),
    problem_id         NUMBER(3),
    problem_desc        VARCHAR2(400) NOT NULL,
    problem_spec        VARCHAR2(400) NOT NULL,
    problem_output      VARCHAR2(50),
    template_top_section VARCHAR2(400),
    template_bottom_section VARCHAR2(400),
    problem_difficulty  VARCHAR2(20),
    problem_keywords    VARCHAR2(200),
    picture             LONG RAW,
);

CREATE TABLE STUDENTS (
    student_name        VARCHAR2(30),
    student_password    VARCHAR2(15),
    problem_set_id      NUMBER(3),
    problem_id         NUMBER(3),
    skill_level         NUMBER(3),
    performance_rating  NUMBER(3),
    performance_history VARCHAR2(2000),
    times_connected     NUMBER(5),
    date_last_connection VARCHAR2(30),
    picture             LONG RAW,
);

CREATE TABLE STUDENT_PROBLEMS (
    student_name        VARCHAR2(30),
    problem_set_id      NUMBER(3),
    problem_id         NUMBER(3),
    number_of_attempts  NUMBER(3),
    solved              CHAR(1),
    students_solution   VARCHAR2(500),
    solution_date       VARCHAR2(30),
);

```

---

Java Intelligent Tutoring System

Welcome e!  
Help Me

NOTE: You have tried this problem only once. You have successfully solved this problem!

Problem: (4 of 4) in Problem Set # 2 (Topic: Java Statements)  
Write a program called ConeVolume which calculates the volume of a cone with a radius of 17.23m and a height of 5m.

Program Specifications:  
This program requires the use of a function. The volume of a cone is  $\frac{1}{3}$  times PI times the radius squared multiplied by the height of the cone. [View the image for this problem.](#)

Required Output:  
Volume = 1553.63

```
public class ConeVolume {
    public static final double PI = 3.14;
    public double volume(double ht, double rd) {

        return 4/3.0 * PI * rd * rd * ht;

    }
    public static void main(String [] args) {
        ConeVolume obj = new ConeVolume();
        double cone_vol;
        cone_vol = obj.volume(5, 17.23);
        System.out.println("Volume = " + cone_vol);
    }
}
```

Submit View Top Hint View All Hints View Solution

Previous Problem Next Problem My Performance Exit

OUTPUT:

Programming Topics

- Java Basics
- Java Statements
- if statement
- for loops
- do while loops
- while loops
- Arrays

Take me there

View the Tutorial

Figure 41. Revised JITS User Interface accommodating a link to the image for the current problem.



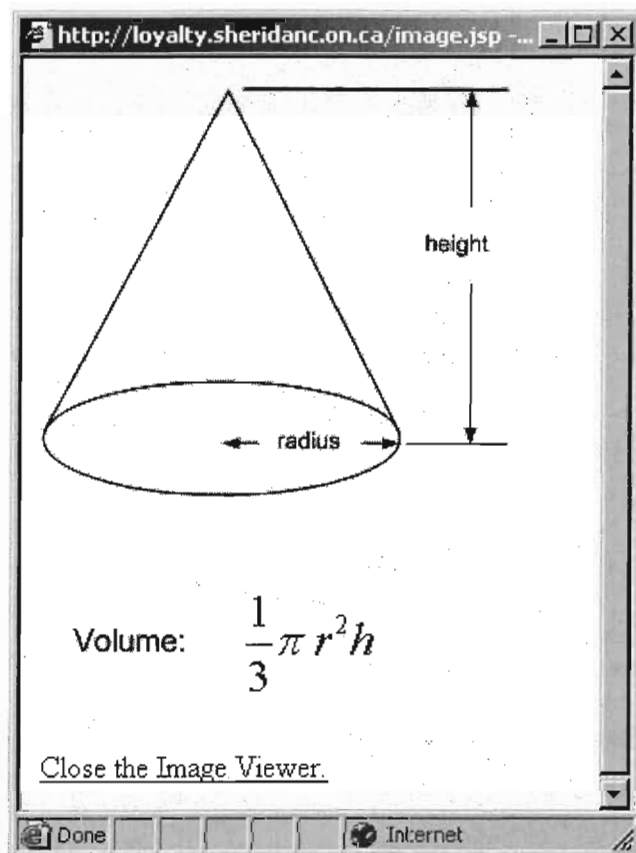


Figure 42. JITS Image Viewer depicting the image for the current problem.

A number of students discovered that if they simply click “Submit” and do not enter any code, JITS responds with some unreasonable comment like:

**Almost!**

It looks like there is a **logic** error in your program.  
Please re-read the Problem Specification.  
You need to use specific constructs in your solution.

In this problem you need to use: **for, if**  
Make corrections to your program and submit it again.

The researcher solved this problem by making a small change to the infrastructure of JITS. JITS now correctly identifies when a student genuinely tries to attempt to solve a problem. When the student does not type anything or any whitespace character (i.e., tab, newline, etc.) and clicks “Submit,” JITS simply ignores the inappropriate submission to the problem and prompts the student to try to solve the problem.

One student found a technical problem with Problem 3 of 4, Problem Set #5. The student’s solution was correct, yet JITS did not allow it as a valid solution. The researcher corrected the problem keyword requirements for this problem. Now the student’s solution is accepted by JITS.

One student found that Problem 3, Problem Set #2 was worded a bit confusingly. The student did not realize that the solution required a variable to be declared. In all other problems the variables are part of the actual problem template. As a result, the researcher corrected the wording of the problem to include in the Problem Specification section: “You need to declare a variable called ‘result’ in your solution.”

***Session #3, Section #1, Part #7: November 9 to November 10, 2004***

Two students mentioned to the researcher on November 5 that it would be nice to have the “My Performance” button contain hyperlinks to specific problems that the student has previously attempted. These students suggested that a link to the specific problem would be beneficial, especially when the student has not yet solved the problem. The link would provide a fast way to bring the student back to that problem. The researcher solved this problem by creating a button in the corresponding row for the problem set # and problem #. The researcher added colour, italics, and bold text to make a distinction between problems that have been solved by the student from those that have been attempted but not solved. Correctly answered problems appear on the screen in green (e.g., “Yes ! It took 5 attempts). Attempted, yet unsolved problems appear in red. Additionally, buttons are provided for quick access to each problem for the student. Figure 43 depicts the revised output of the “My Performance” button.

***Session #3, Section #1, Part #8: November 16 to December 1, 2004***

Some students suggested that the white-space in the JITS User Interface be reduced. This way more information could be presented on the screen. The researcher solved this problem by readjusting the components on the screen to fit more information and modified the “Note” section (located at the top of the JITS User Interface) to draw reference to the “Output” section:

*You have attempted this problem 9 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. See the "Output" section below for more information.*

Due to all the concurrent interactions between dozens of students and JITS and the intense database interactivity between JITS and the ORACLE database in order to

Problem #	Problem Set	Solved?	Solution Viewed?	Average Student	Review this Problem?
1	1	No -- 13 attempts so far.	Yes.	2 attempts to solve.	Review Problem: 1 of set: 1
2	3	No -- 1 attempt so far.	No.	2 attempts to solve.	Review Problem: 2 of set: 3
1	4	Yes ! It took 5 attempts.	No.	2 attempts to solve.	Review Problem: 1 of set: 4
1	6	No -- 3 attempts so far.	No.	2 attempts to solve.	Review Problem: 1 of set: 6
3	6	No -- 1 attempt so far.	No.	1 attempt to solve.	Review Problem: 3 of set: 6
1	7	Yes ! It took 2 attempts.	No.	1 attempt to solve.	Review Problem: 1 of set: 7
2	7	No -- 1 attempt so far.	No.	1 attempt to solve.	Review Problem: 2 of set: 7

Figure 43. Revised “My Performance” button output showing links to previously attempted problems, font, and colour distinctions between solved and unsolved problems.

track all the students in the system, the researcher decided to try to speed up the database connections. As a solution, the researcher developed a Connection Pool of JDBC established connections to the ORACLE JITS schema. This resulted in significantly faster web site performance for students using JITS.

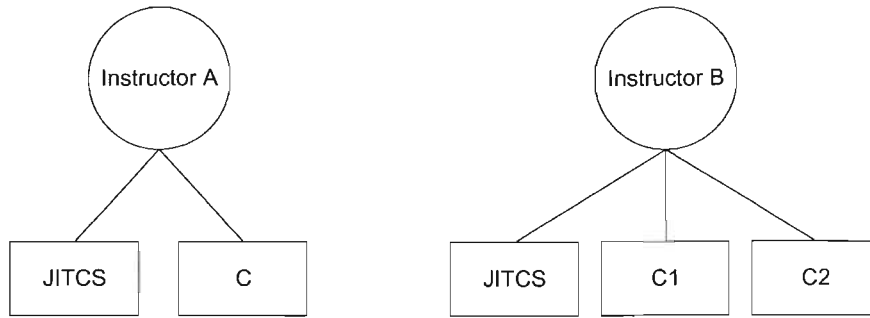
One student suggested that the “tab” key should be implemented; since typing in programming is a very common activity it should be implemented in JITS.

The researcher researched the problem. In HTML all elements in the form have a “tab index.” This means that a user can navigate from one form element to another by pressing the tab key (e.g., from one button to another). So, in order to override the default browser behaviour, the researcher wrote a small set of JavaScript programs that intercepts when the student presses the “tab” key. The scripts then treat the “tab” key as a proper tab as opposed to a form element navigation keystroke. The researcher tested this extended functionality in JITS in browsers such as Microsoft Internet Explorer 6, Mozilla, Netscape 7, and Firebird.

### **Third Program Development Session: Section #2: JITS Performance Score**

#### **Analysis**

There were two instructors involved in the third Program Development Session. Instructor “A” taught two introductory programming courses; one was JITSC and the other C. The second instructor (i.e., Instructor “B”) taught three courses. One was the experimental group (i.e., JITCS) and the others were C1 and C2, representing control groups. Figure 44 illustrates this structure.



*Figure 44.* Classifying control groups and experimental groups based on instructors.

### ***Instructor “A” Performance Score Analysis***

Table 23 and Table 24 display the performance scores of students taught by Instructor “A”. Table 25 presents a summary of the descriptive statistical findings on the performance scores. In order to determine the relationship between the performance scores in C and JITSC, a two-way ANOVA with repeated measures was conducted. Table 26 shows these results.

The students in the JITSC class outperformed students in the C class. There was, however, no significant level of differentiation between C and JITSC taught by Instructor “A” in performance scores. A two-way ANOVA with repeated measures was conducted that confirmed these results:  $F(1,37) = 0.083, p = .775$ , indicates there was no significant difference between the two groups (i.e., C and JITSC). Table 26 shows the results from the ANOVA.

### ***Instructor “B” Performance Score Analysis***

For this session there were two control groups (i.e., C1 and C2) and one experimental group (i.e., JITSC). Table 27 and Table 28 display the performance scores of students taught by Instructor “B.” Table 29 presents a summary of the descriptive statistical findings on the performance scores. In order to determine the relationship between the performance scores in C1, C2, and JITSC, a two-way ANOVA with repeated measures was conducted. Table 30 presents this ANOVA.

Additional statistical measures were conducted to determine the level of significance between specific groups (i.e., C1 vs. JITSC and C2 vs. JITSC). Table 31 presents the results from the ANOVA for between-subjects effects for C1 and JITSC. There was no statistically significant level of difference between C1 and JITSC.

Table 23

*Performance of Students in JITSC Class Taught by Instructor "A"*

---

Class JITSC

JITSC Class		
Student	Pretest (%)	Posttest (%)
S <sub>1</sub>	57.60	50.00
S <sub>2</sub>	68.40	86.29
S <sub>3</sub>	68.00	71.14
S <sub>4</sub>	74.00	88.00
S <sub>5</sub>	71.20	91.14
S <sub>6</sub>	45.20	55.14
S <sub>7</sub>	74.40	81.14
S <sub>8</sub>	48.00	53.14
S <sub>9</sub>	57.20	82.57
S <sub>10</sub>	47.20	76.29
S <sub>11</sub>	68.40	83.14
S <sub>12</sub>	58.80	89.43
S <sub>13</sub>	72.80	92.00
S <sub>14</sub>	66.80	80.00

---



Table 24

*Performance of Students in C Class Taught by Instructor "A"*

Class C

---

Student	Pretest (%)	Posttest (%)
C <sub>1</sub>	62.40	60.57
C <sub>2</sub>	68.00	80.57
C <sub>3</sub>	31.20	54.29
C <sub>4</sub>	57.20	88.57
C <sub>5</sub>	60.00	79.43
C <sub>6</sub>	53.20	54.29
C <sub>7</sub>	66.80	90.00
C <sub>8</sub>	63.20	70.00
C <sub>9</sub>	77.20	87.43
C <sub>10</sub>	51.60	74.29
C <sub>11</sub>	75.60	95.14
C <sub>12</sub>	72.80	98.86
C <sub>13</sub>	71.20	81.14
C <sub>14</sub>	72.00	90.00
C <sub>15</sub>	73.40	92.57
C <sub>16</sub>	58.80	74.29
C <sub>17</sub>	73.60	87.43
C <sub>18</sub>	70.00	90.00
C <sub>19</sub>	74.00	57.43
C <sub>20</sub>	61.20	76.29
C <sub>21</sub>	71.20	73.14
C <sub>22</sub>	60.00	65.14
C <sub>23</sub>	79.60	48.86
C <sub>24</sub>	45.20	38.86
C <sub>25</sub>	52.40	78.00

---

Table 25

*Standard Statistical Measures for C and JITSC Taught by Instructor "A"*

---

<b>Group</b>	<b>Pretest</b> mean and ( <i>standard deviation</i> )	<b>Posttest</b> mean and ( <i>standard deviation</i> )
C	65.272 (9.082)	76.663 (15.570)
JITSC	62.714 (10.313)	77.101 (14.391)

---

Table 26

*Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C and JITSC  
Taught by Instructor "A"*

---

<b>Source</b>	<b>Type IV sum of squares</b>	<b>df</b>	<b>Mean square</b>	<b>F</b>	<b>Sig.</b>
Intercept	356209.373	1	356209.373	1463.495	.000
Group	20.165	1	20.165	.083	.775
Error	9005.667	37	243.396		

---

Table 27

*Performance of Students in JITSC Class Taught by Instructor "B"*

Class JITSC

---

JITSC Class		
Student	Pretest (%)	Posttest (%)
S <sub>1</sub>	75.00	95.00
S <sub>2</sub>	95.00	97.14
S <sub>3</sub>	70.00	77.14
S <sub>4</sub>	85.00	90.00
S <sub>5</sub>	70.00	80.00
S <sub>6</sub>	70.00	100.00
S <sub>7</sub>	95.00	100.00
S <sub>8</sub>	55.00	93.57
S <sub>9</sub>	80.00	81.43
S <sub>10</sub>	25.00	69.29
S <sub>11</sub>	35.00	78.57
S <sub>12</sub>	95.00	92.86
S <sub>13</sub>	82.50	95.71
S <sub>14</sub>	30.00	85.00

---

Table 28

*Performance of Students in C1 and C2 Classes Taught by Instructor "B"*

## Class C1

Student	Pretest (%)	Posttest (%)
C1 <sub>1</sub>	75.00	92.86
C1 <sub>2</sub>	100.00	95.71
C1 <sub>3</sub>	60.00	70.00
C1 <sub>4</sub>	75.00	83.57
C1 <sub>5</sub>	70.00	92.86
C1 <sub>6</sub>	85.00	95.71
C1 <sub>7</sub>	55.00	77.14
C1 <sub>8</sub>	65.00	88.57
C1 <sub>9</sub>	95.00	100.00
C1 <sub>10</sub>	80.00	82.57
C1 <sub>11</sub>	25.00	65.71
C1 <sub>12</sub>	97.50	78.57
C1 <sub>13</sub>	70.00	89.29
C1 <sub>14</sub>	55.00	82.86
C1 <sub>15</sub>	70.00	92.86
C1 <sub>16</sub>	60.00	95.71
C1 <sub>17</sub>	82.50	77.86
C1 <sub>18</sub>	95.00	97.14

## Class C2

Student	Pretest (%)	Posttest (%)
C2 <sub>1</sub>	65.00	70.71
C2 <sub>2</sub>	65.00	69.29
C2 <sub>3</sub>	60.00	71.43
C2 <sub>4</sub>	52.50	80.00
C2 <sub>5</sub>	75.00	62.86
C2 <sub>6</sub>	70.00	72.14
C2 <sub>7</sub>	80.00	64.29
C2 <sub>8</sub>	67.50	52.00
C2 <sub>9</sub>	77.50	81.43
C2 <sub>10</sub>	40.00	80.71
C2 <sub>11</sub>	80.00	71.43
C2 <sub>12</sub>	75.00	80.00
C2 <sub>13</sub>	35.00	34.29
C2 <sub>14</sub>	85.00	78.57
C2 <sub>15</sub>	57.50	67.14
C2 <sub>16</sub>	75.00	62.86
C2 <sub>17</sub>	55.00	83.57
C2 <sub>18</sub>	70.00	82.86
C2 <sub>19</sub>	77.50	81.43
C2 <sub>20</sub>	67.50	68.57
C2 <sub>21</sub>	70.00	68.57
C2 <sub>22</sub>	75.00	78.57
C2 <sub>23</sub>	80.00	70.71

Table 29

*Standard Statistical Measures for C1, C2, and JITSC Taught by Instructor "B"*

---

<b>Study</b>	<b>Statistical Instrument</b>	<b>C1</b>	<b>C2</b>	<b>JITSC</b>
Pre-Test	Mean:	73.055	67.608	68.750
	Standard Deviation:	18.738	12.758	23.913
Post-Test	Mean:	86.626	71.018	88.265
	Standard Deviation:	9.845	11.291	9.650

---

Table 30

*Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C1, C2, and JITSC Taught by Instructor "B"*

---

<b>Source</b>	<b>Type IV sum of squares</b>	<b>df</b>	<b>Mean Square</b>	<b>F</b>	<b>Sig.</b>
Intercept	608112.166	1	618112.166	1963.680	.000
Group	2674.463	2	1337.231	4.318	.018
Error	16103.353	52	309.680		

---

Table 31

*Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C1 and JITSC  
Taught by Instructor "B"*

---

<b>Source</b>	<b>Type IV sum of squares</b>	<b>df</b>	<b>Mean Square</b>	<b>F</b>	<b>Sig.</b>
Intercept	394919.952	1	394919.952	991.540	.000
Group	28.012	1	28.012	.070	.793
Error	30	398.289			

---



Table 32 presents the results from the ANOVA for between-subjects effects for C2 and JITSC. There was a statistically significant difference between C2 and JITSC at the 0.05 level,  $F(1,35) = 4.934, p = .033$ .

### **Third Program Development Session: Section #3: JITS Summary and Recommendations for Further JITS Development**

This section presents a summary and recommendations for the refinement of JITS for the third program development session. The summary presents the results of the performance score analysis, and student and professor perspectives. The recommendations section discusses areas for improvement in the Java Intelligent Tutoring System.

#### ***Summary***

A tremendous amount of redesign and redevelopment occurred during this session. The major reason for these changes was due to student involvement and student interest in this project.

Figure 45 shows a pictorial summary of performance scores between C1, C2, and JITCS using the mean grades as the data. There was a significant statistical difference between C2 and JITSC at the 0.05 level. See Table 30, Table 31, and Table 32 for specific results. Figure 46 presents a plot of the performance scores between C1 and JITSC. Figure 47 presents a plot of the performance scores between C2 and JITSC.

From a qualitative perspective, the results show a difference between experimental groups and control groups for both Instructors “A” and “B” classes. Overall, students enjoyed and benefited from working with the Java Intelligent Tutoring System. Table 33 depicts the summary statistics from the qualitative survey. (See

Table 32

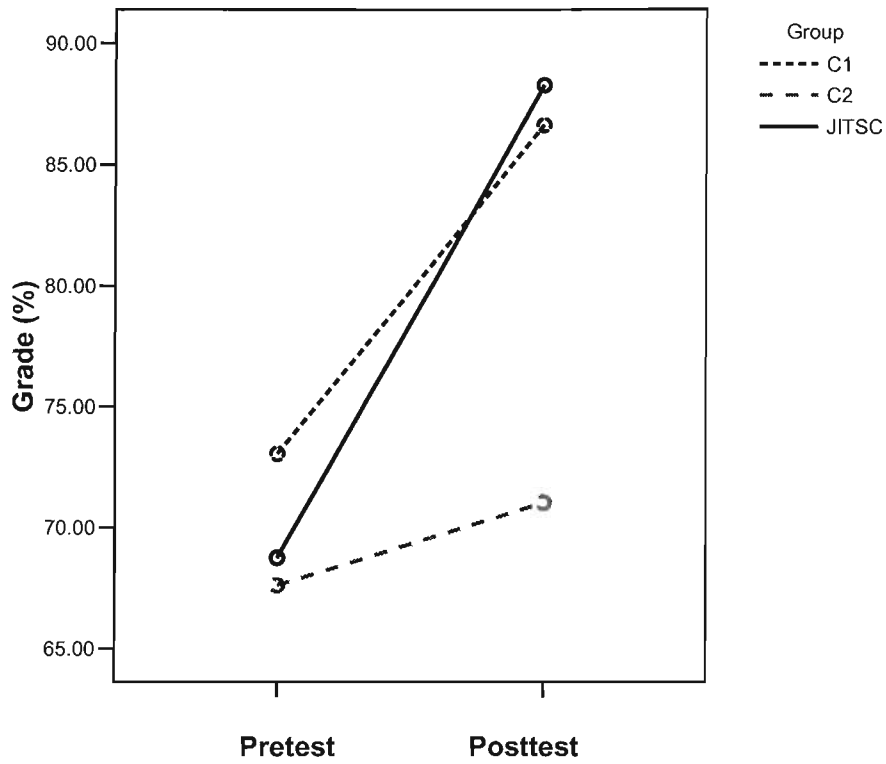
*Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C2 and JITSC  
Taught by Instructor "B"*

---

<b>Source</b>	<b>Type IV sum of squares</b>	<b>df</b>	<b>Mean Square</b>	<b>F</b>	<b>Sig.</b>
Intercept	380327.356	1	380327.356	1275.534	.000
Group	1471.210	1	1471.210	4.934	.033
Error	10435.991	35	298.171		

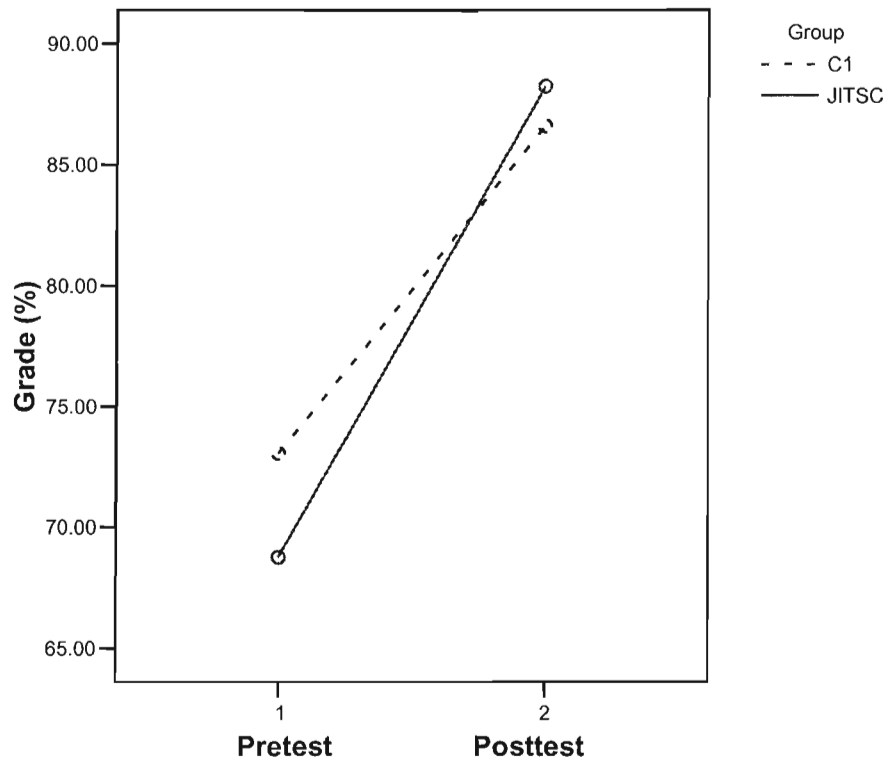
---

**Performance Comparison between JITSC, C1, and C2 using  
Pretest and Posttest Means as Data**



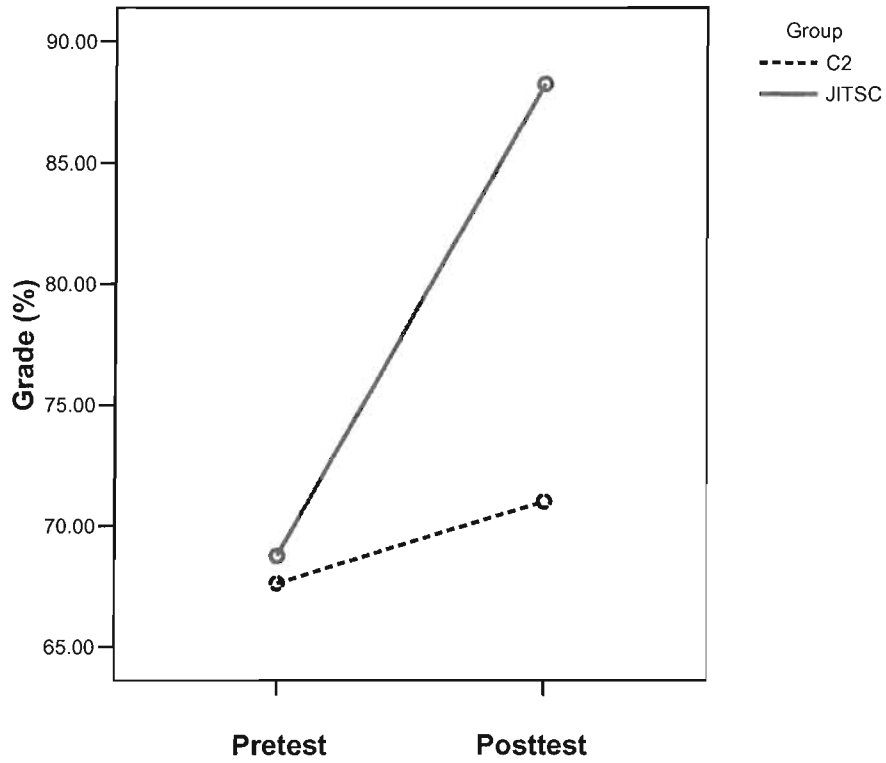
*Figure 45.* JITSC versus C1 and C2 performance comparison using pretest and posttest means as data.

**Performance Comparison between JITSC and C1 using  
Pretest and Posttest means as data**



*Figure 46.* JITSC versus C1 performance comparison using pretest and posttest means as data.

**Performance Comparison between JITSC and C2 using  
Pretest and Posttest means as data**



*Figure 47.* JITSC versus C2 performance comparison using pretest and posttest means as data.

Table 33

*JITS Qualitative Summary Results for JITSC Students*

---

1. Usefulness.....	93%
2. Beneficial .....	86%
3. JITS is better than a traditional classroom.....	0%
4. Ease of JITS Tutoring Style.....	86%
5. Enjoyable.....	79%
6. Learn better using JITS than in a classroom....	43%

---

Table 7 for the interview sheet.) Based on the students' comments, JITS performed extremely well. In most of the categories, there were significant improvements over the First Program Development Session findings (see Table 17).

***The Students' Perspective.*** Most of the students enjoyed working with JITS. The following section presents the main comments as printed on the student's survey.

1. "It is easy to learn using this system instead of using a textbook."
2. "It helps me understand material by specifically emphasizing certain [programming] parts."
3. "It was nice to see my knowledge and application of Java works."
4. "Very beneficial because it actually allowed me time to practice that I could not have had time for otherwise."
5. "Really helped go over important concepts and programming expense especially for someone who has trouble working alone from home."
6. "Overall, [JITS] is an excellent piece of software for serious Java learners."

Students felt that the hints were extremely good when either syntax errors or logic errors occurred. The Java Error Correction Algorithm (JECA) provided JITS with the necessary information to provide an appropriate hint to the student for the syntax errors. Similarly, the AI\_Module provided the rationale behind logic errors to JITS when they occurred. During this last Program Development Session, it is clear that JITS supported students in their development of fundamental Java concepts.

### ***Recommendations***

At the end of the third Program Development Session, there were no further major suggestions offered by students or instructors. However, there were some minor

suggestions. Some students stated, “JITS should include some gaming elements to make it more exciting.” This is not an unreasonable request. In fact, there are a number of Conferences and Journals that revolve around the topic of video streaming and interactively rich media in the field of Intelligent Tutoring Systems. The researcher is currently exploring the technical issues with embedding the Java Intelligent Tutoring System with QuickTime movies using specialized Java modules.

Most of the students seemed content with the performance and manner in which JITS tutored. However, there were two minor suggestions that could not be met. First, one student suggested adding colour to the source code area where the student enters his/her solution to a problem. The user interface technologies used to implement JITS (i.e., HTML, Javascript, and Java ServerPages) unfortunately can not accommodate this request. One student suggested that the error pointer (i.e., the ^ symbol used to identify the exact spot where an error is located in the student’s submission) be shown in the actual source code area. Again, unfortunately, due to the technologies selected for the implementation of JITS, this is impossible to do.



## **CHAPTER SEVEN: SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS**

This chapter presents the summary, conclusion, and recommendations of the research work conducted in this dissertation. The summary section provides a review of the final version of Java Intelligent Tutoring System with specific emphasis on the user interface. The second section of this chapter is the conclusion, which includes possible confounds regarding the performance score results and analysis. Following the Conclusion section is the Implications section that discusses the educational implications of this work. The last section of this chapter presents recommendations for future work in the area of Intelligent Tutoring Systems, paying particular attention to programming tutors and the Java Intelligent Tutoring System.

### **Summary**

The Java Intelligent Tutoring System was designed, developed, and assessed in this dissertation. Recall the generally accepted definition for an ITS is a system that employs artificial intelligence methods to assist trainees to improve their problem solving skills by monitoring their reasoning, tracking errors to their source, and, based on the diagnosis, providing advice and assistance to strengthen problem solving skills (Tracey, 2003). ITS allows for more open-ended programs (Tracey, 2003). JITS satisfies all of the criteria specified above. Accordingly, JITS is thus a fully qualified Intelligent Tutoring System. The following section elaborates this point by presenting the completed Java Intelligent Tutoring System including the user interface and user modeling components as they relate to this ITS criteria. The section also provides a summary of the performance score results from the quantitative experiment that

determined the effectiveness of the Java Intelligent Tutoring System at the Sheridan Institute of Technology and Advanced Learning.

### ***JITS User Interface***

The user interface for the Java Intelligent Tutoring System underwent a number of significant changes throughout the duration of the research study. During some of the experiments, major changes were conducted within very short timelines to ensure the students' suggestions were taken seriously and that significant changes were made to the user interface. Figure 48 depicts the completed JITS user interface.

The first section (i.e., label 1) presents a personalized welcome to the student logged in. Label 2 presents a note relative to the current state of solving the problem at hand. In this section, notes are dynamically created by JITS that are personalized to each student. Label 3 presents the problem template structure including the problem statement, the problem specifications, and the required output. This section also draws reference to the problem number out of the total number of problems available in this programming topic. At the end of Section 3, a link (i.e., label 4) is provided to a picture if the problem has a visual component (i.e., an equation or relevant drawing) to assist the student in more clearly understanding the problem. If the student clicks the link, the picture is shown in a separate window to allow the student to refer to the picture while at the same time working with the main JITS user interface. Label 5 shows the template provided by JITS for each problem in the system. Label 6 presents the editing region where the student types his/her solution. Label 7 depicts the various buttons which the students use to interact with JITS. Buttons include "Submit" to submit a solution to a problem and to receive feedback. "View Top Hint" and "View All Hints" buttons are

Java Intelligent Tutoring System - Microsoft Internet Explorer

Address <http://loyalty.sheridanc.on.ca/jits.jsp>

Java Intelligent Tutoring System Welcome sykes! [Help Me](#)

**NOTE:** You have attempted this problem 4 times. You have not yet solved this problem. Your last attempt is presented for you in the code area. Please try again.

**Problem: (3 of 5) in Problem Set # 2 (Topic: Java Statements)**  
Write a program called Power Generator which calculates the result of a number multiplied by itself.

**Program Specifications:**  
This program requires the use of a function. A skeleton structure of the solution is given. You need to declare the variable: result [View the image for this problem.](#)

**Required Output:**  
Result = 10000

```
public class Power {
    public int powergen(int num) {
        return num * num;
    }
    public static void main(String [] args) {
        Power p = new Power();
        double result
        p.powergen(19);
        System.out.println("Result = " + result);
    }
}
```

[Submit](#) [View Top Hint](#) [View All Hints](#) [View Solution](#)

[Previous Problem](#) [Next Problem](#) [My Performance](#) [Exit](#)

**OUTPUT:**

**Programming Topics**

- Java Basics
- Java Statements
- If statement
- for loops
- do while loops
- while loops
- Arrays

[Take me there](#)

[View the Tutorial](#)

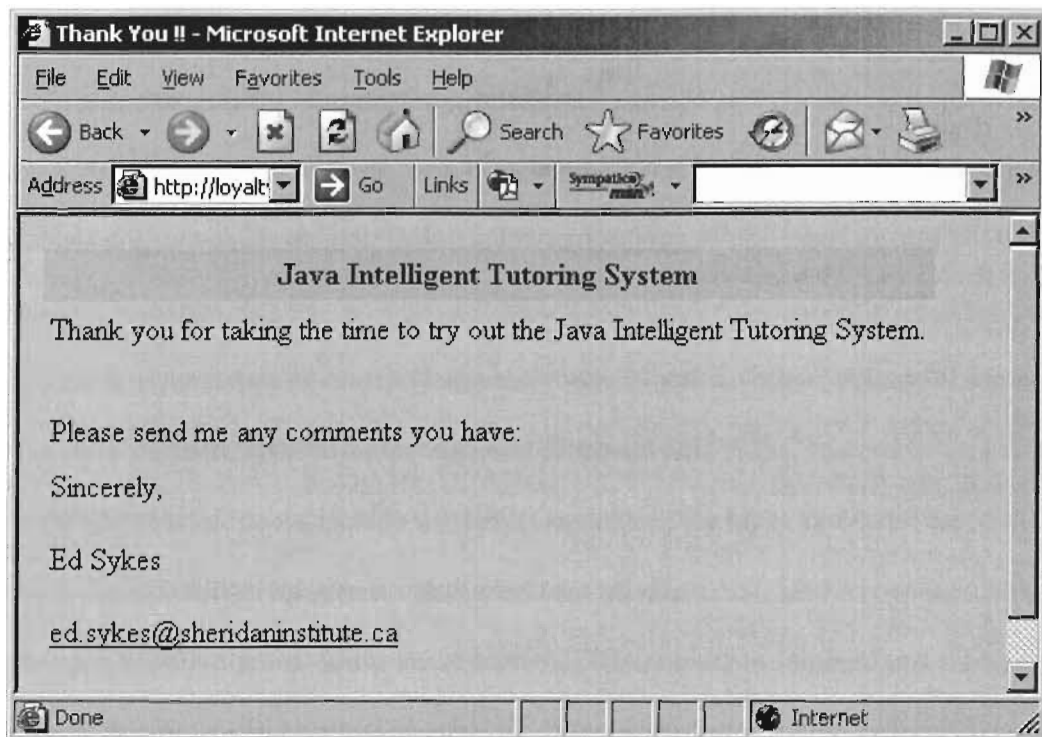
Figure 48. Final version of the JITS User Interface.

the means by which students can see the hints the JITS provides. The “View Solution” button provides potentially various solutions to the current problem. The “Previous Problem” and “Next Problem” buttons are used for navigating within a problem set. The “My Performance” button yields detailed information about the student’s performance including problems solved, problems attempted, the number of attempts for each problem, and comparison information to the “average” JITS student. Links are provided in the “My Performance” output for rapid access to any problem the students wishes to retry. Label 8 shows where the majority of the responses from JITS are presented. Information such as hints, solutions, performance scores, and errors are all shown in this area of JITS.

Label 9 presents the choices of the various programming topics that the student may choose. The “Take Me There” button is used to bring the student to the selected programming topic. Label 10 presents the “View the Tutorial” button, which launches the JITS Tutorial window. The tutorial window may be viewed at the same time as the student is working with the main JITS user interface (i.e., the tutorial may be referenced while working on a problem in JITS). Label 11 shows the “Help Me” button, which opens a separate window displaying the screenshot of JITS with labels to all of the components in JITS. The purpose of this window is to orient new users of JITS so that they feel supported and can more quickly become productive in this Intelligent Tutoring System. The last label (i.e., 12) is the “Exit” button. This button brings up a screen which thanks the student for trying out the system and performs some system-wide cleanup procedures behind the scene. Figure 49 presents the screen once a student exits the system.

## *User Modeling*

The latest version of JITS tracks a great deal of information about the student as s/he works on programming problems in the system. The ultimate goal of gathering this information is to more closely model the student and to more effectively assist the student during the tutoring process.



*Figure 49.* JITS Exit Screen thanking the participant.

The following list describes the information tracked by JITS:

1. time and date when a student logs onto JITS;
2. the number of times the student has connected to JITS;
3. every code submission the student makes on a problem;
4. the number of times the student has tried each problem for each problem the student has tried;
5. the student's solution to a problem;
6. the number and type of misconceptions involving keywords, extended keywords, and identifiers are recorded (e.g., "For," "fro" instead of "for," etc.);
7. whether or not the student pressed the "View Solution" button for a problem;
8. student movement through each Problem Set;
9. student movement to a different topic (i.e., the types and difficulty of problems the student attempts is recorded).

Collectively, this information allows JITS to model the student and effectively engage the student in the tutoring process. When a student exits the system, the next time the student starts JITS, the system brings the student back to the exact state as when s/he left. That is, the problem and code the student was working on are presented to the student, including a detailed message serving to assist in placing the student back in the same mental state when s/he was previously engaged in solving the problem.

## Conclusions

A total of three Program Development Sessions were conducted during this dissertation involving four distinct JITSC groups and five control groups. In all of the experiments, the JITSC students outperformed the corresponding control groups. There were two situations in which the JITSC group performed better than the control group at a statistical significance at the 0.05 level.

During the first term (i.e., May to September 2004) a three-way ANOVA revealed that a larger gap between pretest and posttest occurred early in the semester as compared to the difference late in the semester. This seemed to indicate a more dramatic learning curve took place near the beginning of the semester. The Test by Group interaction showed the superior performance by the JITS group.

Performance score analysis during the second term (i.e., September to December 2004) involving the computation of a 2 x 2, two-way ANOVA revealed the main effect for Test,  $F(1, 92) = 61.12, p < .001$ , which was qualified by a Test by Group interaction,  $F(1, 92) = 5.36, p < .025$ . The Test by Group interaction result is due to the superior performance of the JITS group at posttest.

These results, coupled with the strong positive qualitative feedback from students and instructors, indicate that JITS is beneficial for students learning Java at the beginner level. Due to student involvement and suggestions, JITS includes many features that appeal to many different types of learners. JITS supports multiple intelligences through its effective tutoring approach, interactively rich user interface, and precise user-modeling mechanisms.

## **Contributions to the Fields of Computer Science and Education**

JITS is implemented using advanced e-learning technologies. Its multithreaded distributed architecture makes JITS scalable, robust, and easy to maintain. Through the use of Java ServerPages™, and JavaBeans™, all processing is done at the middle-tier level. The Model-View-Controller design pattern was used to implement JITS. All content is dynamically extracted from an ORACLE database via JDBC and placed into the appropriate Java ServerPage™. From a pedagogical perspective, JITS supports personalized student development by modeling every student in the system. JITS also enhances the learning experience by providing an interactively rich environment where every student receives personalized tutoring—each student receives unique personalized feedback for every situation in the interactive problem solving session. JITS is an online website always available for students and requires only a browser and an internet connection. JITS was designed to be accessible from remote locations and can be effectively used for e-learning for on-site locations as well as remote sites. JITS does not require high-band width internet connectivity; a simple dial-up connection will work equally as fast as a high-band width connection because all of the processing is done on the middle-tier server.

During the design and development of JITS many of these features were unique in the field of ITS research. In other words, no other ITS offers the same degree of “intent” recognition, scalable multithreaded distributed architectural design, extensive personalized instruction, and effective student modeling (Aleven & Ashley, 1997; Graesser et al., 2001; Koedinger, 2001). The performance gain achieved by students who used JITS, it is on par with other ITS (Sykes & Franek, 2004a). The results from



the performance scores indicate a 1.6 standard deviation improvement over traditional classroom environments (Sykes & Franek, 2004a). Most ITS still use traditional system architectural designs which limits the degree of customization to meet student's needs (Graesser et al., 2001; Weber & Brusilovsky, 2001; B. P. Woolf et al., 2001).

Furthermore, most ITS do not offer the same level of user tracking, modeling, and rigor of "intent" recognition that JITS accomplishes with its sophisticated AI JECA module (Albus & Meystel, 2002; Heffernan & Koedinger, 2001; Woolf et al., 2001; Sykes & Franek, 2004b). The Java Error Correction Algorithm is perhaps the most significant contribution of this research because no other ITS is as flexible in being able to guide students towards a potentially unique solution to a programming problem when a solution is not available. This is due to the AI JECA module which rigorously analyzes the student's submission and offers "intelligent" suggestions even though there is no solution for the problem. The term "intelligence" is in reference to the fact that the researcher is working from the perspective of Weak AI not Strong AI. Nonetheless, JITS takes the advice from JECA, combines it with the individual student model, the collective student model, and responds appropriately with personalized feedback for each and every student in the system.

### ***Possible Confounds***

A possible confound associated with the performance score results determined in this study may be attributed to JITS group students having had more "time on task" than the corresponding Control groups. Additionally, due to the on-line accessibility and continuous availability of JITS, it is possible that the JITS students spent more "time on task." However, it is equally reasonable to assume that students in the Control group

spent the same amount of time on coursework activities due to the independent nature of postsecondary education.

Another possible confound is the fact that the JITS groups were receiving more or different attention than Control groups. The Hawthorne effect is a factor that needs to be taken into consideration. The researcher recognizes these possible confounds in the assessment of performance scores in this research project.

On the other hand, it is equally likely that students who identified themselves as being “weak” at programming may have elected to become involved in the research project to gain additional assistance in order to improve their performance in the course.

### **Implications**

Students and instructors alike enjoyed and benefited from working with me in the development of JITS. As is evident in the findings chapter, their involvement in this research was very important in shaping JITS and improving it to meet their needs. Overall, students liked learning in the environment that the Java Intelligent Tutoring System provided. This research has a number of positive implications. Due to the huge amount of student tracking information that JITS collects, there are many different types of reports that may be generated for a variety of reasons. First, personalized student performance reports on individual students using JITS may be created so that each student can reflect on his/her performance and identify areas of improvement. A second implication of the research conducted is from the perspective of the instructor. For instance, instructors may want detailed information on student performance on specific programming topics in JITS. This information could be used to modify

curricula, perform additional classroom instruction, the setting of new tests, etc. This in turn could increase the students' performance in the course.

This research has contributed to the scientific community. For example, the Java Error Correction Algorithm (JECA) is the first of its kind to determine the "intent" of the student's submission by rigorously analyzing the student's code. Behind the scenes, JECA makes changes to the student's submission in order to facilitate this analysis. However, once JECA determines the most reasonable intent of the student, these changes are made known to the student. The results from JECA are passed to the Java Intelligent Tutoring System (JITS) in the form of hints and suggestions, which are then used for instructional purposes. The Java Error Correction Algorithm has appeared in international peer-reviewed journals and international conference proceeding publications. While JECA has contributed to the field of Computer Science, JITS has contributed to the field of Education. The following is a list of publications this dissertation has had in the fields of Computer Science and Education:

### ***Book Publications***

Franek, F., Sykes, E. R. (2007). *Compiler Design: Implementation Using C++ and Java*, Jones and Bartlett Publishers. (in progress)

### ***International Journal Publications***

Sykes, E. R. (2006). Developmental Process Model for the Java Intelligent Tutoring System, *Journal of Interactive Learning Research*. (accepted for publication: issue to be determined)

Sykes, E. R., & Franek, F. (2004). A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java, *International Journal of Computers and Applications*. Vol 1, pp. 35-44, ACTA Press.

### ***International Conference Proceedings Publications***

Sykes, E. R. (2006). Human Computer Interaction in the Java Intelligent Tutoring System, *Fifth IASTED International Conference on Web-Based Education*, Puerto Vallarta, Mexico. (accepted)

- Sykes, E. R., & Mirkovic, A. (2005). A Fully Parallel Implementation of a Scalable Hopfield Neural Network on the SHARC-NET Supercomputer, *The 19th International Symposium on High Performance Computing Systems and Applications*, University of Guelph, Ontario, Canada.
- Sykes, E. R., & Franek, F. (2004). Presenting JECA: A Java Error Correcting Algorithm for the Java Intelligent Tutoring System, *Proceedings of the IASTED International Conference on Advances in Computer Science and Technology*, St. Thomas, Virgin Islands, USA (pp. 151-156).
- Sykes, E. R., & Franek, F. (2004). Field-Report of the Java Intelligent Tutoring System, *Learning Technology Newsletter, Publication of IEEE Computer Society Technical Committee on Learning Technology* (ISSN 1438-0625) (pp. 32-35).
- Sykes, E. R., & Franek, F. (2004). Preliminary Assessment of the Java Intelligent Tutoring System, *International Conference on Education and Information Systems, Technologies and Applications*, Orlando, Florida. (pp. 22-27).
- Sykes, E. R., & Franek, F. (2004). Pedagogical Design of the Java Intelligent Tutoring System, *International Conference on Education and Information Systems, Technologies and Applications*, Orlando, Florida. (pp. 213-218).
- Sykes, E. R., & Franek, F. (2004). Inside the Java Intelligent Tutoring System Prototype: Parsing Student Code Submissions with Intent Recognition, *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education. Innsbruck, Austria.* (pp. 613-618).
- Sykes, E. R., & Franek, F. (2003). An Intelligent Tutoring System Prototype for Learning to Program Java. *Proceedings of the 3<sup>rd</sup> IEEE International Conference on Advanced Learning Technologies, Athens, Greece,* (pp. 485-486).
- Sykes, E. R., & Franek, F. (2003). A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education. Rhodes, Greece.* (pp. 78-83).
- Sykes, E. R. (2003). Java Intelligent Tutoring System Model and Architecture. *AAAI Spring Symposium: Human Interaction with Autonomous Systems in Complex Environments, SS-03-04,* (pp. 187-193). AAAI Press.
- Sykes, E. R. (2002). A Unified Model of Intelligence. *Canadian Society for the Study of Education Press* (pp. 537-545). Toronto, Canada: CSSE Press.
- Sykes, E. R. (2002). *An Intelligent Academic Advising System Model Using Soft Computing Constructs.* Paper presented at the meeting of Computer Science Faculty, St. Catharines, Ontario, Canada.

## Recommendations

Throughout this dissertation, a number of innovative ideas and breakthroughs were accomplished. This is demonstrated by the number of international publications the

researcher has to date on the Java Intelligent Tutoring System. It would be advantageous to conduct more educational research by field-testing JITS in environments where the students would use JITS as a core component of their course. For example, integrating course assignments, exercises, and projects by using JITS may reveal interesting results.

Other areas of recommendation include research relating to relevant education and scientific research communities such as gaming elements, video-streaming, more sophisticated user modeling, and more intelligent feedback mechanisms. The researcher intends on continuing researching and publishing to further promote the Java Intelligent Tutoring System.

## References

- Albus, S. J., & Meystel, M. A. (2002). *Intelligent Systems*: Wiley.
- Aleven, V., & Ashley, D. K. (1997). Case-Based Argumentation Through a Model and Examples: Empirical Evaluation of an Intelligent Learning Environment. In B. Du Boulay & R. Mizoguchi (Eds.), *International Journal of Artificial Intelligence in Education* (pp. 87-94). Amsterdam: IOS Press.
- Anderson, J. R. (1998). Production Systems and the ACT-R Theory. In P. Thagard (Ed.), *Mind readings: Introductory selections on cognitive science* (pp. 59-76). Cambridge, MA: MIT Press.
- Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive Modelling and Intelligent Tutoring. *Artificial Intelligence*, 42, 7-49.
- Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill Acquisition and the Lisp Tutor. *Cognitive Science*, 13(4), 467-505.
- Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons learned. *The Journal of the Learning Sciences*, 4, 167-207.
- Anderson, J. R., & Pelletier, R. (1991). *A Development System for Model-Tracing Tutors*. Paper presented at the The International Conference on the Learning Sciences, Northwestern University, Evanston, Illinois, USA.
- Anderson, J. R., & Reiser, B. J. (1985). The LISP tutor. *Byte*, 10, 159-175.
- Beal, R. C., Beck, E. J., Woolf, B. P., & Rae-Ramirez, A. M. (1998). *WhaleWatch: An Intelligent Model-Based Mathematics Tutoring System*. Paper presented at the Proceedings of the Fifteenth IFIP World Computer Congress, Austria.

- Bloom, S. B. (1984). The 2-sigma problem: The search for methods of group instruction as effective as one-to one- tutoring. *Educational Researcher*, 13, 4-16.
- Boyd, M. (2003). *Center for instructional technologies*. Retrieved February 10, 2003, from <http://www.utexas.edu/cc/webct/about/atut/coursetool/prodrevs.html>
- Burke, M. G., & Fisher, G. A. (1987). A practical method for LR and LI syntactic error diagnosis and recovery. *ACM Transactions on Programming Languages and Systems*, 9(2), 164-197.
- Chen, E. (2004). *Java: A False Sense of Security?* Retrieved Nov 10, 2004, from <http://www.trendmicro.com/en/about/news/coverage/eva-chen.htm>
- Conati, C., & Van Lehn, K. (1999, 19-22 July). *Teaching Meta-Cognitive Skills: Implementation and Evaluation of a Tutoring System to Guide Self-Explanation while Learning from Examples*. Paper presented at the Ninth World Conference of Artificial Intelligence and Education, Le Mans, France.
- Cottingham, J., Stoothoff, S., Murdoch, A., & Kenny, A. (1991). *The philosophical writings of Descartes*: Cambridge.
- de Koning, K., & Bredeweg, B. (2001). Exploiting Model-Based Reasoning in Educational Systems: Illuminating the Learner Modeling Problem. In K. D. Forbus & P. J. Feltovich (Eds.), *Smart Machines in Education*. Cambridge, MA.: MIT Press.
- Fischer, C., & LeBlanc, R. J. (1991). *Crafting a compiler with C*. Redwood City, CA: Benjamin Cummings Publishing.
- Fletcher, J. D. (1995). *What Have We Learned about Computer-Based Instruction in Military Training?* Brussels, Belgium: North Atlantic Treaty Organization.

- Gilbert, F. (2003). *Lakehead University and the Double Cohort*, from <http://www.lakeheadu.ca/cohort/main.html>
- Graesser, A. C., & Person, N. K. (1994). Question asking during tutoring. *American Educational Research Journal*, 31, 103-137.
- Graesser, A. C., Person, N. K., & Harter, D. (2001). Teaching tactics and dialog in autotutor. *International Journal of Artificial Intelligence in Education*, 12, 12-23.
- Heffernan, N. T., & Koedinger, K. R. (2001). *The Design and Formative Analysis of a Dialog-Based Tutor*. Paper presented at the AI in Education 2000 Workshop on Building Dialogue Systems.
- Karolyi, J. (2004). Practice makes perfect. In E. R. Sykes (Ed.) (pp. Personal communication with instructor, John Karolyi at the Sheridan Institute of Technology and Advanced Learning.). Oakville.
- Koedinger, K. R. (2001). Cognitive tutors. In K. D. Forbus & P. J. Feltovich (Eds.), *Smart machines in education* (pp. 145-167). Cambridge, MA: MIT Press.
- Martinez, A. (2002). *Information Technology – Computer And Data Programming Services: Computer Programmers*. Retrieved January 4, 2003, from [http://www.cteresource.org/ipg/certifications/sun\\_jpj2.html](http://www.cteresource.org/ipg/certifications/sun_jpj2.html)
- O'Reilly, R. C., & Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience*. London, England: MIT Press.
- Pawlan, M. (2004). *J2EE Tutorial*. Retrieved October 15, 2004, from <http://java.sun.com/j2ee/1.3/docs/>
- Regian, F. D. (1997). Increased performance gains in individualized human tutoring. *IEEE: Intelligent Systems*, 4, 14-29.



- Rowe, C. N., & Galvin, P. T. (1998). An authoring system for intelligent procedural-skill tutors. *IEEE: Intelligent Systems*, 14, 61-69.
- Schofield, J. W., Evans-Rhodes, D., & Huber, B. R. (1990). Artificial Intelligence in the Classroom: The Impact of a Computer-Based Tutor on Teachers and Students. *Social Science Computer Review*, 8(1), 24-41.
- Searle, J. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3, 417-424.
- Seidel, R. L., & Perez, R. S. (1994). An evaluation model for investigating the impact of innovative educational technology. In H. F. O'Neil, Jr. & E. L. Baker (Eds.), *Technology Assessment in Software Applications* (Vol. 4, pp. 235-256). Los Angeles, California: Graduate School of Education and Human Behaviors.
- Sykes, E. R. (2003). Java Intelligent Tutoring System Model and Architecture. *Proceedings of American Association of Artificial Intelligence Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, 187-193.
- Sykes, E. R., & Franek, F. (2003). A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education*, 78-83.
- Sykes, E. R., & Franek, F. (2004a). *Preliminary Assessment of the Java Intelligent Tutoring System*. Paper presented at the International Conference on Education and Information Systems, Technologies and Applications, Orlando, Florida, USA.
- Sykes, E. R., & Franek, F. (2004b). *Presenting JECA: A Java Error Correcting Algorithm for the Java Intelligent Tutoring System*. Paper presented at the

IASTED International Conference on Advances in Computer Science and Technology, St. Thomas, Virgin Islands, USA.

Sykes, E. R., & Franek, F. (2004c). A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. *International Journal of Computers and Applications*, 1, 35-44.

Tracey, W., R. (2003). *The Human Resources Glossary: The Complete Desk Reference for HR Executives, Managers and Practitioners* (Third ed.): CRC Press.

Vasilevsky, K. E. (2003). *Personal Consumption as the Driving Force of American and Canadian Economy*. Retrieved February 10, 2003, from <http://iskran.iip.net/engl/mag/june.html>

Weber, G., & Brusilovsky, P. (2001). ELM-ART: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12, 129-145.

Wertheimer, B. P. (1990). The Geometry Proof Tutor: An Intelligent Computer-Based Tutor in the Classroom. *Mathematics Teacher*, 83(4), 303-317.

Woolf. (1992). AI in education. In S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence* (pp. 434-444). New York: Wiley.

Woolf, & Hall. (1995). Multimedia Pedagogues: Interactive Systems for Teaching and Learning. *IEEE: Computer*, 28(5), 74-80.

Woolf, B. P., Beck, J., Eliot, C., & Stern, M. (2001). Growth and maturity of intelligent tutoring systems: A status report. In K. D. Forbus & P. J. Feltovich (Eds.), *Smart machines in education* (pp. 100-144). Cambridge, MA: MIT Press.

### **Appendix Brock Ethics Approval**

The following section contains Brock University Ethics Approval form for this research.

**Brock University  
Senate Research Ethics Board  
Extensions 3943/3035, Room AS 302**

**DATE:**           **January 26, 2004**

**FROM:**           Joe Engemann, Chair  
  
                  Senate Research Ethics Board (REB)

**TO:**              Rosemary Young, Education  
  
                  Edward R. Sykes

**FILE:**           **03-262 Sykes**

**TITLE:**           **Java Intelligent Tutoring System Project**

The Brock University Research Ethics Board has reviewed the above research proposal.

**DECISION:**   **Accepted as Clarified**

This project has been approved for the period of **January 26, 2004** to **December 18, 2004** subject to full REB ratification at the Research Ethics Board's next scheduled meeting. The approval may be extended upon request. *The study may now proceed.*

Please note that the Research Ethics Board (REB) requires that you adhere to the protocol as last reviewed and approved by the REB. The Board must approve any modifications before they can be implemented. If you wish to modify your research project, please refer to [www.BrockU.CA/researchservices/forms.html](http://www.BrockU.CA/researchservices/forms.html) to complete the appropriate form ***REB-03 (2001) Request for Clearance of a Revision or Modification to an Ongoing Application.***

Adverse or unexpected events must be reported to the REB as soon as possible with an indication of how these events affect, in the view of the Principal Investigator, the safety of the participants and the continuation of the protocol.

If research participants are in the care of a health facility, at a school, or other institution or community organization, it is the responsibility of the Principal Investigator to ensure that the ethical guidelines and approvals of those facilities or institutions are obtained and filed with the REB prior to the initiation of any research protocols.

The Tri-Council Policy Statement requires that ongoing research be monitored. A Final Report is required for all projects, with the exception of undergraduate projects, upon completion of the project. Researchers with projects lasting more than one year are required to submit a Continuing Review Report annually. The Office of Research Services will contact you when this form ***REB-02 (2001) Continuing Review/Final Report*** is required.

Please quote your REB file number on all future correspondence.

Deborah VanOosten, Research Ethics Officer  
*Brock University*  
*Office of Research Services*  
*500 Glenridge Avenue*  
*St. Catharines, Ontario, Canada L2S 3A1*  
**phone:** (905)688-5550, ext. 3035    **fax:** (905)688-0748  
**email:** [deborah.vanoosten@brocku.ca](mailto:deborah.vanoosten@brocku.ca)  
<http://www.brocku.ca/researchservices/humanethics.html>

**FROM:** Linda Rose-Krasnor, Chair  
Research Ethics Board (REB)

**TO:** Rosemary Young, Education  
Sykes, Edward

**RE:** **Continuing Review/Final Report**  
*File #: 03-262 - Sykes, Edward*  
*Originally Accepted: January 26, 2004*  
Date of Completion: **December 18, 2004**

**DATE:** 1/14/2005

Thank you for completing the *Continuing Review/Final Report* form. The Brock University Research Ethics Board has reviewed this report for:

***Java Intelligent Tutoring System Project***

The Committee finds that research participants are no longer being studied or followed on the above protocol and therefore, this protocol is officially terminated by the Research Ethics Board.

**\* Final Report Accepted.**

LRK/hb

**Heather Becker, Office of Research Ethics**

Brock University

Office of Research Services

500 Glenridge Avenue

St. Catharines, Ontario, Canada L2S 3A1

phone: (905)688-5550, ext. 3035 fax: (905)688-0748

email: [hbecker@brocku.ca](mailto:hbecker@brocku.ca)

[http://www.brocku.ca/researchservices/Certification&Polices/Certification&Polices\\_Human\\_Ethics.html](http://www.brocku.ca/researchservices/Certification&Polices/Certification&Polices_Human_Ethics.html)