

## jTutors: A Web-based Tutoring System For Java APIs

## AN ABSTRACT OF THE PROJECT OF

Aniket Dahotre for the degree of Master of Science in Computer Science presented on March 9, 2011.

Title: jTutors: A Web-Based Tutoring System For Java APIs.

Abstract approved: \_\_\_\_\_

Christopher Scaffidi

For building robust software applications, it is important for the software engineer to make efficient use of the available building blocks. Learning the basic language constructs is only the first step in this process. It is becoming increasingly important for software engineers, especially students, to get acquainted with the available Application Programming Interfaces (API) and the ways to efficiently use them. With the ever increasing number of APIs, it becomes difficult for teachers to expose students to most of the APIs. Hence it becomes important to complement the traditional methods of instruction, such as lectures or API documentation, with techniques that would ease learning by software engineering students.

We have leveraged the freely available code examples on various Internet communities, blogs and software project hosting websites to create an intelligent tutoring system, jTutors, which would aid the teacher in exposing students to multitudes of the available Java APIs. jTutors uses these code samples to generate intelligent tutorials. We evaluated jTutors by performing realistic test cases and a heuristic assessment.

©Copyright by Aniket Dahotre

March 9, 2011

All Rights Reserved

## ACKNOWLEDGEMENTS

I would like to thank my family for supporting my ambitions and bestowing confidence in me.

I will forever be grateful to my advisor, Dr. Christopher Scaffidi for his guidance from day one when he gave me the book "Getting what you came for." I would like to thank him for his advice about research, academics and about being a better professional, and his interest in my learning and development.

Oregon State University has provided excellent facilities and an encouraging atmosphere for studying and research. I would like express my gratitude for an opportunity to learn from such a wonderful faculty. I'd like to give a special thanks to Dr. Timothy Budd & Dr. Ron Metoyer for giving me an opportunity to assist them in their data structures courses.

Vasanth & Matt kept up the team work and contributed brilliant ideas while implementing jTutors, as did Yan on another project, so thank you to them.

I would like to acknowledge all my friends who made each day at OSU a wonderful memory. Thank you to Aarti, Atipol, Crystal, David, Deepthi, Lam, Madhura, Prashanth, Sean, Shalini, and Vignesh for the great combined effort on various, exciting projects during the entire term at OSU. I appreciate all the feedback on jTutors and other topics from Shubhomoy, Akhil, and Karthick. Thanks to Harsha for helping me edit the final draft of the report. Thanks to Vivek for being such a cooperative roommate and all the interesting discussions. Thank you to many others for being such wonderful friends.

## TABLE OF CONTENTS

1	Introduction.....	1
2	Related Work And Background .....	3
2.1	Tutoring systems .....	3
2.2	Help with APIs .....	5
2.3	Searchable code repository.....	7
3	Approach .....	10
3.1	jTutors sends teacher’s query to third-party search engines and filter results.....	12
3.2	jTutors presents code snippets to the teacher .....	14
3.3	Teacher inspects selected snippets.....	14
3.4	Teacher edits the snippet.....	16
3.5	Teacher creates quizzes .....	17
3.6	Teacher describes the tutorial .....	18
3.7	jTutors packages the intelligent tutorial .....	19
3.8	Student accesses the intelligent tutorial.....	19
3.9	Design decisions on the basis of cognitive dimensions .....	19
4	Implementation Details.....	22
4.1	Technology Used .....	22
4.2	Strategies Used.....	23

4.2.1	Sanitizing the input .....	24
4.2.2	Selecting tutorials .....	26
4.2.3	Suggesting words to be blanked for quizzes.....	30
4.2.4	Adaptive testing .....	31
5	Evaluation .....	33
5.1	Snippet search results for typical queries.....	33
5.1.1	Criteria.....	34
5.1.2	Results:.....	35
5.2	Heuristic assessment.....	38
6	Conclusion And Future Work.....	46
7	Bibliography.....	49

## LIST OF FIGURES

Figure 1. Work flow for a teacher using jTutors .....	11
Figure 2. Search a topic.....	13
Figure 3. Scraping code from URLs .....	13
Figure 4. Creating a fill-in-the-blank .....	15
Figure 5. Compiler messages and modified code .....	16
Figure 6. Edit the snippet .....	17
Figure 7. Add new word .....	18
Figure 8. Teacher describes the tutorials .....	18
Figure 9. Students' interaction and Java Web Start Application .....	21
Figure 10. MVC architecture for retrieving and displaying each tutorial (model) .....	23
Figure 11. Code snippets mapped to HTML tags .....	25
Figure 12. CompilationUnitStore .....	26
Figure 13. Code Sample .....	28
Figure 14. Representation of code as vectors .....	28
Figure 15. Sample of queries that programmers have posted to Mica.....	33
Figure 16. Snippets returned by specific versus general queries .....	38
Figure 17. Example of reading XML file .....	41
Figure 18. Explaining use of methods in JDBC API.....	43
Figure 19. Snippet showing print statement. Part of figure 5. ....	45

## 1 Introduction

Java has been used as a programming language in introductory computer science courses at many educational institutions, including Oregon State University. It offers many advantages over previously used languages like Pascal, C, and C++ with support for high level concepts such as Object Oriented Programming and web development. In order to implement such high-level concepts, students need to master an ever increasing set of Application Programming Interfaces (API) that comes along with the Java SDK. Along with the standard API it is also important for a software engineer to be acquainted with APIs provided by third parties like Google (<http://code.google.com/more/>) and Apache (<http://commons.apache.org/>).

With the growing sophistication of Java as a programming platform more time is required for teaching which makes it difficult to cover necessary material in the traditional two term sequence [25]. Students and teachers suffer from conceptual overload due to the existence of large number of available Java APIs [24]. With each new release of Java, this problem of scale is bound to increase. The latest edition (2010) of “Building Java programs” the textbook recommended on the course catalog website of Oregon State University, for the introductory course to computer science, is 1176 pages long as compared to 896 pages for the previous edition from 2007. Although the increased sophistication of Java is a good sign for advancement of software engineering, it proves to be a daunting task for teachers to choose necessary APIs to help the students to build useful programs that are relevant to the recent developments.



This problem has been analyzed by various researchers in academia. The Computing Curricula 2001 report recommends adoption of a three-course introductory sequence rather than a two-course [24] sequence. The ACM Education Board initiated the ACM Java Task Force in 2004 to review Java language and develop a stable collection of pedagogical resources for teaching Java in introductory courses. The final report by this task force went a step ahead of previous recommendations by suggesting the use of an altered version of select Java APIs [24]. This might have reduced the burden on teachers by limiting the resources to be taught. But using an alternate version of APIs will certainly not help the students in keeping their knowledge up-to-date with latest developments. They will still be required to learn the actual APIs while writing applications in the real world.

Results from a prior survey [26] revealed that 40% of the people facing problems with learning APIs could not find adequate examples to teach the needed concepts. We have developed a system that assists in creating instructional material by mining the web for examples of API usage. jTutors helps to rapidly author these materials by leveraging code snippets and programs available freely on the World Wide Web. This semi-automatic approach adopted in jTutors might save the teacher's efforts of authoring various examples and quizzes to teach and test student's knowledge of Java APIs. Teachers are expected to share the authored material on the jTutors website, which could be accessed by the students to learn about Java APIs and their common usage patterns in the real world. jTutors might be used as a complement to the traditional teaching procedure to cover a larger portion of the Java APIs than can be done currently. User evaluation of this system can help to trigger a platform where teachers and students from different institutions could share their knowledge of Java APIs.

## 2 Related Work And Background

There are various approaches that have tackled the issues addressed by jTutors. These issues can be broadly classified as follows:

- a) Tutoring systems
- b) Help with APIs
- c) Searchable code repository; creation and maintenance

### 2.1 Tutoring systems

Many tutoring systems provide a set of instructions to the student and record their answers. Such systems are still popular and used widely for variety of educational purposes like test preparations (e.g. Kaplan Test Prep <http://www.kaptest.com/>) as well as interactive systems for teaching vocabulary and arithmetic [32]. Such systems, called computer assisted instructional systems (CAI) lack adaptability for individual student needs and qualitative evaluation. With the evolution of artificial intelligence, these CAI systems adapted to the student's actions and knowledge. Such systems were then termed as Intelligent Tutoring Systems (ITS) [32]. Generally, a ITS 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 [27]. Modern ITSs analyse what mistake was committed by the student as well as the inferred intent behind the mistake, and they provide hints that can help the student to understand the concept rather than merely rectifying the error.

ITSs have been used in training in various domains like Geometry [17], language learning [10] and commercial games (V-CTC) [20]. V-CTC, Virtual Combat Training Center simulator was originally intended for classroom training for the military but was adapted to enhance the gaming experience of Armored Task Force, which is a commercial strategy war game. ITS has been used extensively in teaching programming languages as well. LISP tutoring system (LISPITS) models the steps required to write a LISP program [3] [30]. The system then compared the steps taken by the students to these ideal steps. SQL T-Web is a web-based ITS for teaching SQL [19] that uses constraint based modeling to form models of the student users. The student model is updated every time the student submits a solution and generates feedback by comparing it with the available ideal solution. New problems are given to the student on the basis of this comparison.

JITS [31] and C-Tutors [28] are aimed at finding the intention of the student in committing errors and generating feedback. C-Tutor is an intelligent tutoring system for novice C programmers. Java Intelligent Tutoring System (JITS) was designed to recognize small Java programs and provide intelligent feedback even when there is no authored solution available. It requires the teacher to describe a problem statement, program specification, required output and a code template to be filled in later by the student. This system employs an algorithm that is able to compile a Java program and analyze the errors or multiple ways of solving the problem to provide feedback to the student. Although we were not able to use the system, various screenshots and descriptions of the system revealed that the teacher is required to provide several inputs along with writing the entire program solution on her own. Also, the program should be arranged in a manner that a large blank is left for the students that should

complement the rest of the code written by the teacher above and below the code. In order to allow students to come up with different versions of the solution, it is necessary for the teacher to write the solution template in such a manner too.

Cognitive Tutor Authoring Tools (CTAT) is a platform to provide a rich set of options for efficient authoring of tutorials and supporting controlled experiments [1]. CTAT can be used for authoring example-tracing tutorials that can be delivered to students via Internet using Adobe Flash or Java Web Start. It does not require the teacher to have any knowledge of programming but rather requires her to follow certain steps to build tutorials by demonstration. The teacher has to install the programming environment (either Adobe Flash or Net beans IDE for Java). Then, she has to create the tutorial by utilizing the GUI components provided by CTAT in form of libraries. Then, she has to open this tutorial in CTAT's client software and demonstrate the solution. As the final step, she has to package both the files generated by CTAT (the tutorial and recorded solution) into a Java Web Start application that can be hosted on the teacher's server.

Although ITS have been used for teaching programming language constructs, it has not be utilized for teaching APIs. We have discussed why the problem of learning APIs is important. So we decided to use CTAT as a platform to author and deliver tutorials related to APIs. jTutors avoids the need for teachers to come up with examples. Thus, with jTutors, the teachers could leverage the time-tested ITS authoring tool with the help of nothing more than an Internet browser.

## **2.2 Help with APIs**

There has been a great deal of research on simplifying the understanding of APIs and designing APIs effectively. A study of developers at Microsoft revealed statistics and details about some of these issues [26]. Approximately 40% of the programmers who faced problems learning APIs could not find adequate examples. Moreover, 28% had problems with understanding the documentation and 18% could not understand how to use an API once they identified it. Another study categorized the learning barriers faced by programmers with respect to the environment and libraries [15]. jTutors helps in finding several examples that explain the use of Java APIs and common ways in which multiple APIs are used together.

Strathcona [12] uses the developer's current context and past projects in the repository to recommend relevant examples. It involves automatic query creation on the basis of the developer's current working context in the Eclipse IDE. The repository on a different server machine is queried which returns relevant example usages. The user can then browse through the returned samples and use whatever example seems suitable. Our approach of returning multiple code snippets and letting the teacher choose from them was motivated by Strathcona.

Jadeite [29] uses results from Google to search for common usage patterns of Java constructors. It uses regular expressions to match these patterns and then displays the most common way to construct a new instance of the Java class in question. This idea of helping users understand instance creation was one of the motivations in our research. Other similar systems include Assieme [11] and MICA [30].

Exemplar [8] takes a different approach on assisting programmers by searching executable applications for rapid prototyping. Semantics-based code search [22] lets the user query keywords along with finer details like method signature and satisfying test cases.

The information and ideas gained from these projects served as a starting point for addressing the problem of teaching APIs in a university setting. The technique of matching only by regular expressions seemed insufficient for recommending code snippets. So we broadened the approach by using more general heuristics. The knowledge of how to create an instance (as in Jadeite [29]) was considered only as one step in understanding the API. So we also support other APIs besides constructors. For understanding the common methods, it seemed important to provide the users with some context of the program. Hence we analyzed the Abstract Syntax Tree (AST) of the code snippets found on the web pages returned by web searches and found common method usages too.

### **2.3 Searchable code repository**

Open Source Software like Apache, Eclipse, OpenOffice.org, etc. provide their source code freely on the Internet. It can be accessed either via source control repositories (e.g.

<http://svn.apache.org/viewvc/>) or direct browsing (all projects hosted at

<http://code.google.com/>).

Google code search [7] does a great job of retrieving code from source control of these open source softwares. Various programming languages are supported by the search. It mainly searches for code from online repositories like Google code, Apache.org, etc. Although this helps in finding code samples that are otherwise not accessible through a normal web search, it

returns highly complex programs. Also, it returns many results pertaining to the actual implementation of the API. These complex code samples are difficult to understand for a student who is trying to learn the use of the API.

Jexamples [14] is a website dedicated for searching through Java open source projects. Source analysis is employed to retrieve examples from these projects. User ratings play an important role in the way these results are displayed. Information regarding complexity of the code is also provided by the source analysis. The advantage of these two methods is that it displays a snippet from a larger program file that matches the query. Although this feature is useful, it results in loss of contextual information. In spite of the complexity rating, Jexamples also suffers from the problem of complex code. The site lacks the breadth of examples for a topic that might interest a novice Java programmer (see section 5 on evaluation). Also, it does not support naïve phrasing that is important when the teacher aims to help in building a useful application rather than choosing a particular class or method from the API.

Another project, SourcererDB [21] statically analyzes code from open source projects and stores it in a relational database. Strathcona [12] also maintains its own repository that can be accessed by its users. Users can also add their projects to this repository using an additional tool. Other websites like krugle [18] and Koders [16] index code from open source repositories.

Lastly, work [5, 28, 29] by researchers at Carnegie Mellon University, used the entire web as repository. We found the results and basis of their formative studies [30] more relevant to the problem we were addressing. Also, the examples of API usage from open source repositories were not representative of how a typical programmer might use the API. Hence we

used the code snippets found on web pages returned by search results of general purpose Internet search engines like Yahoo and Bing.



### 3 Approach

Modern Intelligent Tutorial Systems (ITSs) concentrate on representing the instructional domain, building a student model, and providing adequate help and instructions [2]. Such systems employ artificial intelligence to help learners in solving problems by analyzing their feedback, tracking errors, providing assistance and building an understanding of the learner's knowledge level. Section 2.1 contains detailed discussion on the topic of ITS. We have leveraged the Cognitive Tutors Authoring Tools (CTAT), a toolkit for delivering such intelligent tutorials. It provides a way to assist the student with hints and feedback about correctness of the solution. It also logs the interaction of the student with the tutorial. Our research contribution through jTutors is to provide a system that semi-automatically generates intelligent tutors for common Java APIs. ITSs have been used in the past to teach programming languages such as Java (see section 2.1). Also there have been many attempts to simplify usage of Java APIs (see section 2.2). We believe that the combination of these two methods will be a valuable contribution to academia as well as the API community.

We designed jTutors with an aim to help teachers create study material with examples from the real world in a short time. jTutors analyzes the structure of the code samples found on the Internet and creates tutorials that can be viewed out of the browser by the students as Java Web Start applications. Instead of relying on specialized code repositories (see section 2.3) we decided to rely on search results returned by general purpose search engines. jTutors analyzes the code snippets found and makes intelligent choices about which snippets are useful for teaching an API. The teacher reviews these choices and adds extra information to the tutorials from personal expertise. These choices are then saved as a new ITS constructed CTAT. The work

flow of jTutors has been explained in the figure 1. The details of implementation of these artifacts have been explained in detail in section 4.

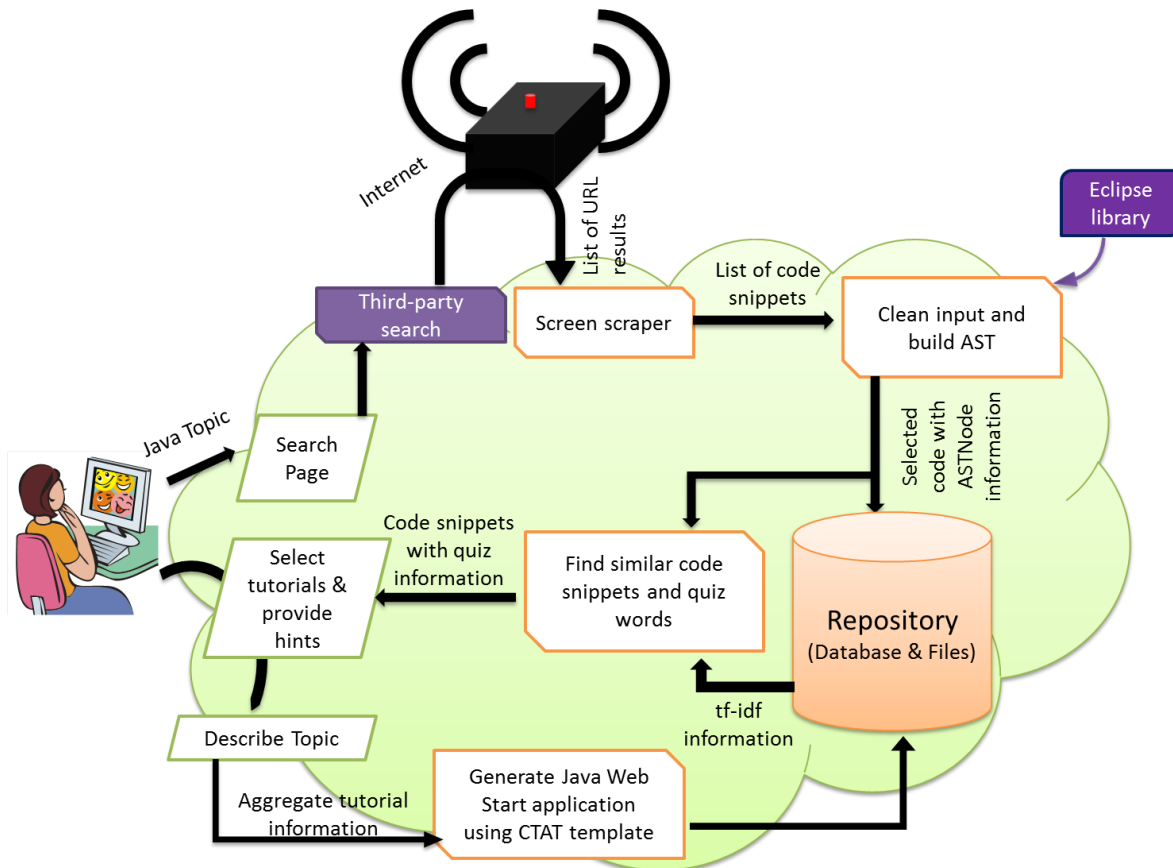


Figure 1. Work flow for a teacher using jTutors

Another important part of jTutors is to allow students to access the tutorials created by their teacher and learn about the Java APIs. Once the student selects a topic to learn, he can read the topic description written by the teacher. After reading the description the student can begin using the tutorial, which is a combination of examples and quizzes. Each example in the tutorial highlights the terms related to the chosen API. Then, a quiz allows the student to test

his knowledge of the API by filling the blanks. Quizzes are marked with different difficulty levels by the teacher. This enables jTutors to provide quizzes to the student on the basis of his answers. For example, if the student struggles to answer a quiz, jTutors will provide an example or another quiz with lower difficulty level. This type of adaptive testing combined with learning by example and intelligent feedback might help to lower the steep learning curve experienced by students.

jTutors has been implemented as a website developed in Java EE. A client server architecture thus enables us to store all the study material generated by the teacher to be stored at a central repository that could be accessed by a student or teacher from anywhere with an Internet connection. This helps the repository to grow continuously (see section 4.2.2 for details) which could ultimately support intelligent choices of selection of study material.

The following sub-sections will explain the flow of events as see in figure 1, with an example. Sub-section 3.9 explains the rationale behind some design decisions.

### **3.1 jTutors sends teacher's query to third-party search engines and filter results**

The teacher decides to create a topic using jTutors to teach how to connect to a database in Java. The teacher types "Connect to database using JDBC java example" and clicks the Search button (figure 2). jTutors calls third party search engines to search the World Wide Web. jTutors currently uses Yahoo's BOSS API (<http://developer.yahoo.com/search/boss/>) and Bing API (<http://www.bing.com/developers>). Both of the search engines are requested to send their top 30 results. A union of these search results is passed to a screen scraper program. It parses the html source of each of these result links to obtain code snippets (see figure 3).

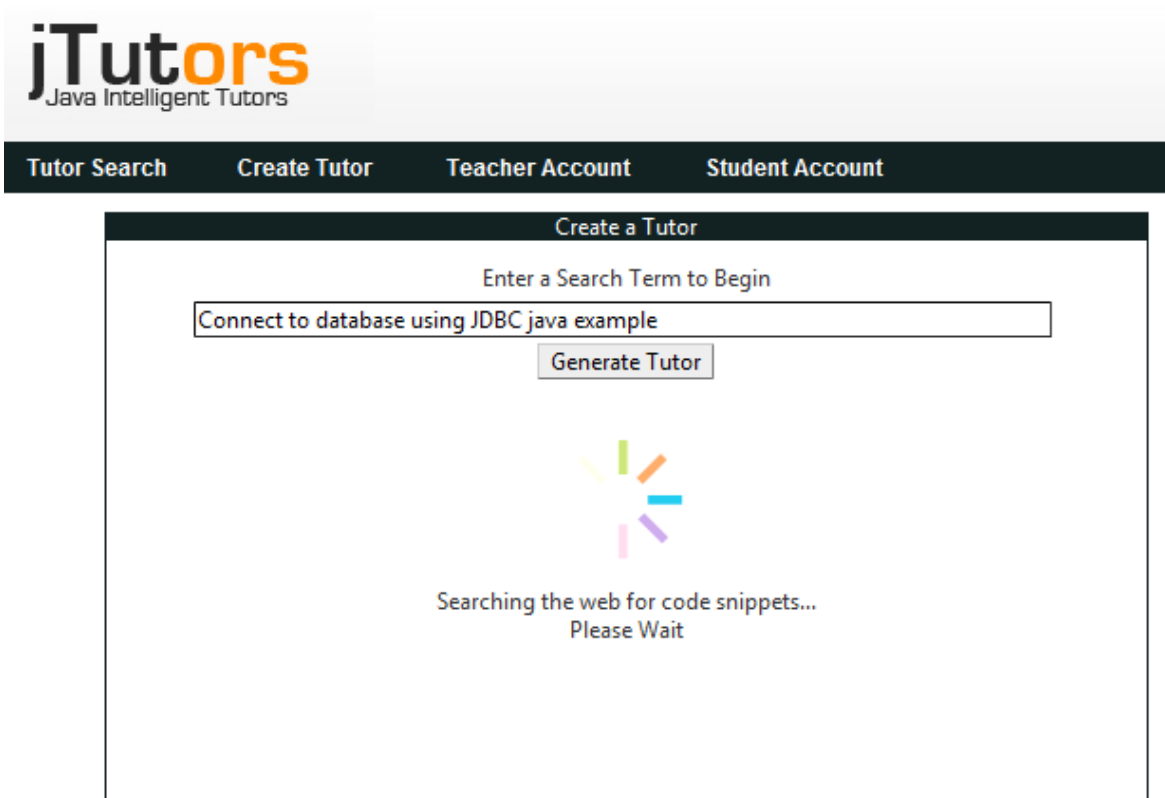


Figure 2. Search a topic

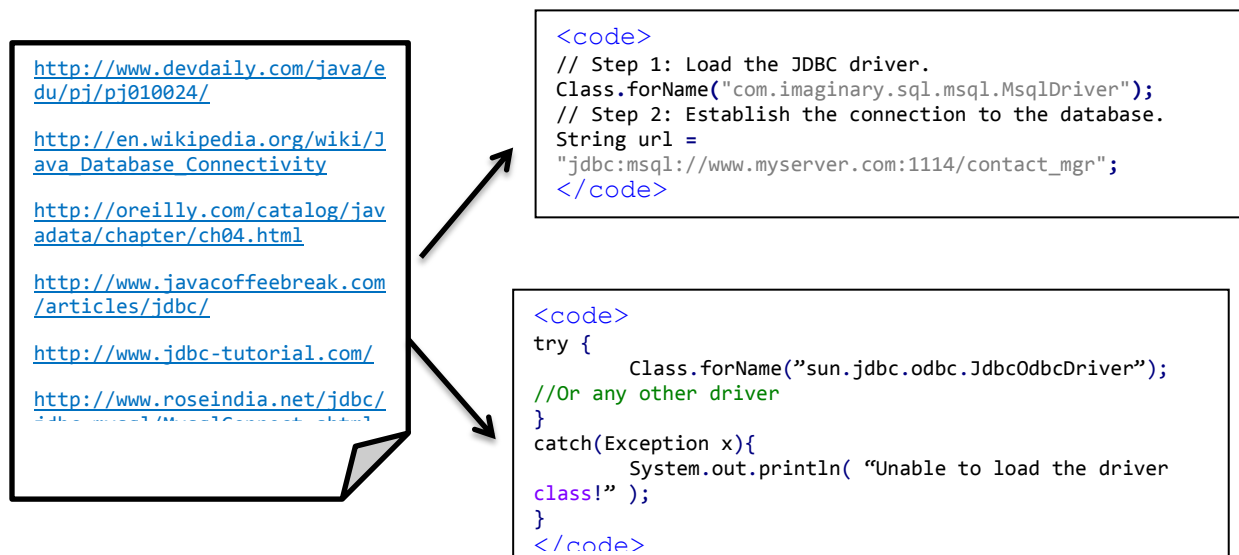


Figure 3. Scraping code from URLs

### 3.2 jTutors presents code snippets to the teacher

jTutors intelligently chooses and arranges these code snippets in the order of relevance. It also selects words from the code snippets that describes important properties and usages of the API found in the code snippet. These words are used later for code highlighting and creating fill-in-the-blank type quizzes. jTutors uses the data stored in the repository to measure the relevance of these code snippets. See section 4.2.2 and 4.2.3 for details on how jTutors makes these selections.

### 3.3 Teacher inspects selected snippets

The teacher is provided with the list of selected code snippets. The teacher judges whether the tutorial is fit for creating a quiz or provide it as an example of API usage. The snippet can be customized by clicking the edit radio button. The teacher can also discard the snippet if she does not find it useful (see “step 1” in figure 4). It can be seen that in this case the teacher has decided to create a quiz from the provided snippet by selecting the corresponding radio button.

Because we use an error-tolerant parser for parsing the snippets, there is a possibility that the code in the snippet won't compile successfully or it might be incomplete. In such cases the compiler makes a few modifications to the code for generating its Abstract Syntax Tree. For example, in the snippet shown in figure 5, illegal characters such as ‘. . .’ are found. The compiler removes such illegal characters. The teacher can access this important information comprising of compiler messages and modified code, by clicking “Toggle compiler messages”, found at the bottom of the snippet (see Figure 5). Although such code snippets are syntactically

incorrect, they might still contain useful information related to usage of the API. If the errors are minor, the teacher could continue to use such snippets as examples.

The screenshot shows the jTutors website interface. At the top, there is a search bar and navigation links for 'Create Tutor', 'Teacher Account', 'Student Account', and 'Find Tutors'. Below this is a navigation bar for snippets, with 'Snippet# 10' selected. The main content area is divided into four steps:

- STEP 1:** 'What would you like to do with the following Snippet?' with radio buttons for 'Quiz', 'Example', 'Discard', and 'Edit'. 'Quiz' is selected.
- STEP 2:** 'Select the words to blank:' with checkboxes for 'forName : Line:7', 'getConnection : Line:8', 'createStatement : Line:9', and 'printStackTrace : Line:16'. 'getConnection : Line:8' is checked. A link 'Looking for a different word? Click to add more' is present.
- STEP 3:** 'Rate the difficulty level of this snippet:' with five radio buttons. The second button is selected.
- STEP 4:** 'Hints Corresponding to: getConnection' with a text input field containing 'Attempts to establish a connecti'.

Navigation buttons '<< Previous Snippet' and 'Next Snippet >>' are located between Step 1 and Step 2. A code editor displays the following Java code:

```

1 public class ConnectSQLite {
2 public static void main(String[] args) {
3 Connection connection = null;
4 ResultSet resultSet = null;
5 Statement statement = null;
6 try {
7 class.forName("org.sqlite.JDBC");
8 connection = DriverManager.getConnection("jdbc:sqlite:C:\\SQLite\\EMPLOYEE.db");
9 statement = connection.createStatement();
10 resultSet = statement.executeQuery("SELECT EMPNAME FROM EMPLOYEEDETAILS");
11 while (resultSet.next()) {
12 System.out.println("EMPLOYEE NAME:" + resultSet.getString("EMPNAME"));
13 }
14 }
15 catch (Exception e) {
16 e.printStackTrace();
17 }
18 finally {
19 try {
20 resultSet.close();
21 statement.close();
22 connection.close();
23 }
24 catch (Exception e) {
25 e.printStackTrace();
26 }
27 }
28 }
29 }

```

At the bottom, there is a note: 'Any doubts about this snippet? Problems in adding new words even though the word is visible in the snippet? Check out what Java compiler has to say about this code by clicking here: [Toggle compiler messages](#)'.

Figure 4. Creating a fill-in-the-blank

```

1 ...
2 String query;
3 Statement stmt=null;
4 try{
5 query="create table EMP " +(EMPNO int, " +"ENAME varchar(50))";
6 stmt = conn.createStatement();
7 stmt.executeUpdate(query);
8 }
9 finally{
10 stmt.close();
11 }

```

Any doubts about this snippet? Problems in adding new words even though the word is visible in the snippet? Check out what Java compiler has to say about this code by clicking here: [Toggle compiler messages](#)

At char: 0::Syntax error on token "...", delete this token  
 At char: 4::Syntax error on token "String", package expected  
 At char: 7::Syntax error on tokens, delete these tokens  
 At char: 108::Syntax error on tokens, delete these tokens  
 At char: 131::Syntax error on tokens, delete these tokens  
 At char: 176::Syntax error on tokens, delete these tokens

Compiler modifies the snippet as follows:

```

{
String query;
Statement stmt=null;
try {
query="create table EMP " +(EMPNO int, " +"ENAME varchar(50))";
stmt=conn.createStatement();
stmt.executeUpdate(query);
}
finally {
stmt.close();
}
}

```

Figure 5. Compiler messages and modified code

### 3.4 Teacher edits the snippet

jTutors presents a way to customize the snippet. Teacher can select the “edit” option in Step 1 (see figure 4) and proceed to the editing screen. The “Edit snippet” (Figure 6) page is presented to the teacher if she decides to customize the snippet. This page presents the teacher with the original snippet in a text area. The column on the right hand presents the compiler messages and code as interpreted by the Java parser. The teacher can edit the code and compile it on the same page by pressing the “Compile” button to verify whether the snippet satisfies her requirements. She can return to the main process of tutorial selection by either selecting to confirm or cancel these recent changes.

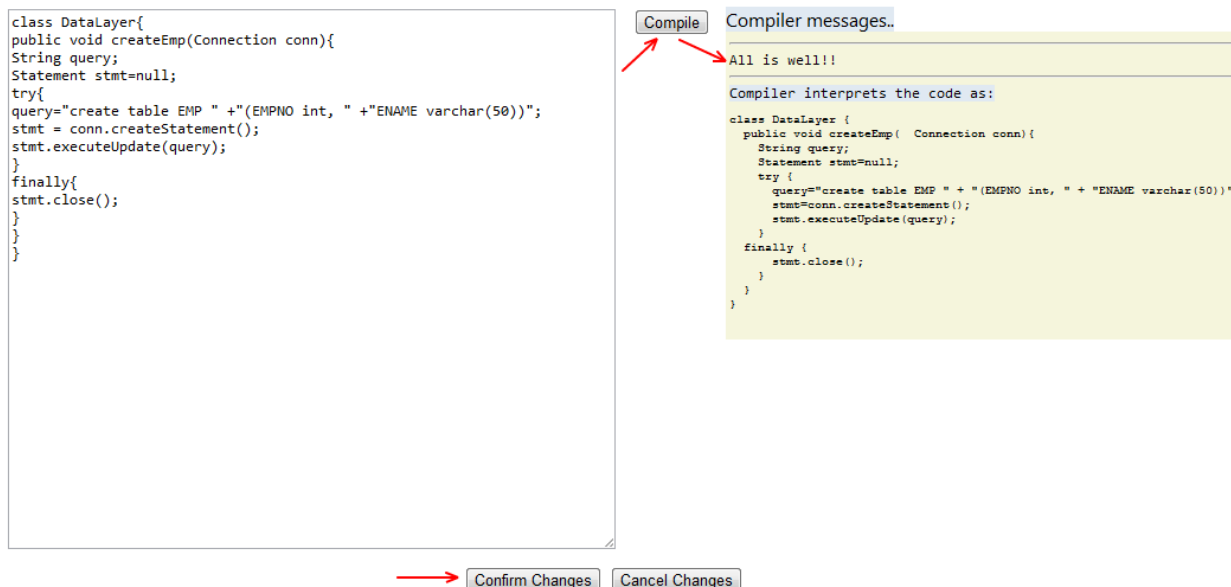


Figure 6. Edit the snippet

### 3.5 Teacher creates quizzes

While creating a quiz, the teacher might use the list of recommended words (see “Step 2” in Figure 4) to blank out words from the code snippet. The teacher could annotate each of the chosen blank word with useful hints. These hints would be shown to the student on request. In the current example, it can be seen that the teacher has chosen to blank out the method named “getConnection” (see “Step 2” in Figure 4). On clicking the check box corresponding to “getConnection”, two text boxes are displayed in the “Step 4” box. These can be used to enter a maximum of two hints explaining something related to the method “getConnection”. In the example, the teacher has entered only one hint. Annotating the words with hints is optional but highly recommended.

If the teacher is not satisfied with the choice of words presented by jTutors, she could introduce new words in the quiz (see Figure 7). Because the quiz should be about learning API,



we verify whether the words requested by the teacher are valid. For example, jTutors won't allow to block variable names or any Java language constructs like *if*, *else*, *for*, etc. since these are not related to the API.

**STEP 2**

Select the words to blank:

createStatement : Line:1

executeUpdate : Line:3

Looking for a different word? [Click to add more](#)

Word 1, Word 2, ...(comma separated)

Note: You can have only 1 word per line in the quiz. If you want to blank a word that exists on the line that contains a suggested word, then the suggested word will be lost.

Figure 7. Add new word

### 3.6 Teacher describes the tutorial

Once the teacher is satisfied by all the snippets and related quiz selections, she could save the tutorial along with a note describing the topic (figure 8). This description will be displayed to a student who plans to use the tutorial from the jTutors website.

**Please Give Your Tutor a Description**

Tutorial Name:

Tutorial Description:

Figure 8. Teacher describes the tutorials

### 3.7 jTutors packages the intelligent tutorial

All the snippets marked as “Example” or “Quiz” by the teacher are saved by the system to a common place on the server. This packaging is done in form of Java web start applications using CTAT’s technology of example-tracing tutor.

### 3.8 Student accesses the intelligent tutorial

A logged in student is directed to a webpage that lists all the available topics to be learned (Figure 9). Tutorials created by any teacher are displayed to the student. In the future this may be changed to map each student to their respective teacher if required. Once the student selects the topic, a page describing the topic is displayed (Figure 9). The student reads the description and begins with the tutorial that is an adaptive combination of code examples and fill-in-the-blank type quizzes. A quiz created in Figures 5 and 6 is presented in figure 9.


### 3.9 Design decisions on the basis of cognitive dimensions

We applied our knowledge of certain design principles while designing the key screen that teachers use for creating tutorials (Figure 4). Applicable cognitive dimensions [9] played an important role in some of the design decisions.

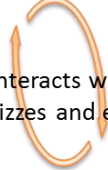
- The teacher is required to make a few choices with simultaneous access to the code snippet (see Figure 4 : Boxes representing Steps 1 to 4). The *visibility and juxtaposability* of these required elements is ensured by their strategic arrangement. If the teacher decides to discard the snippet or use it as an example by choosing respective option in Step 1, then the Steps 2, 3, 4 (figure 4) become obsolete. In such a scenario, jTutors ensures correct input by hiding the Steps 2, 3 and 4.

- Because the task of selecting tutorials requires scrolling through multiple screens, it poses a danger of *premature commitment*. We prevent the teacher from prematurely committing to her choices in each page by saving each page and allowing her to return to a previous page and alter it at any time before the final submission .
- jTutors suggests a set of words that can be blanked in a quiz. Limiting the teacher to select only amongst these words may be restrictive and increase the magnitude of *viscosity (resistance to local change)*. Therefore jTutors allows the teacher to blank words that are not suggested, but do exist in the snippet. This can be seen in Step 2 of figure 4 or its magnified version in figure 7.
- Because the teacher has to make a series of choices it is helpful if the system provides some *progressive evaluation*. For each snippet, when valid choices are selected in Steps 1 to 4, the system provides feedback displaying a check mark on that particular snippet located on the progress bar at the top. Also, each of the selections are validated immediately using Javascript. For example, if the teacher chooses to create a quiz, but forgets to choose a word to be blanked, jTutors will immediately notify her that the selection is incomplete using an alert box.

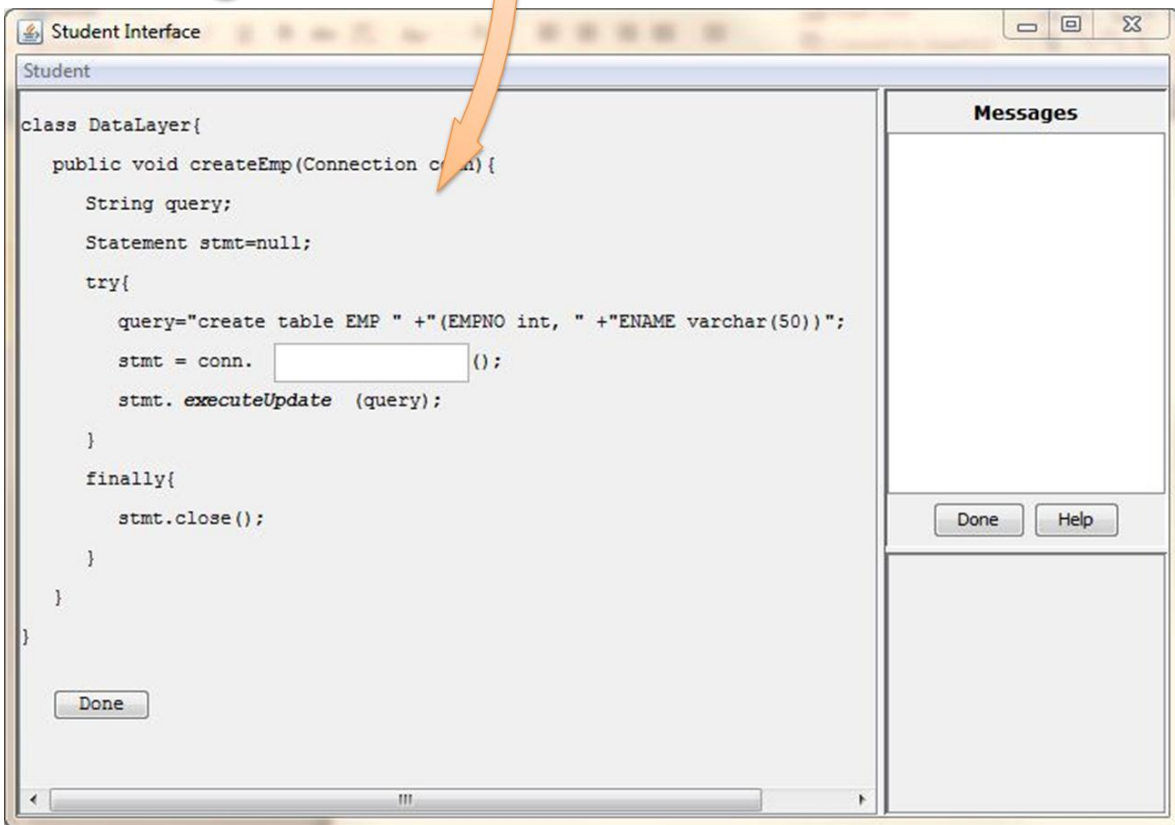
Student chooses a topic from the list



Student interacts with the adaptive quizzes and examples



iTutors Java Intelligent Tutors		
HashMap	Created by: Aniket Date:2011-01-14 Downloads: 2	Description: Map is an object that stores k...
ZipOutputStream	Created by: Aniket Date:2011-01-14 Downloads: 4	Description: This programming task is aimed...
Read XML using JAVA	Created by: Aniket Date:2011-01-21 Downloads: 5	Description: This programming task is aimed...
HashMap methods in JAVA	Created by: Aniket Date:2011-01-21 Downloads: 5	Description: This programming task is aimed...
JDBC	Created by: Aniket Date:2011-01-21 Downloads: 86	Description: JDBC is a SQL-level API -- one...
FileWrite using JAVA	Created by: Aniket Date:2011-01-21 Downloads: 0	Description: We use the classes FileWriter ...
StringBuff using Java	Created by: Aniket Date:2011-01-21 Downloads: 0	Description: You will learn how to work wit...



```
class DataLayer{
    public void createEmp(Connection conn){
        String query;
        Statement stmt=null;
        try{
            query="create table EMP " +(EMPNO int, " +"ENAME varchar(50))";
            stmt = conn.  ();
            stmt. executeUpdate (query);
        }
        finally{
            stmt.close();
        }
    }
}
```

Done

Messages

Done Help

Figure 9. Students' interaction and Java Web Start Application

## 4 Implementation Details

One of the important goals of the system is to make the system available to students and teachers through a widely accessible platform. Therefore, our system is implemented as a web application. Teachers could access the system from any computer equipped with Internet access and a latest browser that supports JavaScript. Students on the other hand, need a computer equipped with Internet access, browser, Java web start and CTAT installed. Java Web Start is a standard component of the Java Runtime Environment (JRE). CTAT is free software available at <http://ctat.pact.cs.cmu.edu>. We chose CTAT because it has been a successful platform for delivering example-tracing tutorials in the past [1]. The interactions of students with the tutorial are logged by CTAT.

Additionally, students may download the tutorials and use them any time in the future without a need to access the website from where it was downloaded. This would be an important feature where students do not have access to Internet. In such cases the teacher might provide the tutorials she creates through any other medium like CD or flash drive. Such offline use of the tutorials would limit certain features of jTutors but would still enable the students to go through the material provided by the teacher.

### 4.1 Technology Used

The system was implemented as a dynamic web project using Java EE. Eclipse Java EE IDE was used to write the code. Apache Tomcat 7.0 (<http://tomcat.apache.org/>) was used as server software to deploy the web application. SQLite (<http://www.sqlite.org/>) was used as the database engine.

The programming style closely matched the Model-View-Controller (MVC Model 2) design pattern (see figure 10). The *view* consisted of Java Server Pages (JSP). The *controller* comprised of the servlets that communicated with the view and helper classes that communicated with the model. The *model* consisted of classes that contained the business logic, for example `Tutorial1.java` and factory classes that instantiated these data models, e.g. `TutorialStore.java`. The factory classes are responsible for performing the input/output operations required to persist the data containers. In the future, even if system dependencies like the database engine or file system are changed, the logic would remain unchanged. Following good programming practices ensures that the business logic is well separated from the user interface. This made testing of the business logic a lot easier by using Junit (<http://www.junit.org/>) rather than testing the user interface which had to be tested manually.

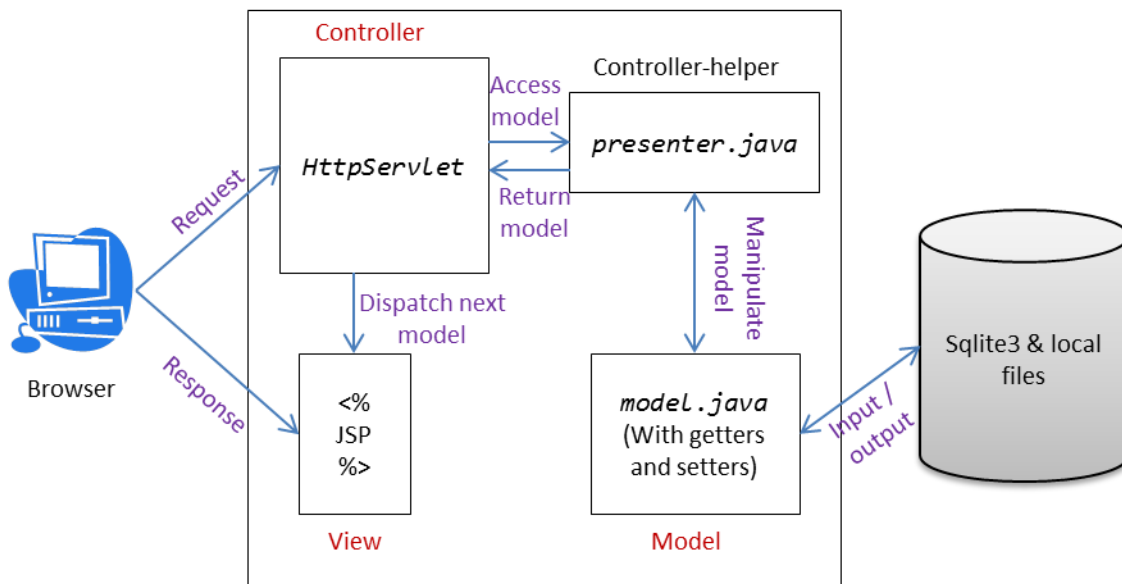


Figure 10. MVC architecture for retrieving and displaying each tutorial (model)

## 4.2 Strategies Used

The following subsections explain how the important modules of the jTutors system were implemented.

#### 4.2.1 Sanitizing the input

The API related query submitted by the teacher was passed on to Internet search engines, Yahoo Boss (<http://developer.yahoo.com/search/boss/>) and Bing Search API (<http://www.bing.com/developers>). Code samples were to be extracted from web pages represented by the URLs returned by these search engines. As webpages contain various types of textual and multimedia elements, it was necessary to decide on a strategy to extract code samples that can be utilized for generating the tutorials. In spite of the existence of searchable code repositories like Google code search and Jexamples among others, searching the web using general purpose search engines proved more effective because it represents how a typical programmer would use the API. This choice is evaluated in section 5.

Therefore, jTutors was equipped with a component for scraping code samples from HTML pages. This scraper selected the code snippets that are enclosed in `<pre>` or `<code>` tags in the HTML pages. A manual inspection of the top 10 results returned by Bing search showed that majority of the code snippets are included in either `<pre>` or `<code>` tags. Figure 11 shows the distribution of code snippets along various HTML tags when two search queries were executed in Bing search. The examples used for evaluation were *“connect to database in java example”* and *“how to use hash map java examples”*. As it can be seen, 86% of the code snippets were found in either `<code>` or `<pre>` tags. This observation helped us to simplify our process of scraping code samples because we ignored the rest of the tags.

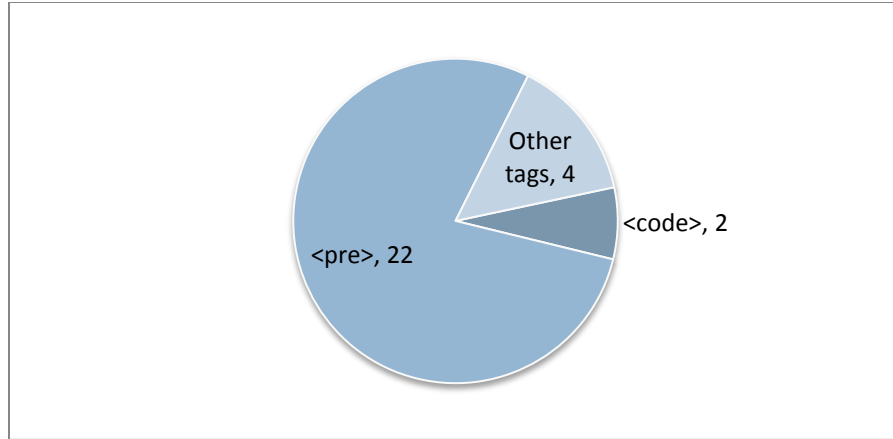


Figure 11. Code snippets mapped to HTML tags

The code snippets returned by the scraper are parsed by passing them to `CompilationUnitStore.java`. This is a factory class to generate `CompilationUnitFacade` which is a wrapper class for accessing various syntax nodes from the code snippets and other necessary information (see figure 12).

This factory class uses the Abstract Syntax Tree (AST) API provided by Eclipse JDT [4]. Eclipse uses the same library classes to create AST for the code that developers write into Eclipse's text editor. This enables to discard non-Java code snippets and java snippets that are malformed or syntactically incorrect or contain errors that are not recoverable. The above steps ensure that the algorithm that follows is supplied with valid Java code. Here, valid does not mean that the snippet should be a complete `CompilationUnit` [33]. It can also be a `Block` [33] of statements.



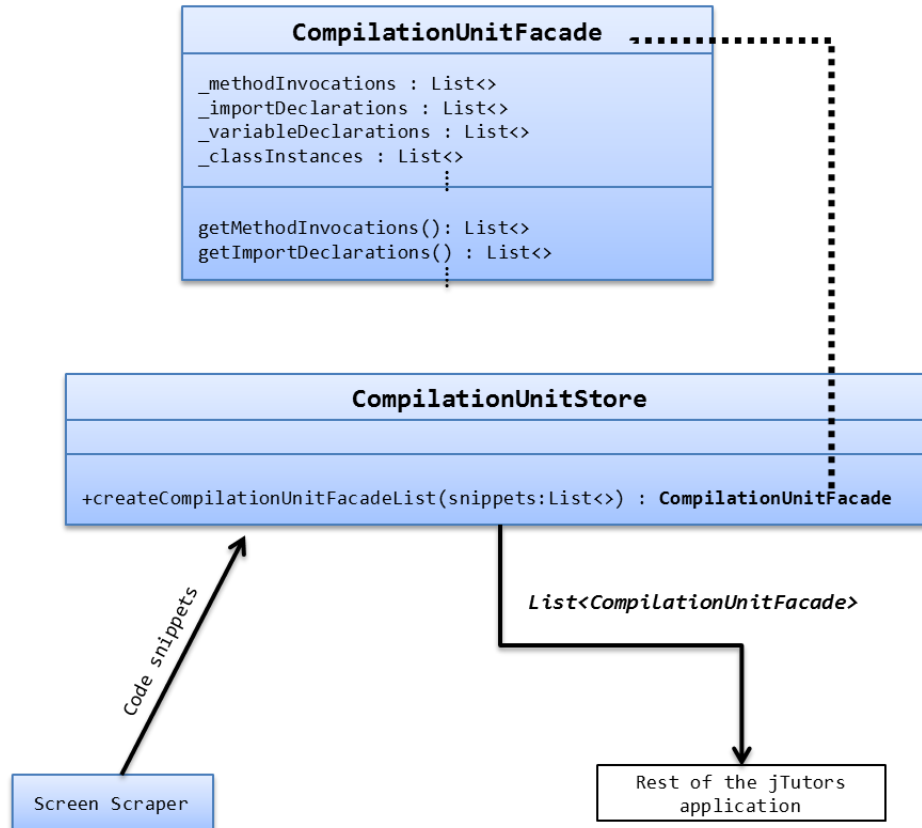


Figure 12. CompilationUnitStore

#### 4.2.2 Selecting tutorials

Each `CompilationUnitFacade` is a wrapper for the corresponding code snippets. We identified four important abstract syntax tree nodes (`ASTNode`) that would be helpful for representing the behavior of the classes, instances and methods in an API. These are as follows:

a) **ImportDeclaration**: It is used to allow a static member or a named type to be referred to by a simple name that consists of a single identifier [33].

E.g. `import java.util.HashMap;`

b) **MethodInvocation**: It is used to invoke methods on classes or instances [33].

E.g. `Collections.sort(list);`

c) **ClassInstanceCreation**: It is used to explicitly create new instances of classes [33].

E.g. `File f = new File("/home/sample.txt");`

d) **VariableDeclaration**: It is used to represent declaration of variables [33].

E.g. `File f = new File();`

The `CompilationUnitFacade` can be upgraded later if any other types of `ASTNode` are deemed necessary.

Each of the newly found `CompilationUnitFacade` was added to the local repository. *Tf-idf* weight (term frequency–inverse document frequency) was calculated for each term that belonged to one of these four types of nodes. *Tf-idf* is a popular scheme for term-weighting [13]. It uses standard term frequency, but weighted by global inverse document frequency. Thus it gives low weight to very frequent terms and high weight for infrequently occurring discriminating terms. For example, the code snippet in figure 13 consists of 3 `MethodInvocation`, viz. `addAll()`, `add()` and `get()`. *Tf-idf* weight was calculated for these 3 terms. Whereas, terms like `public`, `void`, `testThis`, `0`, etc. does not belong to any of these four types of node and hence were ignored for *tf-idf* calculation. Each code snippet was represented as a combination of four vectors, each vector representing *tf-idf* weights of the terms that belonged to the corresponding type of node.

Consider the example in figure 13. This code snippet will be considered as a combination of vectors (weights are only for illustration purpose) as shown in figure 14.

```

import java.util.ArrayList;

public class test {
    public void testThis(ArrayList<Integer> intArr) {
        ArrayList<Integer> arr = new ArrayList<Integer>();
        arr.addAll(intArr);
        arr.add(arr.get(0));
    }
}

```

Figure 13. Code Sample

```

ImportDeclaration: [java.util.ArrayList = 0.35]
MethodInvocation: [addAll = 0.43, add = 0.33, get = 0.13]
ClassInstanceCreation: [ArrayList<Integer> = 0.12]
VariableDeclaration: [ArrayList<Integer> = 0.34]

```

Figure 14. Representation of code as vectors

The calculation of the *tf-idf* weights were done as follows:

$$tfidf_{w,T,d} = tf_{w,T,d} * idf_{w,D}$$

$$tf_{w,T,d} = \frac{n_{w,T,d}}{\sum_x n_{x,T,d}}$$

Where,

$n_{w,T,d}$ : Number of occurrences of the word  $w$  in ASTNode type  $T$  for the document  $d$

And the denominator is the total number of words of ASTNode type  $T$  in the document  $d$ .

$T = \{ImportDeclaration, MethodInvocation, ClassInstanceCreation, VariableDeclaration\}$

$$idf_{w,D} = \log \frac{|D|}{|\{d|w \in d\}| + MIN}$$

Where,

$|D|$  : total number of documents in the repository

$|\{d|w \in d\}|$  : total number of documents in repository that contained the word  $w$

$MIN$  : `Float.MIN_VALUE`. This is required to avoid the `DivideByZeroException` caused in case when none of the documents in the corpus contain the word  $w$ .

Once the *tf-idf* vectors for all the code snippets returned by the search are calculated, it was necessary to rank the snippets according to their similarity to each other. The top 15 similar snippets were selected. Cosine similarity measure was used to calculate the rankings. This was calculated as follows:

$$Similarity = \begin{bmatrix} s_{0,0} & \cdots & s_{n,0} \\ \vdots & \ddots & \vdots \\ s_{0,n} & \cdots & s_{n,n} \end{bmatrix}$$

$$s_{a,b} = \sum_{t \in T} \left( \frac{a \cdot b}{\|a\| \|b\|} \right) \times W_t$$

The numerator is the dot product of the document vectors  $a$  and  $b$ , and the denominator is the product of magnitudes of the vector.

$W_t$ : Weight decided for each `ASTNode` type  $t$ .

Thus similarity between two snippets was the sum of weighted similarity of vectors of each `ASTNode` type for the two snippets. `MethodInvocation` ( $W = 5$ ) and `ClassInstanceCreation` ( $W = 3$ ) were given a higher weight than `VariableDeclaration` and `ImportDeclaration` nodes ( $W = 1$ ).

Because dot product would work only on equal-length vectors, it was necessary to ensure that all the code vectors were represented by an equal number of terms. For achieving this, a dictionary of terms was maintained for each ASTNode type. Every time a new element was found, it was added to the dictionary. Thus each vector contains all the terms even if the snippet that it represents did not contain the term. The weight for such a non-existent term for that vector would be 0. Once the similarity matrix was calculated, the top 15 indices that had maximum similarity were chosen. Diagonal elements were skipped as each element is completely similar to itself.

#### 4.2.3 Suggesting words to be blanked for quizzes

Each of the 15 selected snippets contains weighted *tf-idf* vectors for each of the ASTNode type. The ASTNode type `VariableDeclaration` was not considered as a quiz-able term since instantiating an object or generating it via method invocations is more interesting than only declaring it. Hence a maximum of **one** `ImportDeclaration`, **five** `MethodInvocation` and **three** `ClassInstanceTerm` were selected from the corresponding vectors, according to the descending order of their *tf-idf* weight. So more frequently mentioned words were ranked higher as possibilities for blanking out in quizzes. These numbers were selected only to make the interface useful and easy to use. In the future if teachers found these suggestions insufficient, the configuration can be altered.

These words are mere suggestions for the quiz. The teacher may select or ignore one or more of these suggestions while creating the actual tutorial. Additionally, the teacher may add custom words for creating the quiz.

#### 4.2.4 Adaptive testing

At the beginning of every topic, the student could read the description of the topic as saved by the teacher. To begin with, an example is provided to the student. All the terms in the example that are deemed as relevant by the algorithms mentioned in the previous sections are highlighted.

In order to test whether the student understands the topic after reading the teacher's description and this initial example, he is presented with a quiz. The difficulty level of the first quiz is the lowest level of difficulty available. During the quiz, the student's interaction with the interface is logged. The total score (maximum 100) decreases if the student enters an incorrect answer. If the student enters an incorrect answer in any text box, that text box is highlighted in red color. Whereas if the student enters a correct answer then the text box turns green. If a student does not know an answer, he may seek help from the system. jTutors provides multiple hints saved by the teacher for each of these text boxes. In case the teacher did not provide hints for a blank, the hint would be the actual correct answer. Because the student read the correct answer, filling in the blank becomes trivial; in this case, ninety percent of the score allotted for the particular blank is reduced from the total score for the current quiz.

If the student does not score more than 50% of the total points for the current quiz, then an example is presented next. This gives the student a chance to learn something that might have been missed or forgotten from the previous example or the topic description. It should be noted that the student may access the topic description at any point of time during the tutorial. The next quiz presented would then be of minimum difficulty level again.

If, on the other hand, the student scores at least 50% or more, but less than 90% of the points, a quiz of equivalent difficulty level (or higher if none is available at the same level) is presented, based on the teacher's rating of quiz difficulty. These difficulty levels are set by the teacher initially while creating the tutorials. If the score was more than 90% then the next quiz would be of higher difficulty level. Thus the student is supposed to complete the topic by answering quizzes at each difficulty level.

In future work, the sequence of these quizzes and examples might undergo changes according to the feedback from user evaluations.

## 5 Evaluation

The first part of this section focuses on summative evaluation of the search system that mines code snippets from the web, with the help of realistic test cases. Based on a heuristic assessment the second section assess how jTutors might be used to overcome barriers faced by programmers. We have used the findings and suggestions by Ko et al. [15] as the heuristics for the assessment. We understand that heuristic evaluation can be influenced by limitation or bias of our knowledge. Hence the section 5.2 should be viewed as a qualitative assessment rather than proof of our claims.

### 5.1 Snippet search results for typical queries

In order to evaluate whether the system can produce meaningful outputs for a range of typical queries about programming tasks, we tested it on 16 test queries (Figure 15 as presented on page 201 in [30]). These 16 test cases were selected because they were previously reported as being typical of queries posed by users of the MICA search engine (a system that retrieves code snippets showing how to invoke Java APIs)[30].

Specific	General
toUpperCase	concatenating strings
thread	weak references
WeakHashMap	lazy loading and caching
urlencoder	awt events
DeferredOutputStream	regular expressions
JTable	date arithmetic
JSpinner	load dll jar
class.forName	iterate array

Figure 15. Sample of queries that programmers have posted to Mica



For each query, we used jTutors to retrieve 15 code snippets. We then manually assessed each of these snippets in the results using four criteria mentioned below. The following two sections explain the definition of these criteria and the corresponding results.

### **5.1.1 Criteria**

#### **a) Is the example actually Java code?**

The `<pre>` and `<code>` tags that we used for scraping the code snippets from the web are used for various purposes, such as displaying sample output of a program, displaying formatted XML and SQL queries or other programming language snippets. The purpose of jTutors is to search for Java language snippets only.

#### **b) Does the example actually show how to use the relevant APIs?**

We inspected each query to understand its individual meaning. We then inspected the results returned by jTutors to verify whether or not these code snippets were related to the query. While deciding whether a result was related to the query, we checked for the presence of relevant variable declarations, relevant instance creations, relevant method invocations and relevant import statements. We decided that a code snippet was relevant if it passed at least 2 of these tests.

#### **c) Is it possible to turn the example into a quiz with our system?**

At times a code snippet returned was too long or too short to convert into a quiz. For this evaluation, we considered snippets longer than 40 lines of code and less than 3 lines of code as not fitting for a quiz. Sometimes jTutors presents snippets that might be syntactically

incorrect because we use a fault-tolerant parser provided by Eclipse JDT. This behavior is beneficial to utilize the small snippets that do not constitute an entire program but still illustrate the terms relevant to the API in question. In addition, the users have an option to customize the snippet if required. However, for this evaluation we have considered snippets that are syntactically incorrect as not fitting for a quiz.

**d) Is it possible to create an intelligent tutorial with our system?**

A basic intelligent tutorial in jTutors could be considered as complete if it consists of at least 2 examples and 1 quiz. Thus, if the student fails to answer a quiz after reading one example, he would be presented with at least one more example.

### **5.1.2 Results**

**a) Is the example actually Java code?**

The first column in Table 1 represents the number of Java code snippets returned by jTutors for each of the 16 queries. All the queries except “DeferredOutputStream” returned Java language code snippets. “DeferredOutputStream” did not return any results at all. Because we inspected 15 snippets for each query, the success rate for this criterion was 94%. But, out of the 15 \* 15 snippets actually returned, we found that 100% were indeed Java code.

General /Specific	Query	Number of Java codes	Number of Relevant snippets	Number of quiz-able snippets	Is sufficient for basic tutorial?
S	toUpperCase	15	13	12	<input checked="" type="checkbox"/>
S	thread	15	15	11	<input checked="" type="checkbox"/>
S	WeakHashMap	15	7	7	<input checked="" type="checkbox"/>
S	urlencode	15	8	7	<input checked="" type="checkbox"/>
S	DeferredOutputStream	0	0	0	<input checked="" type="checkbox"/>
S	JTable	15	11	8	<input checked="" type="checkbox"/>
S	JSpinner	15	15	14	<input checked="" type="checkbox"/>
S	class.forName	15	12	9	<input checked="" type="checkbox"/>
G	concatenating strings	15	5	5	<input checked="" type="checkbox"/>
G	Weak references	15	8	7	<input checked="" type="checkbox"/>
G	lazy loading and caching	15	0	0	<input checked="" type="checkbox"/>
G	awt events	15	12	7	<input checked="" type="checkbox"/>
G	regular expressions	15	14	11	<input checked="" type="checkbox"/>
G	date arithmetic	15	4	3	<input checked="" type="checkbox"/>
G	Load dll jar	15	3	3	<input checked="" type="checkbox"/>
G	Iterate array	15	11	10	<input checked="" type="checkbox"/>
	<b>Overall</b>	<b>94%</b>	<b>58%</b>	<b>48%</b>	<b>88%</b>

Table 1. Observations regarding snippets returned by jTutors

### b) Does the example actually show how to use the relevant APIs?

Each cell in Table 2 represents the number of code snippets returned by jTutors that contained the respective type of terms. The second column in Table 1 represents the number of relevant code snippets returned by jTutors. It was observed that 58% of the snippets were related to relevant APIs. The authors of MICA have differentiated the queries as *specific* and *general*. Excluding DeferredOutputSteam, 77% of the results returned by the specific queries were relevant to the topic, whereas 48% of the results returned by the general queries were relevant to the topic.

General /Specific	Query	AST Node Types			
		Import	Variable Declaration	Methods	Instance creation
S	toUpperCase	0	13	13	0
S	thread	0	15	15	14
S	WeakHashMap	5	7	7	7
S	urlencoder	6	6	7	0
S	DeferredOutputStream	0	0	0	0
S	JTable	9	11	13	11
S	JSpinner	12	13	15	15
S	class.forName	9	12	12	0
G	concatenating strings	0	5	5	0
G	Weak references	4	8	8	8
G	lazy loading and caching	0	0	0	0
G	awt events	13	12	12	12
G	regular expressions	7	14	14	0
G	date arithmetic	3	4	4	4
G	Load dll jar	0	3	3	3
G	Iterate array	7	11	11	11

Table 2. Number of snippets relevant to the API

**c) Is it possible to turn the example into a quiz with our system?**

The third column in table 1 represents the number of snippets returned by jTutors that qualified this criterion of a quiz. jTutors generated 114 quizzes from the given 16 queries. That is, 51% of the total snippets were indeed usable to create quizzes with jTutors. These results indicate that jTutors is able to produce meaningful outputs for a range of typical queries about programming tasks.

**d) Is it possible to create an intelligent tutorial with our system?**

jTutors was able to generate enough quizzes for 14 of them (88%) to create intelligent tutor.

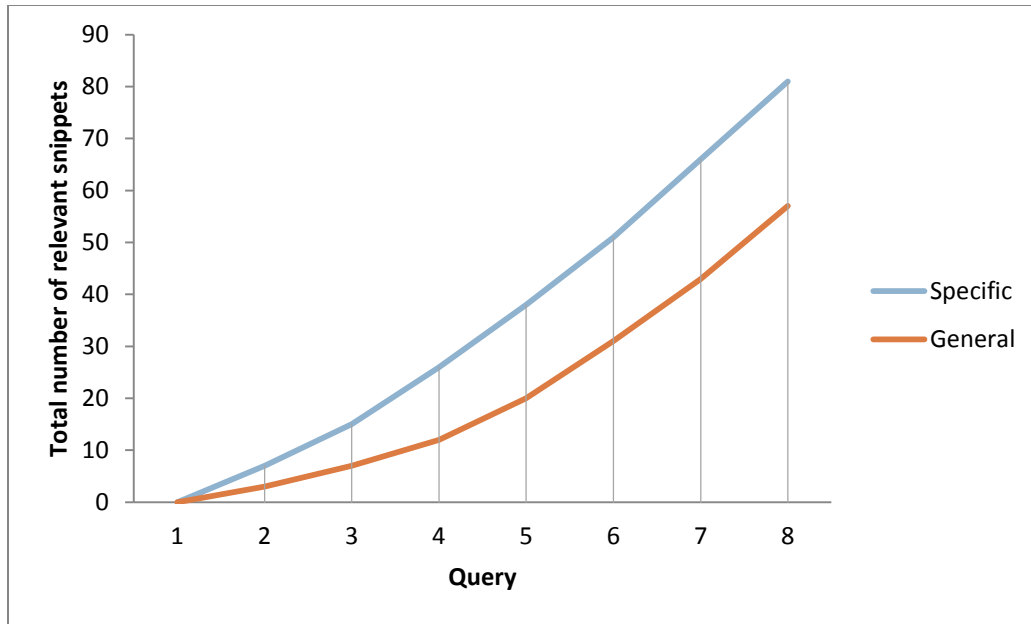


Figure 16. Snippets returned by specific versus general queries

The type of queries expected from novice Java programmers might be different than these queries and can produce different results.

## 5.2 Heuristic assessment

jTutors provides a platform for students to learn API usages that might be used to solve programming problems. We have assessed jTutors against heuristics proposed by Ko et al. [15], in order to explore the ways in which jTutors might be used to overcome learning barriers.

- i. **Design Barriers:** These are the inherent cognitive difficulties in a programming problem. It has been summarized by the authors as “I don’t know what I want the computer to do”.

**Ko’s heuristic:** *Provide solutions to a domain’s difficult problems.*

jTutors might be used to search for illustrative solutions to design problems. This might be achieved by inspecting multiple solutions that implement the particular design. For example,

a design problem in the domain of Java programming could be saving information in a database. Several snippets returned by jTutors could reveal a pattern of actions that are required to save information in the database.

It is difficult to assess jTutors for this heuristic before analyzing real-world design problems faced by student programmers. Also, the lower percentage of relevant snippets returned by jTutors for general queries (48%) as compared to specific queries (77%) hints that jTutors might not be as helpful for overcoming design barriers as compared to other learning barriers.

- ii. **Selection Barriers:** These are the problems faced by the programmer in finding available programming interfaces and identifying which of them can be used for performing a task. It has been summarized by Ko et al as “I think I know what I want the computer to do, but I don’t know what to use”.

**Ko’s heuristic:** *Offer facilities for finding programming interfaces that achieve particular behaviors.*

jTutors identifies packages, classes, objects and methods for accomplishing a task. It tries to select only those snippets from the Internet that are relatively similar in utilizing these terms for performing the task mentioned in the query. Thus, by highlighting the required APIs, jTutors might help in overcoming the selection barriers.

**Example:** One of the snippets returned by jTutors on searching ‘how to connect to a database’ is shown in figure 4. The snippet shows various programming artifacts (e.g.

Connection, DriverManager, Statement, ResultSet) required by the programmer to achieve this behavior.

- iii. **Coordination Barriers:** It represents the problems faced in combining multiple programming interfaces to achieve intended results. The authors summarize this barrier as “I think I know what things to use, but I don’t know how to make them work together”.

**Ko’s heuristic:** *Show how to coordinate programming interfaces to achieve common behaviors.*

The authors also suggest improving usability of commonly used machines. The algorithm explained in section 4.2.2 and 4.2.3 shows how jTutors finds commonly used APIs and methods that are used together and ranks them higher than others. By presenting necessary APIs that work in conjunction and highlighting the peculiar `MethodInvocation` and `ClassInstanceCreation` nodes that coordinate together, jTutors attempts to overcome the coordination barrier.

**Example:** A snippet in figure 17 shows how the `File` API needs to be used in coordination with the `DocumentBuilder` API required in achieving the behavior of reading a XML file.

**STEP 1**

What would you like to do with the following Snippet?

Quiz  Example  Discard

**STEP 2**

Select the words to blank:

- org.w3c.dom.Node : Line:6
- File : Line:11
- normalize : Line:15
- getElementsByTagName : Line:17
- item : Line:22
- getElementsByTagName : Line:25
- item : Line:26
- parse : Line:14

Looking for a different word? [Click to add more](#)

<< Previous Snippet
Next Snippet >>

To read more about this snippet visit: [^ link.](#)

```

1 import java.io.File;
2 import javax.xml.parsers.DocumentBuilder;
3 import javax.xml.parsers.DocumentBuilderFactory;
4 import org.w3c.dom.Document;
5 import org.w3c.dom.Element;
6 import org.w3c.dom.Node;
7 import org.w3c.dom.NodeList;
8 public class XMLReader {
9 public static void main(String argv[]) {
10 try {
11 File file = new File("c:\\MyXMLFile.xml");
12 DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
13 DocumentBuilder db = dbf.newDocumentBuilder();
14 Document doc = db.parse(file);
15 doc.getDocumentElement().normalize();
16 System.out.println("Root element " + doc.getDocumentElement().getNodeName());
17 NodeList nodeList = doc.getElementsByTagName("employee");

```

Figure 17. Example of reading XML file

- iv. **Use Barriers:** These barriers arise when programmers know what interfaces to use but were misled by the obscurities. It is summarized as “I think I know what to use, but I don’t know how to use it”.

**Ko’s heuristic:** *Programming interfaces should suggest for what they can be used and how to use them.*



Such barriers are common in using factory methods to create objects [6]. jTutors highlights method invocations and constructors that may return objects of a certain type. This might suggest the effect of such API constructs. Additionally, such terms could be blanked out in quizzes and annotated with hints by the teacher. Hints are aimed at explaining the use of the terms.

**Example:** Creating a `Statement` or `Connection` object using factory methods could provide a similar use barrier. The teacher could choose the snippet as shown in figure 18 (a) and then quiz the students on a snippet from figure 18 (b). The underlined lines suggests what the `Connection` and `Statement` object can be used for and how to use them. Also, while creating a quiz the teacher could annotate the factory methods `getConnection` or `createStatement` with hints that can make the use of these methods clearer.

- v. **Understanding Barriers:** These barriers are properties of a program's behavior that obscure what the program did or did not do. The authors summarize this as "I thought I knew how to use this, but it didn't do what I expected". The authors give an example of the inability of the users in using the "Timer" object since the user missed the fact that it needs to be enabled before use.

```

{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
url="jdbc:odbc:cityzoo";
Connection con2=DriverManager.getConnection(url,"Admin","sql");
Statement stmt2=con2.createStatement();
ResultSet rs=stmt2.executeQuery("Select company_id," + "company_name,
address_1, address_2, address_3," + "city, state_cd, country_cd, postal_cd,
phone_nbr "+ "from company");
}

```

(a)

```

import java.sql.*;
public class ViewingMySQL {
public static void main( String[] args){
Connection con;
Statement stmt;
ResultSet rs;
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:TOY2","root","root");
stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
rs=stmt.executeQuery("SELECT * FROM Humans");
while (rs.next()) {
System.out.println(rs.getInt("ID") + " " + rs.getString("LastName")+ " "+
rs.getString("FirstName"));
}
}
catch ( Exception e) {
System.err.println(e);
}
}
}
}

```

(b)

Figure 18. Explaining use of methods in JDBC API

**Ko's heuristic:** *Reveal what a program did or did not do.*

jTutors might be helpful in overcoming such a barrier by showing *multiple* examples with similar combinations of API usages. It might be difficult to understand behavior of a programming interface by looking at only one program. But looking at multiple examples might help to reveal the behavior of the program in different scenarios.

**Example:** An example analogous to the ambiguous behavior of the “Timer” object can be the manner in which a connection to a database engine is obtained in Java. A method named `Class.forName(<parameter>)` needs to be executed (prior to JDBC 4.0) before creating a connection to the database. This method performs the task of loading the mentioned JDBC driver but the returned object ‘Class’ is not accessed anywhere in the code that follows. Thus behavior of this method in the context of the program might not be clear. But by looking at multiple examples that use this method before actually obtaining the connection to the database might reveal the significance of this method. We cannot be confident about complying with this heuristic before any user evaluation because it depends heavily on the user’s cognition and assumptions.

- vi. **Information Barriers:** These are the barriers faced by programmers when they are unable to use the environment’s facilities to test their hypothesis. The authors summarize this as “I think I know why I didn’t do what I expected, but I don’t know how to check”.

**Ko's heuristic:** *Help inspect a program’s internal behavior.*

jTutors does not address this barrier directly. However, examples often show appropriate print statements or exception handling blocks, revealing the program's internal behavior.

**Example:** The snippet shown in Figure 19 represents how this type of selection might be made. In such a case the teacher might select only those examples that contain print statements or some type of guard (e.g. caught exceptions) after the API usage. Such statements might give information about the expected behavior of the program. In this case, the print statement suggests what is expected from the object `resultSet`. The drawback of trying to overcome this barrier may result in reduction in the number of available snippets.

```
Class.forName("org.sqlite.JDBC");
connection=DriverManager.getConnection("jdbc:sqlite:C:\\SQLite\\EMPLOYEE.db");
statement=connection.createStatement();
resultSet = statement.executeQuery("SELECT EMPNAME FROM EMPLOYEEDETAILS");
while (resultSet.next()) {
    System.out.println("EMPLOYEE NAME:" + resultSet.getString("EMPNAME"));
}
```

Figure 19. Snippet showing print statement. Part of figure 5.

## 6 Conclusion And Future Work

Our research contribution through jTutors is to provide a novel way of creating intelligent tutorials by leveraging code snippets available on the Internet. The results returned by the system contains more terms representing API usage as compared to other code repository search engine like Google code search. jTutors attempts to overcome the common barriers posed to a programmer. Although we did not create the mechanism of creating an intelligent tutoring system, we provide a new way in which such tutoring systems might be utilized in the domain of teaching Java APIs.

The jTutors website could provide a platform for sharing tutorials among the Java teaching community. A fellow researcher is already working on this aspect of jTutors. Such a website could help users to share the tutorials they create with students and other users. Currently, jTutors does not support editing the tutorials once they are created. In future, this website could allow the users to customize the tutorial and share it again. Users may be allowed to allocate ratings or comment on the tutorials. This would allow other users to read the reviews from other users before trying out the tutorial. The sheer number of online communities dedicated to discussions on Java shows the popularity of the language and willingness amongst novices to learn the language. The effectiveness of such a platform can be evaluated by a field study of user interactions with the website.

Currently we are providing jTutors only as a way to solve quizzes or read examples with some hints provided by the teacher. But it would be an interesting project to combine these quizzes with a more detailed explanation if requested by the student user. It could be achieved

by extracting the texts or images displayed on the webpage from which the code snippet was extracted. Also, the intelligent feedback system of JITS based on its Java Error Correction Algorithm (JECA) was impressive [31]. Assimilating such a system that identifies the intent of the student's submission into our jTutors might enhance the student's experience. These ideas can be built upon to create curriculum planning for teaching Java APIs. Instead of generating only examples and quizzes, the jTutors system could be tailored to generate chapters that teach a Java API. Since the system could be used to test the knowledge of the student with the help of quizzes, this information could be used to create a model of the student. This could help in recommending the next topic by judging the student's proficiency in the current one.

jTutors uses any snippets from the web as long as they pass the error-tolerant parsing algorithm implemented by the Eclipse AST API. The teachers could use their judgment to create more elaborate quizzes from these snippets. This would mean, instead of blanking out one word at a time, the teacher could blank out a text area and annotate it with comments explaining the purpose of the lines of code that should go in the blank area. The students would then be required to write a solution to the given task rather than only filling out single words. This would help the teacher to test the student's problem solving skills after the students have mastered their understanding of the API by using the jTutors system as it is in the current state. The Eclipse AST API can then be used to find the student's syntax errors and generate constructive feedback in the form of error messages and suggestions. Since this system would be highly interactive, its success would depend on feedback from user studies on teachers and students.

The work done in creating jTutors could be extended to other domains in the future. Similar to Java, other programming languages like C#, PHP, etc. also rely heavily on APIs for providing all the exciting features to the programmers. It would be interesting to adapt jTutors to different languages.

Like any other research, jTutors is not perfect. Laboratory testing with teachers and students will help to improve the usefulness and also the ease of use of the system. In the cases where the results returned are not useful, further analysis should be done to identify the reasons.

## 7 Bibliography

- [1] V. Alevan, B. M. McLaren, J. Sewall, K. R. Koedinger, "Example-tracing tutors: A new paradigm for intelligent tutoring systems," *Int. J. Artif. Intell.* Ed. 19, Amsterdam, The Netherlands, 2009, pp. 105-154.
- [2] C. Conati, "Intelligent Tutoring Systems: Improving Student Modeling," *Talk at CSE Colloquia - 2005, The University of Washington Computer Science & Engineering Colloquium*, [Online], Available:  
[http://www.digitalwell.washington.edu/rcuwtvdownload/uw\\_cse05\\_inteltu\\_ipodv.m4v](http://www.digitalwell.washington.edu/rcuwtvdownload/uw_cse05_inteltu_ipodv.m4v) , 2005.
- [3] T. Corbett and J. R. Anderson, "LISP intelligent Tutoring System: Research in Skill Acquisition," In *Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches*, J. H. Larkin & R. W. Chabay, Eds., pp. 73-109. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 1992.
- [4] Eclipse Java development tools (JDT), [Online], Available: <http://www.eclipse.org/jdt/>
- [5] D. S. Eisenberg, J. Stylos, B. A. Myers, "Apatite: A New Interface for Exploring APIs," *Proc. 28th Int. conference on Human Factors Computing Syst. (CHI '10)*, Atlanta, GA, 2010, pp. 1331-1334.
- [6] B. Ellis, J. Stylos, B. A. Myers, "The Factory Pattern in API Design: A Usability Evaluation," *Proc. 29th Int. conference on Software Eng. (ICSE '07)*, IEEE Computer Society, 2007, pp. 302-312.
- [7] Google Code Search, [Online], Available: <http://www.google.com/codesearch>



- [8] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, C. Cumby, "Exemplar: EXEcutable exaMPles ARchive," *Proc. 32nd ACM/IEEE International Conference on Software Eng. - Volume 2 (ICSE '10), Vol. 2. ACM*, May 2010, Cape Town, South Africa, 2010, pp. 259-262.
- [9] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' approach," *Journal of Visual Languages and Computing*, 1996, pp. 131-174.
- [10] T. Heift and D. Nicholson, "Web delivery of adaptive and interactive language tutoring," *Int. J. Artif. Intell. Ed.*, 12, 4, 2001, pp. 310-324.
- [11] R. Hoffmann, J. Fogarty, D. S. Weld, "Assieme: finding and leveraging implicit references in a web search interface for programmers," *Proc. 20th annual ACM symp. on User interface software and technology (UIST '07)*, Newport, Rhode Island, 2007, pp. 13-22.
- [12] R. Holmes, R. J. Walker, G. C. Murphy, "Strathcona example recommendation tool," *Proc. 10th European Software Eng. conference held jointly with 13th ACM SIGSOFT Int. Symp. Found. Software Eng. (ESEC/FSE-13)*, Lisbon, Portugal, 2005, 237-240.
- [13] P. Husbands, H. Simon, C. Ding, "Term norm distribution and its effects on Latent Semantic Indexing," *Inf. Process. Management.* 41(4), 2005, pp. 777-787.
- [14] Jexamples, [Online], Available: <http://www.jexamples.com/>
- [15] A. Ko, B. A. Myers, H. H. Aung, "Six Learning Barriers in End-User Programming Systems," *Proceedings 2004 IEEE Symp. on Visual Languages - Human Centric Computing (VLHCC '04)*. IEEE Computer Society, Washington, DC, 2004, pp. 199-206.
- [16] Koders, [Online], Available: <http://www.koders.com/>

- [17] K. R. Koedinger and J. R. Anderson, "Reifying Implicit Planning in Geometry: Guidelines for Model-Based Intelligent Tutoring System Design," *In Computers as Cognitive Tools*, S. P. Lajoie and S. J. Derry, Eds., (pp. 15-45). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 1993.
- [18] Krugle, [Online], Available: <http://www.krugle.org/>
- [19] A. Mitrovic, "An Intelligent SQL Tutor on the Web," *Int. J. Artif. Intell. Ed.* 13, 2-4, 2003, pp. 173-197.
- [20] W. R. Murray, "Intelligent Tutoring Systems for Commercial Games: The Virtual Combat Training Center Tutor and Simulation," *Proc. second Artif. Intel.l and interactive digital entertainment conference*, Marina del Rey, CA, 2006, pp. 66-71.
- [21] J. Ossher, S. Bajracharya, E. Linstead, P. Baldi, C. Lopes, "SourcererDB: An aggregated repository of statically analyzed and cross-linked open source Java projects," *Proc. 2009 6th IEEE Int. Working Conference on Mining Software Repositories (MSR '09)*, 2009, pp.183-186.
- [22] S. P. Reiss, "Semantics-based code search," *Software Eng, 2009. ICSE 2009. IEEE 31st International Conference*, 2009, pp.243-253.
- [23] E. Roberts, (2006). "ACM Java Task Force. Project Rationale," [Online] Available: <http://jtf.acm.org/rationale/>
- [24] E. Roberts and G. Engel, "Computing Curricula 2001: Final Report of the Joint ACM/IEEE-CS Task Force on Computer Science Education," Los Alamitos,CA: IEEE Computer Society Press, December 2001.

- [25] E. Roberts, "The dream of a common language: The search for simplicity and stability in computer science education," *Proc. Thirty-Fifth SIGCSE Technical Symposium on Computer Science Education, Norfolk, VA, 2004*, pp. 115-119.
- [26] M. P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers," *IEEE Softw.*, vol. 26, 6, Los Alamitos, CA, Dec. 2009, pp. 27-34.
- [27] D. Sleeman and J. S. Brown, "Introduction: Intelligent Tutoring Systems," In *Intelligent Tutoring Systems*, D. Sleeman and J. S. Brown, Eds., pp. 1-11, New York: Academic Press, 1982.
- [28] J. S. Song, S. H. Hahn, K. Y. Tak, J. H. Kim, "An intelligent tutoring system for introductory C language course", *Comput. Educ.* 28, 2, February 1997, pp. 93-102.
- [29] J. Stylos, A. Faulring, Z. Yang, B. A. Myers, "Improving API Documentation Using API Usage Information," *Proc. 2009 IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC)*, Corvallis, OR, 2009, pp.119-126.
- [30] J. Stylos, and B. Myers, "Mica: A Web-Search Tool for Finding API Components and Examples," *Visual Languages and Human-Centric Computing (VL/HCC'06)*, 2006, pp. 195-202.
- [31] E. Sykes, "Qualitative Evaluation of the Java Intelligent Tutoring System," *Journal of Systemics, Cybernetics and Informatics*, 3(5), 2005, pp. 49-60.
- [32] M. Urban-Lurain, (1996). "Intelligent tutoring systems: An historic review in the context of the development of artificial intelligence and educational psychology," [Online]. Available: <http://www.cse.msu.edu/rgroups/cse101/ITS/its.htm>

[33] J. Gosling, B. Joy, G. Steele, G. Bracha, "Java™ Language Specification," [Online].

Available: <http://java.sun.com/docs/books/jls/>