



Thank you for downloading this document from the RMIT Research Repository.

The RMIT Research Repository is an open access database showcasing the research outputs of RMIT University researchers.

RMIT Research Repository: <http://researchbank.rmit.edu.au/>

Citation:

Cheong, C 2010, 'Coding without sight: Teaching object-oriented java programming to a blind student', in Andrew Burge (ed.) Eighth Annual Hawaii International Conference on Education, Honolulu, Hawaii, 7-10 January 2010, pp. 1-12.

See this record in the RMIT Research Repository at:

<https://researchbank.rmit.edu.au/view/rmit:13231>

Version: Accepted Manuscript

Copyright Statement: © Christopher Cheong

Link to Published Version:

http://www.hiceducation.org/proceedings_edu.php

PLEASE DO NOT REMOVE THIS PAGE

CODING WITHOUT SIGHT: TEACHING OBJECT-ORIENTED JAVA PROGRAMMING TO A BLIND STUDENT

Christopher Cheong
School of Business Information Technology
RMIT University
Melbourne, Australia,
christopher.cheong@rmit.edu.au

Abstract

In this paper, I describe my experience of teaching object-oriented Java programming to a blind student. This includes the particular environment setup used (a screen reader, JAWS, and an advanced Windows-based text editor, Textpad) and alterations made to the course to accommodate the blind student's special needs. I also discuss how a number of difficulties encountered by the blind student, such as compiling Java applications using the command-line interface and javac, a Java compiler, was addressed and provide some practical recommendations based on my experience.

Keywords: Programming, teaching and learning, blind student, Java, object-oriented.

1 INTRODUCTION

In the first semester of 2008, a blind student enrolled in *Applications Development 2¹ (AD2)*, a Java programming course in the Bachelor of Business (Business Information Systems) program. Although I taught AD2 for a number of semesters and I also previously taught the same blind student in another course (which focused on operating systems and was more theoretical in nature), this was a new challenge as I have never taught programming to a blind person before. Furthermore, the design of AD2 did not cater for blind students and it includes topics, such as graphical user interface (GUI) design and implementation, which are more challenging to teach to the blind as these topics are visual in nature.

In this paper, I discuss my experience of teaching a blind student to program in Java and provide some practical recommendations based the personal experience obtained.

2 A REVIEW OF TEACHING COMPUTING TO THE BLIND

The term “visually impaired” covers an array of visual conditions (Ludi and Reichlmayr 2008, Murray and Armstrong 2004), however, the focus of this paper is on the blind (i.e., those who lack the sense of sight completely). In this section, background material is provided and includes a basic overview of assistive technology available for the blind (Section 2.1), how the blind are able to operate computers (Section 2.2), and a brief description of the course, *Applications Development 2* (Section 2.3), in which I taught the blind student object-oriented Java programming.

¹ The course has since been renamed to “Business Information Systems Development 2”.

2.1 Assistive technology for the blind

As the blind cannot see, they must use their other senses to learn and interact. In regard to computer interaction, facilitated by the use of assistive technology, the sense of touch and hearing are the most useful.

Assistive technology is generally composed of specialized hardware and software, the most useful of which are *screen readers*. A screen reader is a piece of software that synthesizes textual information displayed on the screen to speech. Synthesis of text to speech can be carried out at various levels, such as reading sentences, words, or individual characters. Blind users can navigate through displays on the screen, including graphical user interfaces (GUIs), by using keyboard shortcuts and receive feedback through the screen readers' audio output.

One popular screen reader is *JAWS (Job Access With Speech) for Windows*² (referred to as "JAWS" henceforth for brevity). As the name implies, the software is for the Windows platform, although it was originally created for the MS-DOS operating system.

In terms of hardware, it is possible to use special printers to print tactile diagrams, which the blind can "read" by feeling the imprints. *Refreshable Braille displays* are also useful. These devices can display a line of text from the screen in Braille and can dynamically update the Braille display as the user navigates between text.

2.2 Computing by the blind

There are a number of different types of research related to the blind and computing. These range from making computing accessible to the blind (such as the game, AudioQuake (Atkinson *et al.* 2006)), to educating future programmers about accessibility by incorporating it in computer science curriculum (Cohen *et al.* 2005, Gellenbeck 2005), and to research on how to assist the blind to learn programming. As the purpose of AD2 is to teach programming (to a blind student), the focus of this study is on the latter.

There have been a number of programs to encourage the blind to pursue programming. These include the design and delivery of specialized short courses for the blind involving the use of Lego Mindstorms (Ludi and Reichlmayr 2008) and instant messaging chatbots (Bigham *et al.* 2008) to generate interest in blind youths. To overcome the challenges of accessibility, screen readers were used in these short courses, and in one of the courses, computer scientists were paired with blind participants to assist with programming.

Although screen readers are very useful accessibility tools for general computing tasks, simply translating information from the visual channel to the auditory channel is sometimes not adequate for advanced and specialized computing tasks, such as programming. Some programming languages make heavy use of brackets and braces, and abbreviated naming conventions that are not completely pronounceable in human languages when read literally. For example, the Hungarian notation, which is commonly used in Visual Basic, uses an abbreviated prefix as part of variable names to denote the data type of the variables (e.g., `dblValue`, `btnOK`).

More importantly, as a programmer, it is necessary to understand the structure of programs, which visually-abled programmers understand by seeing the indentation of code. Indentation of code is achieved through whitespace and when screen readers translate text to speech at sentence level, the blind are unable to determine the amount of whitespace in between words. Instead, they must use the

² http://www.freedomscientific.com/fs_products/software_jaws.asp

screen reader to translate at character level to determine the location and amount of whitespace, a cumbersome and tedious process. Thus, other alternatives have been pursued to assist the blind to learn programming.

Audio Programming Language (APL) (Sánchez and Aguayo 2005) was developed to motivate and interest the blind in programming. It was designed by blind users for blind users and is different to conventional programming in that sound is used as the fundamental unit. APL programmers store sounds that are later manipulated. Although an innovative approach to teaching novice blind programmers, APL is only a learning tool and not an alternative to conventional programming. That is, it is not intended for professional software development.

*JavaSpeak*³ (Smith *et al.* 2000, Francioni and Smith 2002, Smith *et al.* 2004) is another tool developed to assist the blind to program, in this case, in the Java programming language. Unlike APL, it is not solely for the purposes of learning to program, it is also intended for the professional development of software. Similarly to screen readers, JavaSpeak translates text to speech. However, it provides more specialized features and functionalities than general-purpose screen readers that are specific to programming in Java. These are designed to assist blind programmers in the same way features such as code indentation and code highlighting assist visually-abled programmers, however, these features are through the auditory rather than the visual channel. JavaSpeak can produce audio output at various different levels for the following program components (Francioni and Smith 2002):

1. *Basic compilation unit*
2. *Class composition*
3. *Method composition*
4. *Nesting levels*
5. *Block composition*
6. *Block details*
7. *Token by token*
8. *Character by character*

Given the code in Listing 1, at the class composition level, the text presented in Figure 1 would be conveyed through the audio output by JavaSpeak.

```
public class Calculator {
    private int result;

    public int sum(int x, int y) {
        result = x + y;
        return result;
    }

    public int difference(int x, int y) {
        result x - y;
        return result;
    }
}
```

Listing 1. Example Java Code.

³ <http://cs.winona.edu/cscap/javaspeak.html>

```
Begin class declaration.  
Public class calculator.  
Class variable: private int result.  
Method: sum.  
Method: difference.  
End class declaration.
```

Figure 1. Rendered Speech by JavaSpeak based on Listing 1.

For more details on the different levels of audio output, the reader is referred to Francioni and Smith (2002).

One particular hindrance for blind programmers are graphical user interfaces (GUIs) for more emphasis on graphics and other visual effects. With advances in user interfaces, many operating systems have moved from command-line interfaces to GUIs. This has typically made the computer more usable and user-friendly to average users. However, these advances have had a reversed effect for the blind (Siegfried 2006). Although advances in assistive technologies have made computers more accessible to the blind, it is very challenging for a blind programmer to design and program a GUI because of the visual aspects involved.

In some programming languages, such as Java and Visual Basic, it is possible to define a GUI through code. In the case of Visual Basic, this is difficult in practice, as it requires the programmer to consider a lot of details about the GUI. It is even more difficult for blind programmers as they are unable to visually inspect the results of the code. A scripting language has been developed to assist the blind to create Visual Basic GUIs (Siegfried *et al.* 2004, 2005, Siegfried 2006, Franqueiro and Siegfried 2006). Originally, the developers of the language intended to create a scripting language to create GUIs on various programming platforms, however, feedback from the community of blind programmers⁴ indicated that there were more interest in the creation of GUIs for Visual Basic.

2.3 Applications Development 2 course

In the Bachelor of Business (Business Information Systems) (BBIS) program offered by RMIT University's School of Business Information Technology, students are taught a combination of business and information technology (IT) courses. *Applications Development 2 (AD2)* is a core second year programming course that forms part of the BBIS program's IT "stream". Although students are required to complete two pre-requisite programming courses to qualify for AD2, AD2 is the first course in which students are introduced to object-oriented programming through the Java programming language. Students are taught object-oriented design in a co-requisite course (i.e., typically taken at the same time as AD2).

The AD2 course covers various topics, such as:

- Java data types and constructs
- Java beans and exceptions
- Inheritance, polymorphism, and interfaces
- GUI design and implementation
- Database connectivity and manipulation

⁴ <http://www.nonvisualdevelopment.org>

3 TEACHING OBJECT-ORIENTED JAVA TO A BLIND STUDENT

In this section, details are provided on how the blind student prefers to interact with a computer (Section 3.1), how programming was taught to the student (Section 3.2), suitable alterations made for the blind student (Section 3.3), and a number of practical recommendations are outlined (Section 3.4).

3.1 Operating a computer without sight

The blind student operates a computer with the assistance of JAWS, which is available to him on the majority of computers in the teaching and general-purpose laboratories. The blind student is generally able to achieve most tasks a visually-abled user can by using JAWS and keyboard shortcuts. He is obviously unable to use the mouse as he cannot receive (i.e., see) any appropriate feedback from the device. In terms of feedback, the student uses the audio output from JAWS and also the beeps the computer emits. In situations where JAWS is not running (i.e., the operating system must first load before loading the screen reader application) and the computer does not beep, the student uses other noises emitted by the computer, such as listening to the hard disk or optical drive spinning. The student also memorizes certain sequences in order to progress through situations in which feedback that is discernible to him is not produced from the computer. For example, he is able to install the Windows operating system by executing memorized keystrokes (JAWS cannot run during the installation of Windows).

3.2 Programming without sight

AD2 students use the *Eclipse*⁵ *Integrated Development Environment (IDE)* to develop their Java applications. Eclipse is a popular and powerful IDE with many features which AD2 students find useful, including syntax highlighting, syntax checking, and content assist.

The features of Eclipse greatly enhance productivity for students in general. However, a number of these features, which are useful for those the visually-abled, proved to be a hindrance for the blind student. One initial issue with Eclipse is that, as with most modern IDEs, its main window is composed of a number of smaller tabbed windows (refer to Figure 2). Although the blind student can navigate between these smaller tabbed windows using keyboard shortcuts, it is time consuming and there are some operational issues with JAWS (e.g., JAWS does not read out the name of the tabbed window which is in focus).

⁵ <http://www.eclipse.org>

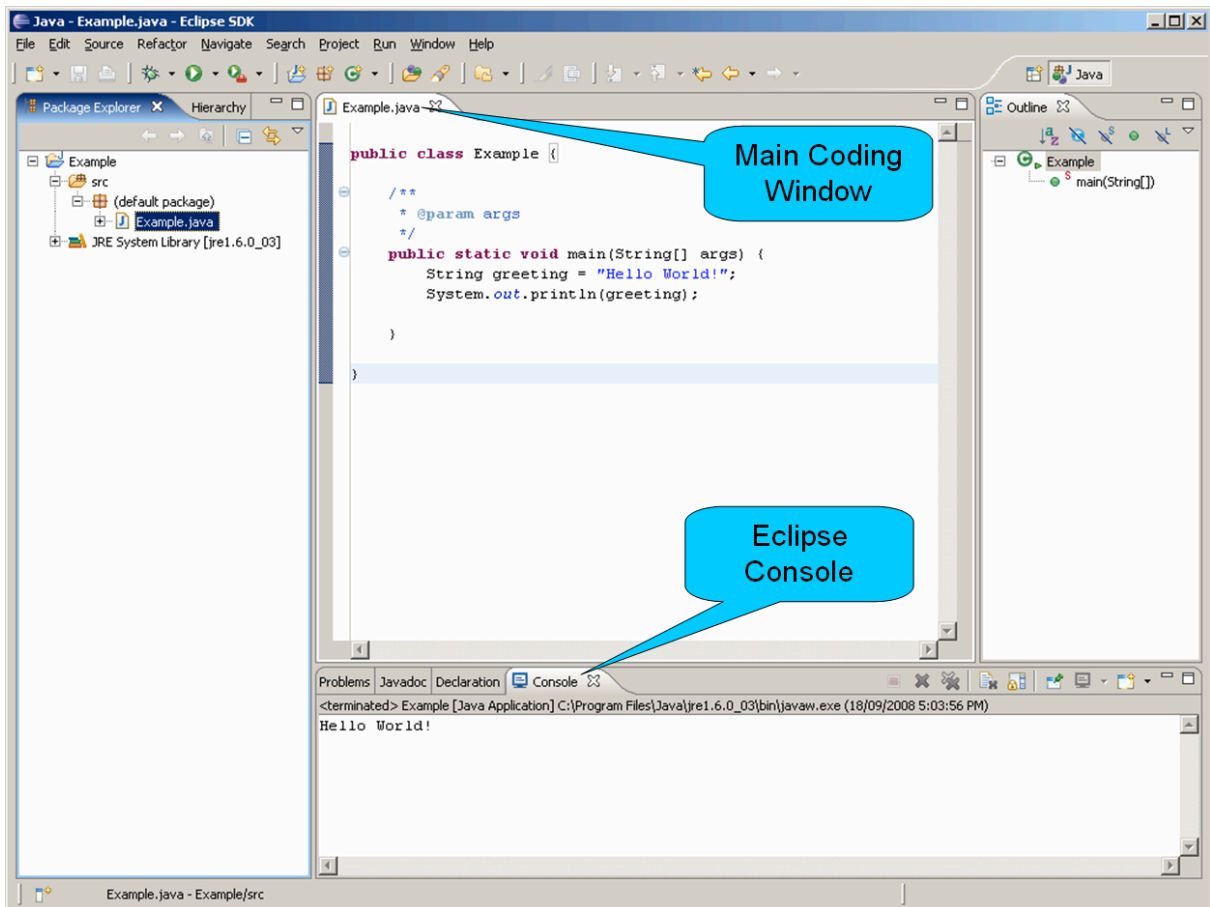


Figure 2. The Eclipse Integrated Development Environment.

Features, such as the ability to click on compilation or run-time errors which appear in the Eclipse console to quickly access the offending line of code in the main coding window (refer to Figure 2) is useful for most AD2 students. However, this is, in fact, a hindrance to the blind student as he cannot use the mouse to click on the links. Thus, the combination of Eclipse and JAWS was not a suitable option. Although JavaSpeak (Smith *et al.* 2000, Francioni and Smith 2002, Smith *et al.* 2004) would have been ideal, especially since it is available as an Eclipse plug-in, I was, unfortunately, unaware of its existence at the time. Instead, it was believed the blind student would be better served by using an IDE with a minimal number of smaller windows or at worse, a plain and simple text editor such as *Notepad*.

I investigated the suitability of *Textpad*⁶, an advanced Windows-based text editor. Textpad met the requirement of being composed of a minimal number of smaller windows as part of its main window (refer to Figure 3). Furthermore, there were not many compatibility issues between JAWS and Textpad.

⁶ <http://www.textpad.com>

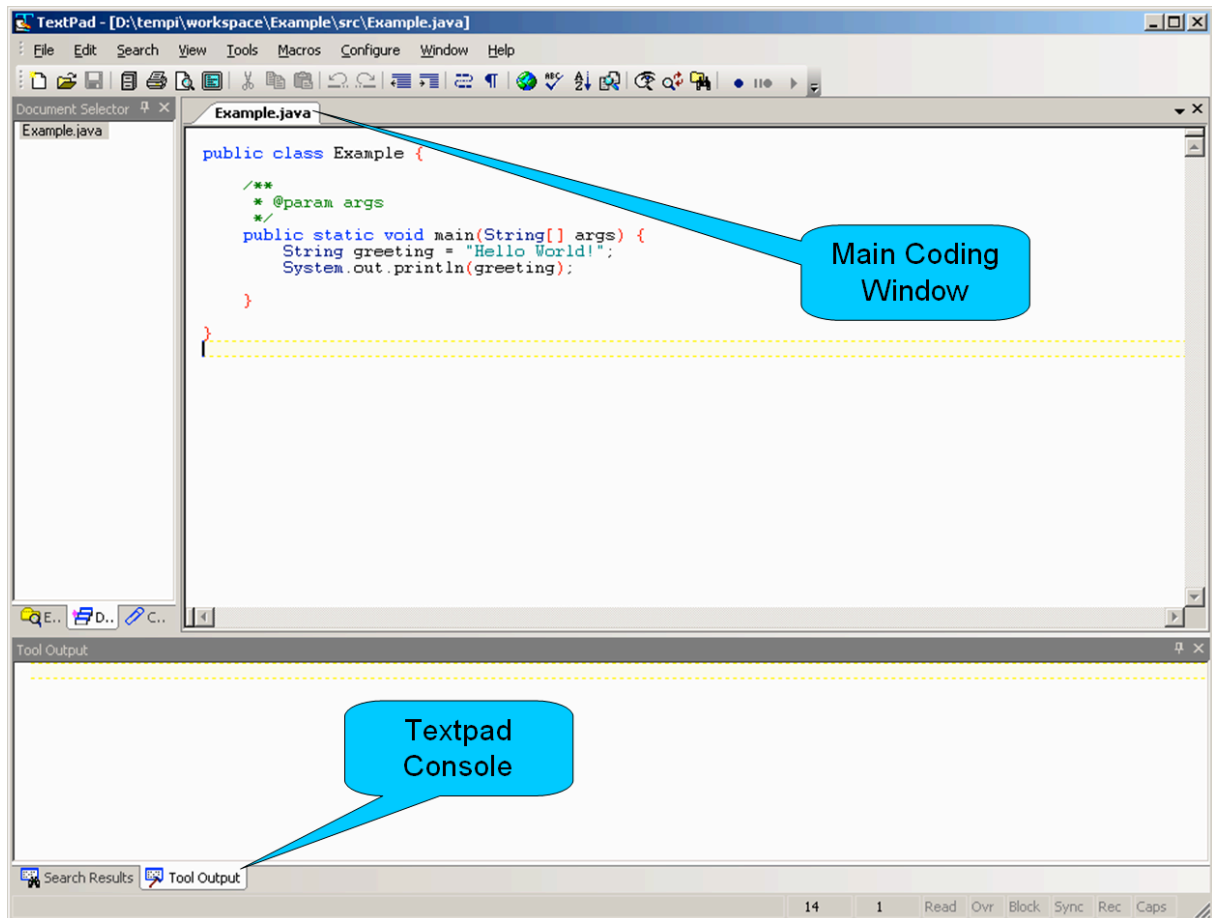
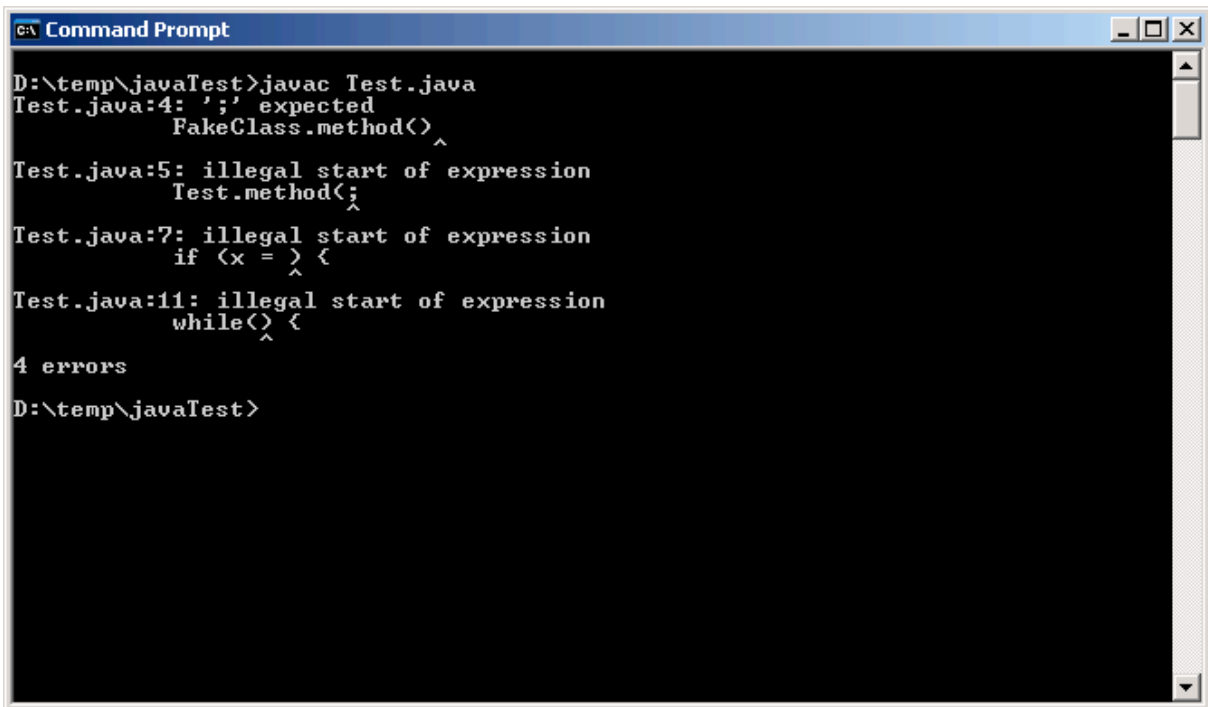


Figure 3. The Textpad Advanced Text Editor.

Textpad has a number of practical features that is advantageous to a blind user over a plain text editor such as Notepad. A single Textpad instance can load and operate on multiple text files. The user can navigate between the opened files using a keyboard shortcut. The user is also able to move to a specific line number in a text file by using a keyboard shortcut and entering the desired line number. Textpad also provides auto-indentation using a keyboard shortcut and allows the user to compile source code. Although indentation of code has no value to a blind programmer, it is important when visually-abled programmers need to review, understand, and edit the code. Thus, a blind programmer can produce indented code by using Textpad's auto-indentation feature.

Although a programmer can compile code using keyboard shortcuts from within Textpad, the blind student preferred to compile his applications using the MS-DOS command prompt. One problem with code compilation is that when compilation errors are produced, whether in Textpad or at the command prompt, the blind student is unable to determine the exact location of the error. The main problem is that although the Java compiler, `javac`, indicates the file name and line number of the error, a caret is used to point to the exact location of the error (refer to Figure 4). Furthermore, `javac` generates a list of errors, which is quite verbose. The difficulty is that JAWS reads all the error output generated sequentially, and the blind student is unable to navigate through the text on the command prompt. Navigation is important, as the blind student needs JAWS to re-read certain parts of the error message, such as offending line numbers.

The image shows a Windows Command Prompt window titled "c:\ Command Prompt". The prompt is at "D:\temp\javaTest>". The user has entered "javac Test.java". The output shows four compilation errors: "Test.java:4: ';' expected FakeClass.method() ^", "Test.java:5: illegal start of expression Test.method(; ^", "Test.java:7: illegal start of expression if (x =) { ^ {", and "Test.java:11: illegal start of expression while() { ^ {". The output ends with "4 errors" and the prompt "D:\temp\javaTest>".

```
c:\ Command Prompt
D:\temp\javaTest>javac Test.java
Test.java:4: ';' expected
    FakeClass.method() ^
Test.java:5: illegal start of expression
    Test.method(; ^
Test.java:7: illegal start of expression
    if (x = ) { ^ {
Test.java:11: illegal start of expression
    while() { ^ {
4 errors
D:\temp\javaTest>
```

Figure 4. Error Output from the Java Compiler.

To overcome this problem, a batch file was created for the blind student to use to compile his application. The batch file compiles the student's code and redirects any output from `javac` to a text file, which is then opened in a text editor. The blind student is then able to navigate through the output in the text file at his leisure.

One difficulty that was not addressed well is run-time errors. When a run-time error occurs, the output is displayed in the console. This provides the same difficulty as compile-time errors, i.e. JAWS reads all the output sequentially and the blind student is unable to navigate through it for JAWS to re-read particular parts of the error message. Since the blind student was creating text-based applications, it was difficult to find a simple and effective approach similar to that used for compile-time errors (the program output was mixed with error messages). Thus, the blind student required additional personal assistance with run-time errors.

This is an important issue that needs to be addressed in order to facilitate debugging for the blind. Surprisingly, in the available JavaSpeak material, it was unclear whether JavaSpeak addresses this issue.

3.3 Course alterations for accommodating the blind student

Initial changes to the course to accommodate the blind student involved altering the format of the course materials to make them accessible to him. This was not particularly challenging due to my previous experience with teaching him in a different course. Although JAWS is able to read most text displayed to the screen, the blind student claimed that the material was more accessible to him in rich text format (RTF) or tagged portable document format (PDF). I provided the student with all the course materials in RTF as he had a preference for this particular format.

The current lecture notes were written in Microsoft PowerPoint and the tutorial notes in Microsoft Word. Conversion to RTF was relatively straightforward for most of the materials as it is simply a matter of saving the files to the RTF format. Some of the material, such as diagrams and tables, cannot be saved as RTF. Thus, these were made accessible to the student by mainly describing the tables and diagrams in text. Fortunately, most of the diagrams in the course were simple Unified

Modeling Language (UML) diagrams and were able to be described in text, which, subsequently the blind student could access using JAWS.

An alternative to describing diagrams in text is to reproduce tactile diagrams using a special printer. Although the Disability Liaison Unit of the university provides such a service, it was deemed not necessary in the case of AD2.

Course materials were provided to the student ahead of time and he preferred to listen to the material in his own time. He also attended the lectures to listen to the material being delivered live. In the lectures, I usually draw diagrams on the whiteboard to assist in explaining the material being delivered. Thus, I had to pay careful attention to orally describe the diagrams as opposed to simply pointing to parts of them. I initially thought visually-abled students would find this tedious, however, I received no negative feedback in that regard. I am not sure if they even noticed or realized what I was doing, or perhaps, they understood the compromise being undertaken for the sake of the blind student.

In regard to tutorials, it was at times difficult to assist the blind student during the regular tutorials as providing him with personal assistance is time consuming compared to assisting other students. Although he had special needs, I did not think it was fair to spend a large amount of time with him during the tutorials at the expense of other students. Thus, the blind student was provided with an additional weekly 2-hour one-on-one face-to-face session throughout the semester. Typically, he would be assisted, given explanations, and provided with examples about a number of different concepts in these sessions. He would then be assigned work to carry out during the regular tutorial sessions he attended with the other students. This arrangement worked fairly well throughout the semester.

Once course materials were made accessible to the blind student, the suitability of the topics covered in the course for him was investigated. Most of the topics were suitable for him, with the exception of graphical user interface (GUI) design and implementation. Although the blind student had some experience with GUIs and would likely be able to create GUIs in Java, I decided that it was not worthwhile. I was not convinced of the value of teaching a blind student to design and implement GUIs in Java in the restricted time frame of one semester. Instead of GUIs, it was decided to teach the blind student how to implement text-based menus. I could clearly see the value of him learning to design and implement such menus. The text-based menus would definitely be accessible with JAWS, and their implementation would require the use of numerous decision and looping structures to create multi-level menus.

3.4 Discussion and Recommendations

Although most AD2 students find Eclipse to be useful, it is disadvantageous to their learning as it is used to automatically generate some of the code. This became obvious by the end of the second week of the semester when the blind student, who had been manually typing all the code, had learnt more syntax and understood coding convention better than most of the others student. This was especially noticeable about code that is automatically generated by Eclipse, such as class and method headers.

In regard to coding convention, it was found that the style shown in Listing 2 was suitable for the blind student. This style places opening and closing braces individually on a line, annotated with a comment to indicate the start and end. The annotated comments are particularly useful as without them, JAWS would simply read out “opening brace” or “closing brace”, and the blind student would be unable to determine to which construct the opening or closing brace belonged. The annotated comments also allowed the blind student to better understand the code and to more easily navigate through and search for particular sections of the code when debugging.

```
public class Example
{ // Start class Example
  public static void main(String[] args)
  { // Start main()
    if (true)
    { // start if
      // ...
    } // end if
  } // End main()
} // End class Example
```

Listing 2. Recommended Coding Layout Convention.

In terms of the case-sensitivity of the Java programming language, the blind student did not have any difficulty as JAWS uses a different pitch for lowercase and uppercase letters. Although, one difficulty with JAWS is that its output does not seem to differentiate the spaces in between words. For example, the output for both “getName()” and “get name ()” is identical. This poses some difficulty for the typical Java naming convention, as the blind student must use JAWS to read each individual character to determine the difference.

4 CONCLUSION

The challenge faced was to teach a blind student object-oriented Java program. Although I had experience in teaching programming to visually-abled students, I had never taught programming to a blind student previously. Due to my lack of experience with the blind, I was unaware of the existence of JavaSpeak, a plug-in for the Eclipse IDE to facilitate programming for the blind.

In my approach, I advised the blind student to use the Textpad text editor to edit programming code. This was due to its compatibility with the screen reader, JAWS, and its lightweight and simple interface. Although unknown to me at the time, it is comforting to know that Textpad was recommended for blind programmers by blind consultants in another study (Bigham *et al.* 2008).

I also recommended the blind student to use a particular style of code layout, which included the annotation of the start and end of programming structures with comments. This is a common technique used by blind programmers (Smith *et al.* 2000), which, once again, was unknown to me at the time.

One issue the blind student faced with compiling Java applications from the command-line interface was the verbose output of errors produced by `javac`, a Java compiler. JAWS attempted to read all of the output generated by `javac`. To overcome this problem, I provided the student with a batch file that compiles given Java files and redirects the output to a text file. The batch file then opens the text file in a text editor for the student to browse.

Although JavaSpeak is designed to assist blind programmers to program in Java, from the material available, it was unclear whether JavaSpeak deals with this particular difficulty or with run-time errors. From the available material, it seems that JavaSpeak focuses on the development of code only as there was no mention of how it facilitates the compilation and debugging process for blind programmers.

In hindsight, it was very unfortunate that I was unaware of JavaSpeak at the time. Although the approach undertaken in this paper is relatively easy, practical, and effective (the blind student passed with course with high marks), the difficulties and challenges encountered indicate a strong need for such specialized applications to assist the blind to program. One area that appears to be lacking is assistive features to help the blind debug programs more effectively. Such features will greatly help in making programming more pragmatic for blind programmers.

The experience of teaching a blind student programming has added a new dimension to my teaching. I am now aware of different studies related to teaching programming to the blind and I am now more knowledgeable and better equipped to do so in the future.

Acknowledgements

I would like to thank the blind student who enrolled in Applications Development 2. Through teaching him, I gained an insightful view of what it is like to use a computer and program without sight. Furthermore, I would like to thank my colleagues who have taught the blind student before me, especially Vince Bruno, who shared his experience with me and was always happy to help and offer suggestions.

References

- Atkinson, M. T., Gucukoglu, S., Machin, C. H. C. and Lawrence, A. E. (2006), Making the mainstream accessible: redefining the game, in 'Sandbox 06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames', ACM, New York, NY, USA, pp. 21–28.
- Bigham, J. P., Aller, M. B., Brudvik, J. T., Leung, J. O., Yazzolino, L. A. and Ladner, R. E. (2008), Inspiring blind high school students to pursue computer science with instant messaging chatbots, in 'SIGCSE 08: Proceedings of the 39th SIGCSE technical symposium on Computer science education', ACM, New York, NY, USA, pp. 449–453.
- Cohen, R. F., Fairley, A. V., Gerry, D. and Lima, G. R. (2005), Accessibility in introductory computer science, in 'SIGCSE 05: Proceedings of the 36th SIGCSE technical symposium on Computer science education', ACM, New York, NY, USA, pp. 17–21.
- Francioni, J. M. and Smith, A. C. (2002), 'Computer science accessibility for students with visual disabilities', SIGCSE Bulletin 34(1), 91–95.
- Franqueiro, K. G. and Siegfried, R. M. (2006), Designing a scripting language to help the blind program visually, in 'Assets 06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility', ACM, New York, NY, USA, pp. 241–242.
- Gellenbeck, E. (2005), 'Integrating accessibility into the computer science curriculum', Journal of Computing Sciences in Colleges. 21(1), 267–273.
- Ludi, S. A. and Reichlmayr, T. (2008), Developing inclusive outreach activities for students with visual impairments, in 'SIGCSE 08: Proceedings of the 39th SIGCSE technical symposium on Computer science education', ACM, New York, NY, USA, pp. 439–443.
- Murray, I. and Armstrong, H. (2004), A computing education vision for the sight impaired, in 'ACE 04: Proceedings of the sixth conference on Australasian computing education', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 201–206.
- Sánchez, J. and Aguayo, F. (2005), Blind learners programming through audio, in 'CHI 05: CHI 05 extended abstracts on Human factors in computing systems', ACM, New York, NY, USA, pp.1769– 1772.
- Siegfried, R. M. (2006), Visual programming and the blind: the challenge and the opportunity, in 'SIGCSE 06: Proceedings of the 37th SIGCSE technical symposium on Computer science education', ACM, New York, NY, USA, pp. 275–278.
- Siegfried, R. M., Diakoniarakis, D., Franqueiro, K. G. and Jain, A. (2005), 'Extending a scripting language for Visual Basic forms', ACM SIGPLAN Notices 40(11), 37– 40.
- Siegfried, R. M., Diakoniarakis, D. and Obiano-Agu, U. (2004), Teaching the blind to program visually, in 'The 21st Annual Conference on Information Systems Education (ISECON 2004)'.
- Smith, A. C., Cook, J. S., Francioni, J. M., Hossain, A., Anwar, M. and Rahman, M. F. (2004), 'Nonvisual tool for navigating hierarchical structures', SIGACCESS Accessibility and Computing (77-78), 133– 139.

Smith, A. C., Francioni, J. M. and Matzek, S. D. (2000), A Java programming tool for students with visual disabilities, in 'Assets 00: Proceedings of the fourth international ACM conference on Assistive technologies', ACM, New York, NY, USA, pp. 142– 148.