JavaCHIME: Java Class Hierarchy Inspector and Method Executer

Pallavi Tadepalli
Department of Computer and Information Science
University of Mississippi
201 Weir Hall
University, MS 38677
1-662-915-7396
pallavi@cs.olemiss.edu

H.Conrad Cunningham
Department of Computer and Information Science
University of Mississippi
201 Weir Hall
University, MS 38677
1-662-915-5358
cunningham@cs.olemiss.edu

ABSTRACT

Java has emerged as one of the most dynamic programming languages today. During the past eight years it has become a leading choice as the programming language for an introductory course in computer science. Object-oriented programming (OOP) is often considered to be complex and difficult to grasp by both beginners and experienced procedural language programmers. Using Java in an introductory programming language serves a dual purpose; it teaches the syntax and constructs of the Java language as well as general object-oriented programming concepts. This paper describes an instructional environment for Java applications called JavaCHIME. This graphical class browsing and execution environment allows users to examine classes and objects interactively and execute individual methods within them without actually creating a class containing a `main` method for testing purposes. JavaCHIME serves as a pedagogical tool where novice Java programmers can examine variables and methods, interactively alter the values of variables and test the methods.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments – *Graphical environments, integrated environments.*

General Terms

Languages

Keywords

Java, pedagogy, class browser, object inspector.

ACMSE'04, April 2–3, 2004, Huntsville, Alabama, USA.

1. INTRODUCTION

During the past eight years, Java has emerged as a frequent choice for the programming language to use in the introductory programming course (CS1) in colleges. However some consider object-oriented programming (OOP) languages like Java to be too complex for beginners to grasp.

While lectures are useful in specifying concepts, it is "properly designed labs that constitute an integral component of CS1." [9] The success of labs lies in the presence of tools that support an exploratory approach allowing students to "develop skills and processes for discovering behaviors" [9] of programs. The presence of appropriate tools in the lab for teaching CS1 helps the students learn the concepts of object orientation and programming using Java. It is necessary that students understand that classes and objects are basic entities in object-oriented programming languages such as Java. In this regard, a shortcoming of Java is the `main` method. The `main` method introduces a procedural aspect into dealing with classes and objects. The program must be wrapped in a non-object-oriented main function to enable testing [4]. The signature of the `main` method itself consists of advanced concepts for a novice student.

The other pitfall that Java suffers from is complicated text input/output, more precisely text input. "Standard input requires several declarations to set up the input stream and conversion methods to read numeric values." [7] These complex input mechanisms in Java can be simplified by using a graphical interface for accepting user input.

To address these issues, a graphical instructional environment, JavaCHIME has been implemented that:

- provides a class browser
- provides an object inspector
- allows dynamic creation of objects
- enables users to call methods and provide their arguments interactively

Thus, the users are not restricted to using the `main` method to test classes, and a simple GUI simplifies the input process. This tool promotes an exploratory approach to learning.

In the work described in this paper, a graphical class browsing and execution environment for Java applications called

JavaCHIME (Java Class Hierarchy Inspector and Method Executer) was built to address the above requirements.

2. PEDAGOGY

JavaCHIME was designed to help beginning students learn the principles of programming and object orientation. Since beginners find it difficult to grasp these concepts it is useful for them to have access to special instructional programming environments instead of the commercially available integrated development environments (IDE), which are meant for experienced professionals. For ease of use, a simple programming environment ideally should have a graphical user interface [4].

Students are often intimidated by the logistics of writing programs; this shifts their focus away from concentrating on learning the object-oriented concepts. Support from the environment proves to be a critical factor to enable the students to focus on the object-oriented concepts they need to learn.

In plain Java, a `main` method is required to create the initial objects. The JavaCHIME tool enables students to explore existing classes and create objects interactively. Several objects may be created for one class; each exists as a separate entity, while exhibiting similar behavior. This process is simplified by representing the created objects graphically. Once created, the tool allows the objects to be maintained interactively. Users can call methods and inspect the state of the object. Students can thus identify objects and classes as the basic entities in object-oriented programming. The tool can take input from the user interactively, without burdening the students with complex code. They can modify existing classes and easily view the results of the changes made. They can incorporate pre-written classes into Java code and learn to discover the functionality of classes along with code reuse [11]. This direct interaction "enables a student to better relate to black-box testing, software specification and software validation." [9]

In JavaCHIME, the functionality is achieved by using class browsers and object inspectors. While there are a number of programming environments and environments that support class browsers, not all of them are suitable for use as teaching aids. Two significant environments that have a number of features for Java programmers are the NetBeans [2] and Eclipse [10] platforms. There are also several other commercial Java IDEs. These environments, while providing ease in editing of programs, are lacking in some significant features such as supporting dynamic creation of objects, interactive calls to methods, interactive argument specification and object inspection. It is these deficiencies in conventional integrated development environments (IDEs) that fueled the need for a new environment like JavaCHIME.

3. JavaCHIME

JavaCHIME is a graphical class hierarchy inspection and method execution environment. It is an easy to use, object-oriented programming environment with support for teaching. It is built upon the Java 2 Standard Edition (J2SE) platform using the Java Reflection and the Java Swing application programming interfaces (API).

The environment provides a simple graphical environment that:

- focuses on object-oriented principles of classes and objects
- permits users to interactively create objects and execute methods
- allows testing of methods without the need for the `main` method
- accepts user input for parameters through interactive dialog boxes

Overall, it is an environment that supports the teaching of Java from a truly object-oriented perspective.

3.1 Overview

The JavaCHIME environment (Figure 1) is divided into four panels, each for a specific function. The four panels are the class browser, the object inspector, the source-file display, and the console display.

The tool allows users to work by selecting files individually or by loading an entire directory. The environment allows loading of only Java class files or directories. When the environment starts to load a class file, it first checks for a newer version of the source file. If it finds one, it asks the user whether to compile the newer version and load it.

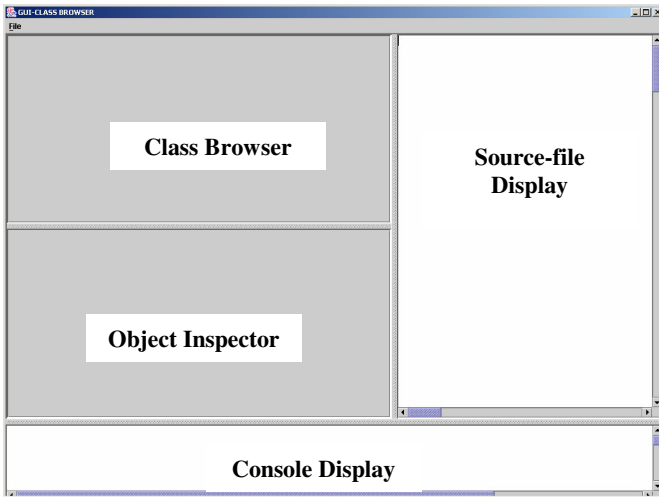

Figure 1. Opening view of JavaCHIME

3.1.1 Class Browser

The class browser provides a view of a program's class hierarchy. The loaded view of a class is a tree structure displaying the name of the class as a node with class properties as children of that node. The properties are represented with appropriate nodes like interfaces (being implemented), superclasses, constructors, fields and methods. The constructors node displays the constructors of the class, while the fields node displays the static and instance variables and the methods node displays the instance methods. The superclasses node displays the superclass of the class along with its own superclass and this hierarchy continues until the root (`java.lang.Object`) is reached.

The right-click menu on the class node (Figure 2) displays the constructors and the static methods and provides an option to view the source file. The user can create a new object of the class by selecting the appropriate constructor from the menu. The system then prompts the user for a name for the object

(Figure 3). This is to simulate an environment similar to creation of an object in the user's `main` method. The system prompts the user for the arguments of the constructor using a dialog box (Figure 4). Once the arguments are entered, the system creates the object and gives it the name specified by the user. The new object is represented by a node in the object inspector. The object created by JavaCHIME in Figures 3 and 4 is equivalent to the statement:

```
hello a= new hello(Integer, int)
```

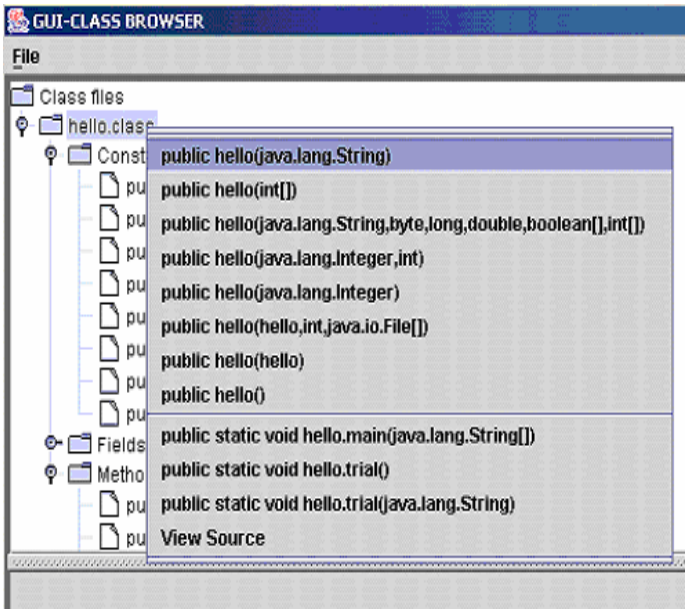

Figure 2. Class browser with right-click menu on class showing constructor selected

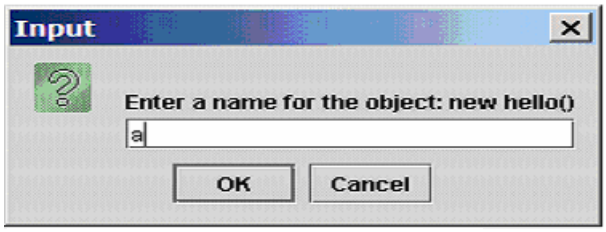

Figure 3. Dialog box for name of the object

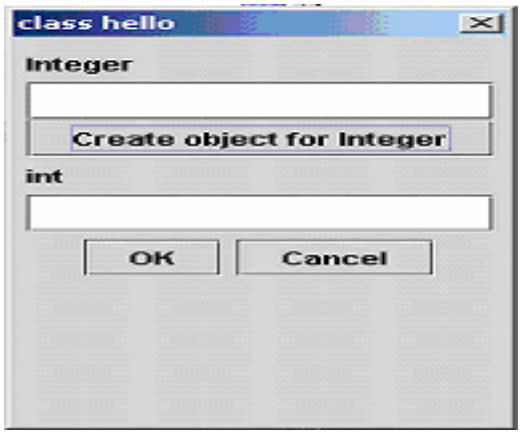

Figure 4. Specifying parameters for creating objects

If an argument of a constructor is non-primitive, then the system asks the user to create an object for the non-primitive argument. The constructors of that non-primitive type are displayed and an appropriate constructor can be chosen to create the needed object (as described above). The user is then returned to the original dialog box where the newly created non-primitive argument can be specified using its object name. This process continues until all arguments have been entered. The arguments entered may be primitive or non-primitive, scalar or array. The values of array arguments are accepted as comma-separated entries (Figure 5). If the array elements are objects, then the object names are entered.

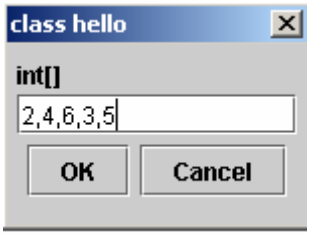

Figure 5. Accepting array parameters

The right-click menu on the class node also allows execution of static methods of the class (Figure 6). Selection of a method causes its execution. Parameters can be specified in a dialog box that is displayed as needed. The result of the execution is shown in the console window at the bottom of the tool (Figure 7).

Figure 6. Static method being selected

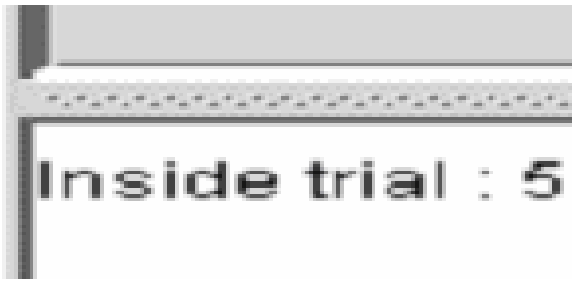

Figure 7. Output of static method executed

3.1.2 Object Inspector

The object inspector allows inspection of information about an object and also enables operations to be performed on an object. An object that is created is displayed as a clickable node in a tree structure that consists of other created objects. The right-click menu displays the instance methods for that object and an option to inspect the value of the object (Figure 8).

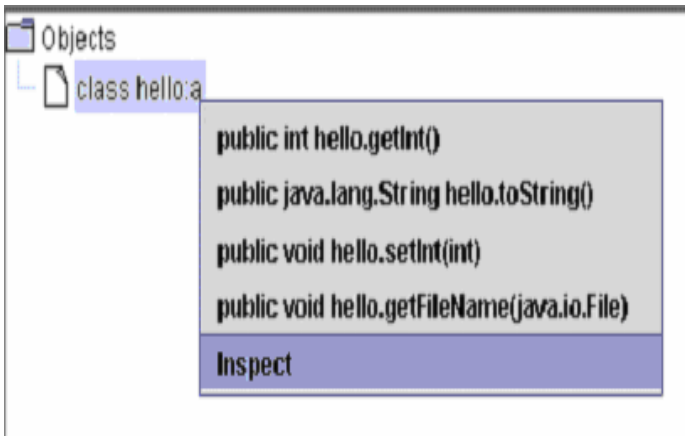

Figure 8. Object Inspector showing an object and its instance methods on the right-click menu

Method invocation proceeds much the same way as static method invocation on classes. If the method has a return type other than `void`, then the system first creates an object of the return type. It prompts the user for a name. Inspection of the object displays its values which consist of values of the static and instance variables declared in the class (Figure 9).

Figure 9. View of the object values

3.1.3 Source File Display

The source file is displayed when the **View Source** option is selected on the right-click menu of the class node in the class browser (Figure 2). The source is displayed in a plain text area that can only be viewed, but not edited (Figure 10).

3.1.4 Console Display

The console display (Figure 11) merely mirrors the output of the program execution shown in the terminal window. Thus the user is saved from the inconvenience of switching windows to view the output.

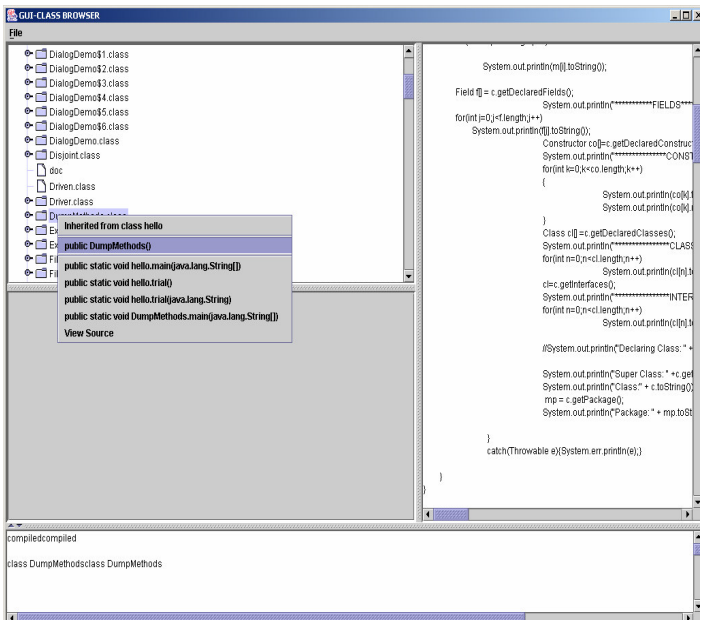

Figure 10. View of the source file in JavaCHIME's right panel

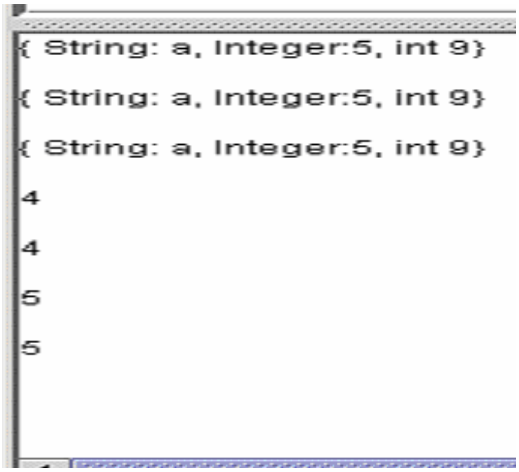

Figure 11. View of the console display

4. EXISTING ENVIRONMENTS

Environments to support teaching of object-oriented concepts have been designed by adapting traditional procedural environments to a particular object-oriented language.

These environments lack in some of the fundamental features mentioned above. Kolling summarized these problems as [6]:

- not object-oriented
- too complex
- focused on user interfaces

Object-oriented languages like Smalltalk [8] and Eiffel [3] have environments that do not suffer from the above problems and can be used as teaching environments. There are few teaching environments readily available for Java. A readily available teaching environment that has been able to solve the above problems has been the BlueJ [6] environment for Java. It was developed and implemented successfully at Monash University in Australia. Another significant application, DrJava has been successfully implemented and developed at Rice University [1]. A comparative analysis of JavaCHIME, BlueJ and DrJava is addressed in the next section.

5. DISCUSSION

The JavaCHIME tool is *object-oriented* and has been built with the aim of teaching object-oriented concepts. The system has been designed and implemented to allow extensibility.

JavaCHIME provides an *easy to use* interface. There is direct interaction with classes and objects. It allows students to enter the required values. It is not necessary for students to use a `main` method for testing classes. The presence of the class browser promotes *code reuse*. The difference between static and instance methods is highlighted by distinguishing their appearance at different places. Static methods (class methods) are available for invocation only at the class node while instance methods are available with a specific instance (object).

The system has been designed to simulate a real programming experience which allows users to create objects, test the methods, store the results of the method invocation and use the results later in the course of program execution. At a basic level, this functionality has been successfully provided. The tool does not currently provide an integrated editor; however, it does provide the option to use Notepad as the default editor on a Windows machine. Another missing feature is a debugger. These integrated features are available in the BlueJ [5] environment.

There is a fundamental similarity on an operational level between BlueJ and JavaCHIME; however, there are certain features that are not well addressed in BlueJ, which have been incorporated in JavaCHIME.

In BlueJ, it is not possible to load a single class and perform operations on it. Also, it requires the presence of the source and the class files to allow processing. The presence of only a class file without its corresponding source file is not meaningful. JavaCHIME can work with single or multiple classes as well as directories loaded with classes. It does not require the presence of the source code.

In BlueJ, every directory opened is converted into a BlueJ project type. While this is convenient in speeding up reload of the same directory later, its demerit is that the sub-directories located are also converted into BlueJ project types which are now an integral part of the project type for the directory. This prevents loading of those individual sub-directories without first loading the parent directory project. The Java class hierarchy inspector and method executer allows loading of the desired directories/sub directories.

Passing parameters that are of some object type is confusing in BlueJ. When passing parameters that are of a certain object type in JavaCHIME, the user is prompted for entering the name of the object. If the object does not exist, the user is prompted to create the object, which is saved and can be passed as a parameter. Such a feature is found to be missing in BlueJ.

Another related application, DrJava [1], provides a "read-eval-print loop" where students can evaluate Java expressions and statements. Every time the current class is compiled, it is reloaded. Thus, the focus of both JavaCHIME and DrJava is on interactive software development [1]. However, DrJava provides only a single medium for describing programs, namely program text [1]. There is no graphical interface for interactively creating objects and method calls. Additional features provided by DrJava are brace matching and updating the highlighting of comments and quotations [1]

A significant improvement in JavaCHIME will be creating sessions for each user, so that he/she may return to the tool at a later stage and work with the tool. It will involve saving the state of the object inspector, the class browser and the text display. Once the contents are serialized and stored, it is necessary to recreate them at a later stage when the user wants to work with them again. A decision has to be made about how much data is to be serialized and for how long it will be kept.

6. CONCLUSION

The pedagogical purpose of the JavaCHIME tool is served since it allows users to interact with classes and objects and solves the two underlying problems of using Java to teach beginning programming – the `main` method and text-based input/output. Integration of the pedagogical features juxtaposed with an easy to use, flexible environment has produced a graphical class hierarchy inspector and method execution environment, JavaCHIME.

7. ACKNOWLEDGMENTS

This project was supported, in part, by a grant from Acxiom Corporation titled "The Acxiom Laboratory for Software Architecture and Component Engineering (ALSACE)". This paper was critically reviewed by Hui Xiong and Ravi Darbhamulla whose valuable suggestions were also incorporated.

8. REFERENCES

[1] Allen, E., Cartwright, R., and Stoler, B., "DrJava: A Lightweight Pedagogic Environment for Java," *Proceedings of SIGCSE'02*, Covington, Kentucky, March 2002.

[2] Boudreau,T., Glick, J., Greene, S., Spurlin, V., and Woehr, J., *NetBeans$_{TM}$ The Definitive Guide*, O'Reilly, October 2002.

[3] Eiffel Software Inc., "EiffelBench: The Dream Object-Oriented Environment," http://archive.eiffel.com/products/bench/page.html, Accessed: September 12, 2003.

[4] Kolling, M., "The Problem of Teaching Object-Oriented Programming, Part 2: Environments," *Journal of Object-Oriented Programming*, 11(9):6-12, 1999.

[5] Kolling, M., "BlueJ--The Interactive Java Environment," http://www.bluej.org/, Accessed: April 03, 2003.

[6] Kolling, M., Quig, B., Patterson, A., and Rosenberg, J., "The BlueJ System and its Pedagogy," *Journal of Computer Science Education,* Special Issue on Learning and Teaching Object Technology, Vol. 13, No 4, December 2003.

[7] Lewis, J., "Myths about Object-Orientation and Its Pedagogy," *Proceedings of SIGCSE'00*, ACM, Austin, Texas, 245-249, March 2000.

[8] Lount, P., "A Brief Introduction to Smalltalk." http://www.smalltalk.org/#ABriefIntroduction, Accessed: April 03, 2003.

[9] Roumani, H., "Design Guidelines for the Lab Component of Objects-First CS1," *Proceedings of SIGCSE'02*, Covington, Kentucky, March 2002.

[10] Shavor, S., D'Anjou , J., Fairbrother, S., Kehn, D., Kellermen, J., and McCarthy, P., *The Java Developer's Guide to ECLIPSE.* Addison-Wesley, June 2003.

[11] Smith, P., and Boyd, G., "Introducing OO Concepts from a Class User Perspective," *The Journal of Computing in Small Colleges*, Volume 17 Issue 2, December 2001.