JIGSAW, A PROGRAMMING ENVIRONMENT FOR JAVA IN CS1

Carl Burch
Hendrix College, 1600 Washington Ave, Conway AR 72032
501-450-1377
cburch@cburch.com

ABSTRACT

This paper introduces a new open-source, cross-platform programming environment designed specifically for Java-based CS1 courses. Jigsaw, as it is called, uses an intentionally simple single-window interface, incorporating several features useful for the classroom context. One such feature is that upon successful compilation, the environment checks the student program for common student errors with Java, such as mistakenly writing "`void`" as a return type on what is intended as a constructor declaration.

INTRODUCTION

Instructors of Java-based CS1 courses face many choices of programming environments. Some choose a traditional command-line compiler coupled with a popular text editor. Others prefer an integrated development environment and choose a professional IDE such as Eclipse, JCreator, or Netbeans. Many feel that the number of features in such professional environments is a distraction for students, and they choose an environment designed specifically for beginning students. The most prominent choices in the latter category are BlueJ, DrJava, and jGrasp [2, 4, 6].

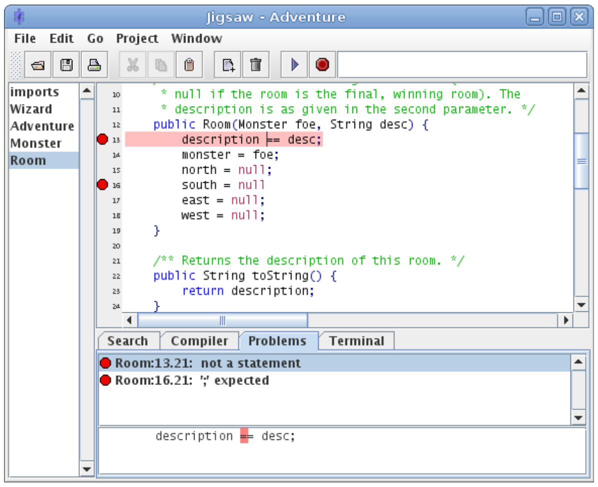
In the author's view, each of these environments suffers either from undue complexity or excessive focus on a particular pedagogy. Hence, the author developed another introductory programming environment, named Jigsaw. The most notable difference between Jigsaw and most other IDEs is an aggressive approach to identifying problems commonly encountered in beginners' programs, such as an instance variable without "`private`" protection or a class whose name starts with a lower-case letter. (Eclipse identifies some such warnings, but Jigsaw has a more extensive set corresponding to typical beginner errors.) Also unlike most other IDEs, Jigsaw uses a one-window interface with a minimum of clutter; it stores each project in a single file to aid with transferring projects between computers; and it also treats saving a project and executing it as separate concepts, to accord more closely with student expectations from word processors.

Our college has used Jigsaw in CS1 for eight semesters, with three different instructors, and it has proven an effective tool. Jigsaw is now publicly available as open-source software released under the GPL. Jigsaw works under any system with Java 1.4 or later installed, including Windows, MacOS X, and Linux systems. Those who are interested are free to download the software through Jigsaw's Web page, *http://www.cburch.com/jigsaw/*.

This paper introduces Jigsaw through a discussion of its more prominent features, and it discusses some experiences with using Jigsaw as an introductory programming tool.

FEATURES

Perhaps the most important feature of Jigsaw is its simple one-window interface, shown in Figure 1. Besides the menus and toolbar, the interface includes three major regions. The list on the window's left allows the student to select among the classes that constitute the program or to edit the imports that are common to all the classes. The currently selected item is displayed in the major portion of the window for editing. And at the window's bottom is a tabbed pane, primarily for viewing compiler output and interacting with the user's program as it executes.



```
10      * null if the room is the final, winning room). The
11      * description is as given in the second parameter. */
12     public Room(Monster foe, String desc) {
13         description == desc;
14         monster = foe;
15         north = null;
16         south = null
17         east = null;
18         west = null;
19     }
20
21     /** Returns the description of this room. */
22     public String toString() {
23         return description;
24     }
```

```
Room:13.21:  not a statement
Room:16.21:  ';' expected
```

```
        description == desc;
```

Figure 1. The main Jigsaw window.

This design already reflects a central design decision in Jigsaw: It assumes that the program consists of a single file of classes (all in the default package – i.e., with no package line). Placing all classes in one file has the disadvantage that the compiler must process the entire program with each compilation, but this is not a problem for the scale of program typical to CS1 and CS2. A bigger drawback traditionally is scrolling through a large file, but Jigsaw resolves this by splitting the file into sections which are listed on the window's left side. Thus, using a single file does not feel significantly different from using multiple files. The one-file approach has a particular advantage when transferring the project from one computer to another: The student has only to transfer a single file. This is particularly convenient when students submit their programs electronically.

Jigsaw supports all the standard text-editing features, such as text highlighting, multiple undo, line numbers, brace matching, and automatic indentation. One standard feature that many text editors include but only through a customization option is a vertical bar to indicate when lines are longer than 80 characters. In Jigsaw this bar is present by default, to encourage students to keep their programs in a format that prints onto paper nicely. This is important for those of us who typically grade CS1 assignments on paper printouts.

Executing Programs

One feature somewhat unusual to Jigsaw is that its toolbar includes a one-click button for compiling and executing a program. When the user clicks this button, Jigsaw writes the current project into a temporary directory, compiles the program from there, and executes the resulting file. This design avoids the common difficulty of beginners wondering why their changes to the program are ineffective when they simply have not saved the changes to disk before compiling. It also separates the process of saving a project from the process of using the project; this fits with the mental model developed by users of word processors, where one can edit and print a document without saving it.

Because Jigsaw works in a temporary directory, the user does not have to worry about using the correct filename or viewing the ".class" files generated by the compiler.

Behind the scenes, Jigsaw uses the `javac` and `java` command-line tools from Sun's JDK, which must be installed in order for Jigsaw to work. When executing the program, the program executes in a separate process; this protects Jigsaw from major problems such as infinite loops in the student's program.

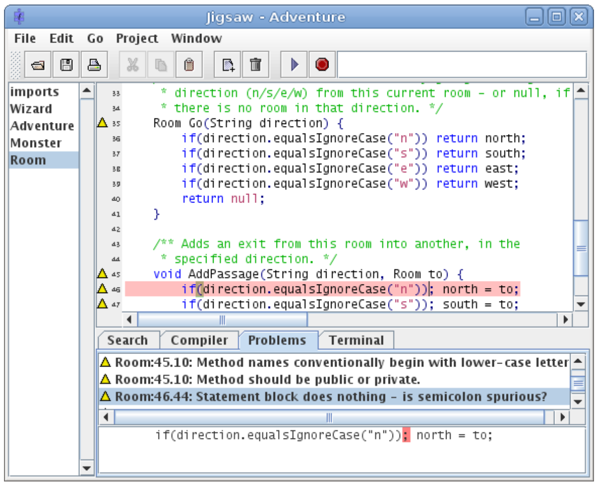
For programs using text-based I/O, Jigsaw includes a Terminal tab at the bottom of the window; this tab is selected after compilation completes successfully. Including this as part of the window saves the user from having to switch between windows. Regular output appears in black, standard error appears in red, standard input appears in green, and Jigsaw messages are displayed in blue.

When the user requests Jigsaw to execute a program, Jigsaw first terminates any previously created process. This eliminates a common problem where users end up with several processes representing previous program versions. Executing the same program multiple times simultaneously is seldom desired, particularly for CS1.

Some instructors prefer to create assignments that receive input through the command-line arguments. This is easy with Jigsaw: The user enters the command-line arguments in a text field in the toolbar before clicking the button for executing.

Compiler Errors and Warnings

If the student program has any errors preventing compilation, Jigsaw interprets the output of Sun's compiler and displays the error messages in an interactive list under its Problems tab. The user may click on any element of the list, and Jigsaw goes to the relevant line, which it also highlights with a stop-sign icon indicating that there is a problem with the line. Figure 1 illustrates this. As soon as the user edits the line, the stop-sign icon dims. When the user adds or deletes lines, Jigsaw adapts all error indicators accordingly. The result is that a user can hope to address several independent error messages from a single compile.



```
     * direction (n/s/e/w) from this current room - or null, if
     * there is no room in that direction. */
    Room Go(String direction) {
        if(direction.equalsIgnoreCase("n")) return north;
        if(direction.equalsIgnoreCase("s")) return south;
        if(direction.equalsIgnoreCase("e")) return east;
        if(direction.equalsIgnoreCase("w")) return west;
        return null;
    }

    /** Adds an exit from this room into another, in the
     * specified direction. */
    void AddPassage(String direction, Room to) {
        if(direction.equalsIgnoreCase("n")); north = to;
        if(direction.equalsIgnoreCase("s")); south = to;
```

```
Room:45.10: Method names conventionally begin with lower-case letter
Room:45.10: Method should be public or private.
Room:46.44: Statement block does nothing - is semicolon spurious?
```

```
        if(direction.equalsIgnoreCase("n")); north = to;
```

Figure 2. Warnings displayed in Jigsaw.

If a compile is successful, Jigsaw goes through another stage before executing the program: Jigsaw processes the program itself and attempts to discover some common user errors. Any detected problems are tagged as warnings, which again are displayed under the Problems tab with a yellow triangle icon, as Figure 2 illustrates. Regardless of whether warnings are found, Jigsaw still displays the Terminal tab and begins to execute the program. But if there are warnings, it first prints into the terminal a blue line reading: "Warnings under Problems tab should be addressed, but program will proceed." This message has proven effective at guiding students to address the warnings identified by Jigsaw.

Supporting a broad set of warnings is a particularly important feature to Jigsaw; it currently identifies 16 different warning categories. Each of the 16 types can be disabled through the Preferences window.

a) Variables within a class should be declared private.
b) Constants within a class should not have default protection.
c) Methods should not have default protection.
d) Method named same as class but is not a constructor.
e) Method names conventionally begin with a lower-case letter.
f) Variable names conventionally begin with a lower-case letter.
g) Constant names conventionally have only capital letters.
h) Type names conventionally begin with a capital letter.
i) Null statement block (‘`while(true); {`…’).
j) Local variable declaration hides other variable.
k) Integer division results in integer (e.g., ‘`1/2`’ is 0).
l) Use ‘`str`’ rather than ‘`new String("str")`’.
m) Variable changes should occur as their own statements.
n) Assignment to itself has no effect (‘`x = x;`’).
o) Boolean assignment within condition (‘`if(flag = value)`…’).
p) Assignment to parameter has no effect outside method.

The warnings fall into two basic categories: Errors that lead to erroneous program behavior and stylistic errors that the student would otherwise never recognize until the program is submitted.

An example of a programming error is (i), a superfluous semicolon between the condition of a loop and the braces that supposedly contain the loop's body. The Java compiler accepts this, though when executed it will result in an infinite loop. This behavior baffles beginning programmers, still unaccustomed to loop syntax. Even when a beginner shows the program to an experienced programmer, the expert often has difficulty noticing this extra semicolon, since it's small and the error is not one an expert would ever make.

Another programming error is integer division where none is intended (k). This is difficult to identify in general, since sometimes integer division is intended. But Jigsaw at least identifies when the user attempts to divide two constants: If the user writes "`1/2`" in a program, it is tagged with a warning.

A final example of a beginners' error is including a return type "`void`" on what is intended as a constructor (d). Including "`void`" creates an instance method rather than the intended constructor. If it's meant to be a no-arguments constructor, the program will compile successfully, but the "constructor" will never execute. Again, students rarely can discover the source of the problem on their own – and even experienced students have difficulty identifying the problem. But Jigsaw identifies it automatically.

Two examples of the stylistic warnings that Jigsaw identifies are instance variables without "`private`" protection (b) or any methods whose names begin with a capital letter (e). Were these not tagged by Jigsaw, the student would have no way of knowing better until the instructor gives them feedback in their assignment evaluation.

The students learn more effectively to avoid such stylistic errors if their development environment encourages them to fix it as soon as it occurs.

EXPERIENCES AND EVALUATION

Three different instructors have taught CS1 at our college using Jigsaw as the primary programming environment, and the current version represents a successive and substantial refinement of the original concept. In the first version, used in Fall 2005, we intended Jigsaw as a tool that completely removed the need for writing or learning the code involved in setting up a class. Instead, the programmer would add and edit the members of a class (such as instance variables, constructors, and methods) using GUI dialogs; the only code written would be the bodies of the constructors and methods. Such an environment is commonly associated with Smalltalk, such as the programming environment within Squeak [5]. The approach proved viable, but we abandoned the approach due to a lack of compatible textbook resources and the difficulty of printing programs for grading and of composing test questions without a syntax for describing the members of a class. Beginning with Spring 2007, we switched to the more traditional interface described in this paper. The current version represents four semesters of refinement from the Spring 2007 version.

We have found that Jigsaw accomplishes its objectives well. Students quickly learn how to use the tool, so they can concentrate on learning to program. We introduce students to Eclipse in CS2, and this transfer to a different system again introduces very little difficulty for students. After all, Jigsaw's user interface bears some similarity to Eclipse's.

Using a simple-to-install cross-platform tool is particularly convenient for students who choose to work on their own computers. The most problematic issue for student installations is getting the proper JDK release. Also, while Jigsaw attempts to automatically find the latest JDK installation, it doesn't always find it; and in that case, students must locate the relevant executables from the Preferences window, which is often difficult for students.

Ultimately, the greatest success we could have with Jigsaw is that students would not notice that there was anything unusual about using it. This ideal is an accurate way of describing how Jigsaw works in the classroom.

To evaluate Jigsaw, we distributed a brief survey in one section of CS1 at the end of the Fall 2008 semester. All twelve students completed the survey. Students were asked to rate the statement "Jigsaw was a good environment in which to learn Java programming." Seven students "strongly agreed" with this statement, while five "agreed" with it; none marked "neutral" or lower. Admittedly, this survey is of students who have not tried alternatives, so the result does not permit conclusions about Jigsaw relative to other IDEs or the command line. The most we can conclude is that Jigsaw works well enough that students did not perceive major problems with it.

Installation difficulties can be a hurdle in getting students to practice outside the classroom. Thus, the survey also asked students about their experiences with installing Jigsaw on non-laboratory computers. Of the ten who tried it, eight indicated that "Installing Jigsaw was a very easy process," while two checked "Installing Jigsaw was quite difficult, and I needed help to get it fully working."

FUTURE WORK

Though Jigsaw works quite well in its current state, of course there are plans for future enhancement. A student is currently working on enhancing the warning facility to perform dataflow analysis of the program. Through this, we hope to identify and flag as warnings any instance variables that should instead be defined as local variables, and we hope to identify variable assignments whose effects are never used.

Another enhancement that would be nice would be better support for programs that draw graphical shapes via educational libraries such as ObjectDraw or the ACM Java Task Force library [1, 3]. Optimally, the shapes drawn would appear in the Jigsaw window rather than a separate program window, removing the need to switch between windows. This is challenging because Jigsaw executes programs in a separate process to protect Jigsaw from user program errors such as infinite loops; the process cannot draw directly into Jigsaw's window.

Finally, Jigsaw could be improved through adding features for interactively manipulating objects. One model for this would be an area where a user can enter a Java expression to be evaluated immediately, as with DrJava's interactions pane or BlueJ's "code pad." Another is a workbench as with BlueJ, where the user can create and interact with objects through a point-and-click interface.

Despite these missing features, Jigsaw has proven to be very usable in its current form, whether the user programs interact using an educational graphics library, terminal I/O, or a GUI API such as Swing. Indeed, our CS1 students routinely write programs of all three forms with a minimum of challenges being presented by the development environment.

REFERENCES

[1] ACM Java Task Force, 2006, *http://jtf.acm.org/*, retrieved December 2, 2008.

[2] Allen, E., Cartwright, R., Stoler, B., DrJava: A lightweight pedagogic environment for Java, *SIGCSE* 2002, Covington, KY, 137-141.

[3] Bruce, K., Danyluk, A., Murtagh, T., *Java: An Eventful Approach*, Upper Saddle River, NJ: Prentice-Hall, 2005.

[4] Hendrix, T., Cross, J., Barowski, L., An extensible framework for providing dynamic data structure visualizations in a lightweight IDE, *SIGCSE* 2004, Norfolk, VA, 387-391.

[5] Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., Kay, A., Back to the future: The story of Squeak, a practical Smalltalk written in itself, *OOPSLA* 1997, Atlanta, GA, 318-326.

[6] Kölling, M., Quig, B., Patterson, A., Rosenberg, J., The BlueJ system and its pedagogy, *Journal of Computer Science Education 13* (4), 249-268, 2003.