Input/Output for a CS1 Course in Java

Some Considerations

Elliot B. Koffman

Computer and Information Science Department, Temple University, Philadelphia, PA, USA
koffman@temple.edu

Abstract: This paper discusses considerations for input/output for a first course in Java. It includes the opinions of several computer science educators regarding the pros and cons of using a non-standard package for input/output and the requirements of such a package. It also describes the evolution of a simple package for input that it is easy to implement using the Swing class and gives guidelines for using standard Java for simple input/output.

Key words: Computer Science 1 (CS1), Novice programming in Java

1. INTRODUCTION

1.1 Non-standard packages for input/output

Many Java textbooks for CS1 use simple packages for Java input/output. Some classes that were developed to simplify console input/output are: `ConsoleReader` [3], `Text` [2], and `SimpleInput` [1]. These classes were written because the textbook authors and teachers of CS1 recognized that console-based input in Java is difficult for novices to use. At the same time, a few packages were also written that permitted GUI-like interactivity. Examples of these packages and classes are: `simpleIO` [7] and `Java Power Tools` [8]. These packages were developed because their implementers recognized that console-based input/output was tedious and uninteresting to students who were accustomed to applications with Graphical User Interfaces (GUIs). The package developers also felt that novice students should be able to write programs with GUI-like interactivity before

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35619-8_15](https://doi.org/10.1007/978-0-387-35619-8_15)

L. Cassel et al. (eds.), *Informatics Curricula and Teaching Methods*

© IFIP International Federation for Information Processing 2003

mastering the Java APIs that would enable them to build their own GUIs: AWT and Swing.

1.1.1 Educator comments on using non-standard packages

In early April, 1999 there was a thread on the SIGCSE listserve that discussed the use of non-standard packages for teaching CS1 in Java. The messages seemed to be split fairly evenly between those who advocated their use and those who were against their use. The following comment by H. Conrad Cunningham (University of Mississippi) succinctly summarizes the two points of view:

"On one hand, use of simplified I/O or GUI packages can make Java easier to teach and use in CS1/CS2. So the various packages that are available with various textbooks can be helpful. And we want to convey to our students that it is easy to add on significant capabilities to Java and other such languages.

On the other hand, there is concern that use of a non-standard, add-on package might be confusing to first year students. Students tend to take what they learn as being part of Java. When they move to another environment, they are somewhat lost and confused. They have the overhead of relearning an I/O package or installing the one they are accustomed to. There is thus some sentiment to only teaching the "standard" (whatever that means with Java)."

A. Joseph Turner (Clemson University), makes the following point in favor of using packages:

"I haven't tried teaching CS1 or CS2 using only the classes in the standard Java library, but unless you want to limit yourself to reading only a character at a time or a line at a time (as a string), you don't want to use only the I/O facilities in standard Java. (There are some ways to do a bit better than reading only characters or lines, but the overhead and complexity of doing so are out of the question for beginning students as far as I am concerned.)"

David Arnow (Brooklyn College of City University of New York) makes the following argument against the use of packages:

"There is another argument in favor of NOT providing a special I/O package for student use in CS1/Java. While constructions such as

```
BufferedReader br = new BufferedReader(
    new InputStreamReader(
        new FileInputStream(
            new File("myinput"))));
```

may be introduced as "boilerplate", they also provide an opportunity to:

- emphasize the idea of classes as models, as repositories of particular sets of behavior.
- illustrate the use of composition
- foster the notion and UTILITY of abstraction

In connection with the latter, students, even CS1 students, can appreciate the fact (and the accompanying discussion) that the last two lines of the above code can be replaced with

```
System.in
```

or

```
new URL("http://www.acm.org").openStream());
```

or

all sorts of other things eventually

because each of these things play the "role" of an `InputStream`. In other words, if one's goal is to take an object-oriented approach in CS1 Java's I/O classes can be viewed as an asset, not a liability. (As a side note, this i/o stuff usually turns out to be the LEAST of the students' problems in CS1.)"

Michael G. Branton (Stetson University) advocates teaching basics of the AWT first instead of using packages:

"You can also do it with Java without that much fuss. I've taught the course 3 times now using AWT. you don't need very much of it to put a text field or 2 and a button on the screen, folks. There seems to me a lot of hand-wringing over this, but I've found that my students don't find this difficult to deal with at all. It's the algorithm development that's still the "hard part." My experience is that teaching them enough of the GUI for CS1 doesn't take much time at all. I still have plenty of room for the topics I consider important, and text fields and buttons give them something very concrete (in a virtual sort of way :-)) to think of as objects right off the bat."

Finally, Judy Bishop (University of Pretoria) makes the following points in favour of non-standard packages:

- "Java is an extensible language, with extensions coming via packages (APIs). In today's world, the ability to harness this power is more integral and more important for fresh minds to understand in *week one*, than a long-winded object instantiation, four brackets deep.
- An I/O class is an excellent case study in its own right as it illustrates exception handling, as well as tokenizers and string functions. It also does not have to be long: our `Text` class is 2.5 book pages of code, providing 10 methods for input, output and file opening.
- If a lecturer feels very strongly about teaching Java at the rock face, and is not convinced by point 1, then I would still suggest that when writing your own package! comes along in the course (which it ought to at some stage), then an I/O or Graph package is an ideal candidate."

1. There are rather strong opinions about the pros and cons of using packages.
2. If authors or teachers wish to use a package, they should think of the package as "training wheels" to give a student the feel for programming with GUIs in Java without the overhead of learning the AWT (or Swing) from the start.
3. Because the package will be discarded, it should be relatively easy to learn how to use it.
4. The package should be fairly easy to implement, so students have little difficulty understanding it as a case study.

2. EVOLUTION OF A PACKAGE FOR USER INPUT

In earlier papers [5, 8], Koffman and Wolz, described two packages for user interactivity in a Java course. The first, simpleIO [8], contained a class SimpleGUI with instance methods for displaying dialog windows and menus for user input. Class SimpleGUI also had instance methods for incorporating user interactivity in graphics programming and for simplifying file input/output. Finally, it provided as "output" a history of all user interaction with objects that extended class SimpleGUI. Class SimpleGUI was not easy for a beginner to use because a class performing input/output operations needed to instantiate SimpleGUI or extend it in order to call its methods. Also, it was not easy to implement and understand SimpleGUI because it was designed using the Java 1.0 AWT class. For these reasons, SimpleGUI did not meet criteria 3 and 4 in the paragraph above.

The second package [5] was more modest in its goals. It enabled novice programmers easily to display dialog windows for input, including menus. It also enabled students to display windows with simple messages, or to display text areas containing several lines of output. All methods are static methods, so there is no need to instantiate a class or to extend it. Also, all methods are relatively easy to implement (between 5 to 15 lines each), so the package can be used as a case study.

The input class KeyIn provides methods readChar(), readDouble(), readInt(), and readString(), which display a dialog window and return a data value of the type indicated by the method name. For example, the statement

```
int pennies =
    KeyIn.readInt("Enter the count of pennies");
```

displays the dialog window in Figure 1 and returns as its result the integer value (155) typed in by the user. For readChar(), readInt(), and readDouble(), if the user enters an item that is not the required data type, an

error window appears (see Figure 2). After the user presses OK, the dialog window reappears and the user has another chance to enter a valid data value. Also, the user can also restrict the range of values by supplying additional arguments:

```
int pennies =  
    KeyIn.readInt("Enter the count of pennies", 0, 100);
```

If the user enters an out-of-range value, an error window appears (see Figure 3) and the user is prompted again to enter a valid data item. Methods `readChar()`, `readDouble()`, `readInt()` call method `showInputDialog()` of Swing's `JOptionPane` class to display dialog windows, and they call method `showMessageDialog()` to display error message windows.

Figure 1. Dialog window for integer data

Figure 2. Error message for invalid data

Figure 3. Error message for out-of-range data

Method `readBoolean()` enables a student to display a window with buttons labels "Yes" (for true) and "No" (for false). The statement

```
boolean mood = KeyIn.readBoolean("Are you happy");
```

displays the dialog window show in Figure 4.

Method `readChoice()` enables a student to display a menu and returns an integer corresponding to the button pressed by the user. The statements

```
String[] coffeeChoices = {"Rain Forest", "Decaf",  
                           "Espresso", "Latte"};  
int myCoffee = KeyIn.readChoice("Select a coffee",  
                                coffeeChoices);
```

display the menu window shown in Figure 5 ("Rain Forest" is 0, "Latte" is 3).

Figure 4. Reading a Boolean value

Figure 5. A Simple Menu

A companion class, `UserOut`, has a method `displayResult()` that displays its argument string in a message window. For multiple line output, method `appendOutput()` enables a student to append output lines to a text area. Method `displayOutput()` displays the text area in a window. Method `clearOutput()` clears the text area. The statements below fill and display the text area shown in Figure 6.

```
for (int i = 0; i < 3; i++) {  
    int next = KeyIn.readInt("next number");  
    UserOut.appendOutput("next number is " + next);  
}  
UserOut.displayOutput();
```

Figure 6. Output window with text area

3. A MORE CONSERVATIVE APPROACH TO I/O

After presentation of a paper describing this package at ACM SIGCSE [5], the author began to rethink the advisability of using a non-standard class for performing output. The output methods of class `UserOut` do not provide sufficient benefit beyond what is available in standard Java 2.0 to offset the overhead of teaching students how to use these methods and later abandoning them when GUIs are introduced. Students can call `JOptionPane`'s method `showMessageDialog()` to display a string in a message window. Since it is desirable to teach students how to use the Java console window for debugging, students can also use the Java console window to display multiple line program output instead of using an output text area.

Although a non-standard package is still useful for input, it is reasonable and feasible to teach students how to call the methods of class `JOptionPane` directly. This would enable some faculty to restrict their students to standard Java if they wished, but would enable others to simplify their students' code with a non-standard package. This approach also ensures that students who use the package would understand how to implement its methods using standard Java.

As an example, the first statement below

```
String penniesStr = JOptionPane.showMessageDialog (
    null, "Enter the count of pennies");
int pennies = Integer.parseInt(penniesStr);
```

displays the dialog window shown earlier in Figure 1; the second statement converts the data item entered by the user (a string) to an integer that is stored in `pennies`. Unfortunately, an invalid data item would raise an exception that would cause the program to terminate because the code does not catch invalid numeric data. Students can write simple static methods that

```

public static int readInt(String prompt) {
    String xStr =
        JOptionPane.showMessageDialog(prompt);
    int xVal = Integer.parseInt(xStr);
    return xVal;
}

```

The standard Java call to method `showConfirmDialog()` below displays the Yes/No dialog window shown earlier in Figure 4. The assignment statement stores true or false in variable `mood`. Similarly, students can call `showOptionDialog()` to display a menu (Figure 5) and return a type `int` result.

```

int happyNum = JOptionPane.showConfirmDialog(null,
                                             "Are you happy?");
boolean mood = (happyNum == 0);

```

4. SUMMARY

The paper describes instructors' attitudes towards using non-standard packages for input/output in a first course in Java and some guidelines for developing and using such packages. It describes the evolution of such a package. Because the package is easy to implement and understand, it can be used as a case study to illustrate design techniques and to help students understand new Java concepts. The paper also describes a simple approach to using standard Java for input/output in which the console window is used for program output and dialog windows are used for input.

5. REFERENCES

- [1] Barnes, D. J., *Object-Oriented Programming with Java*, Prentice Hall, Upper Saddle River, NJ, 2000.
- [2] Bishop, J., *Java Gently*, Addison-Wesley Longman, Essex, England, 1997.
- [3] Horstmann, C., *Computing Concepts with Java 2 Essentials*, Second Edition, John Wiley & Sons, New York, NY, 2000.
- [4] Koffman, E., and Wolz, U., *Problem Solving with Java*, First Edition, Addison Wesley, Reading, MA, 1999.
- [5] Koffman, E., and Wolz, U., A Simple Java Package for GUI-like Interactivity, *Proceedings of 32nd Annual SIGCSE Symposium*, February, 2001, 11-15.
- [6] Koffman, E., and Wolz, U., *Problem Solving with Java*, Second Edition, Addison Wesley, Reading, MA, 2002.
- [7] Raab, J., Rasala, R., and Proulx, V. K. Pedagogical Power Tools for Teaching Java, *SIGCSE Bulletin*, 32(3), 156-159.
- [8] Wolz, U. and Koffman, E., *simpleIO: A Java Package for Novice Interactive and Graphics Programming*, *SIGCSE Bulletin*, 31(3), 139-142.