An Example Oriented On-Line Java Tutorial for University Students

Jianna J. Zhang and Huy Nguyen
Department of Computer Science
Western Washington University
Jianna.Zhang@wwu.edu
chanhuyn@hotmail.com

Abstract

As the number of Internet users grows exponentially all over the world, Java has become one of the most favored computer languages for Web programming. Java application and Web developers are in high demand. Java courses are offered by almost every university, college, and it even has a presence in high schools. To help students learn Java efficiently, educators have created all kinds of tutorials, short and long courses, white papers, Web help pages, menus, and books. Although there are numerous resources to help students learn Java, we find that there is still at least one important part missing: a set of well organized practical examples that correspond to the kernel elements of Java language. This insufficiency slows down the learning speed and brings frustration to students. In this paper, we present a different Java tutorial which is example oriented instead of explanation oriented. We believe that our tutorial can significantly reduce the above insufficiency.

1. Introduction

We present the design, frame work, and partial (knowledgebase) implementation of an example oriented on-line Java tutorial. The main goal of this project is to provide an efficient Java language learning tool for university students. The Java tutorial contains a set of well organized practical examples that correspond to the kernel elements of Java language defined by Sun Java Company [1]. Unlike previous Java language learning tutorials or other kinds of learning resources [1-19] which are explanation oriented, our tutorial is interactive example oriented. That is, for each Java concept, we create a set of short and simple examples with corresponding output. Students can click on the output button to see the program results or click on the SunJava button to see the original explanation and syntax.

This paper is organized as follows: in Section 2 we state the 5 principles that lay the theoretical foundation of our design and implementation; in Section 3 we give a general description of the overall design with emphasis on the knowledgebase component that has been currently implemented; in Section 4, we show sample tutorial examples with discussion on programming hints, common errors and cures; in Section 5, we give an overview of the software organization; and in the last section we discuss the impact of our tutorial and future plans.

2. The 5 Principles

We define the following 5 principles for an efficient example oriented Java tutorial:

(1). Examples First, Examples First!

We often hear from our students: “We need more examples!”, “More examples, please!”, or “All we need are examples!” Plain English is apparently insufficient; and a successful guided tutorial for Java requires a carefully crafted sequence of examples. Indeed, our experience suggests that the right sequence of examples enables students to master the language very quickly.

(2). Examples must be Short

Long examples confuse students and often obscure the feature illustrated. We decompose long examples into short examples so that the features being taught can be isolated conceptually. Exercises that combine these short examples give students the satisfaction of a first-hand experience programming in Java and thus arouse enthusiasm.

(3). Examples must be Simple

Unnecessarily complicated examples have the same bad consequences as ones that are too long. In addition, a complicated example is a bad exemplar, encouraging students to take advantage of arcane features of Java and develop bad programming style. If a simpler example can replace a more complicated example, replace it!

(4). Examples must be Correct

Needless to say, incorrect examples are a source of great confusion. Students invest authority in a tutoring system, especially if it is otherwise well designed, and incorrect examples can be remarkably difficult for them to unlearn. We must provide correct examples and also tips on how to avoid common errors. These tips should be given for every relevant example in the entire tutorial.

(5). Examples must be Interactive

Showing examples alone is not enough. We must interactively show the result of the examples. Immediate feedback speeds the learning process. Ideally, for each example, a student should be able to explore its meaning on a single mouse click. Students can trace the results back to its source code line by line to achieve immediate understanding.

3. General Description of the Tutorial Knowledgebase

The contents of our Java tutorial collaborate with the Sun Java library. There are 1,841 classes in the Java 2 (Platform Std. Ed., v1.3.1) library according to Sun Microsystems, Inc. [1], and these classes are mapped into 67 packages. The Tutorial does not re-state the syntax or re-explain each class/interface or package. It has at least one example and its output for each important and commonly used class or interface. Figure 1 shows the knowledgebase of the Java tutorial.

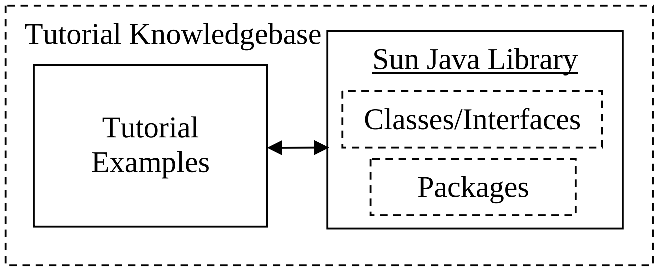


Figure 1. Tutorial Knowledgebase

Because our java tutorial is divided into 4 different levels: beginner, intermediate, advanced, and professional, the database is virtually divided. The virtual divisions of the Tutorial are "memorized" by the Tutorial's search engine. This search engine is an AI agent which is attached to the Tutorial database. When a request is passed through the Tutorial's user interface, the search agent will find the appropriate tutorial examples using Bayesian network, and the Web Page Maker (see Figurer 2) will display these examples on the user interface. We are currently implementing the AI search engine and therefore it will be discussed in a separated paper.

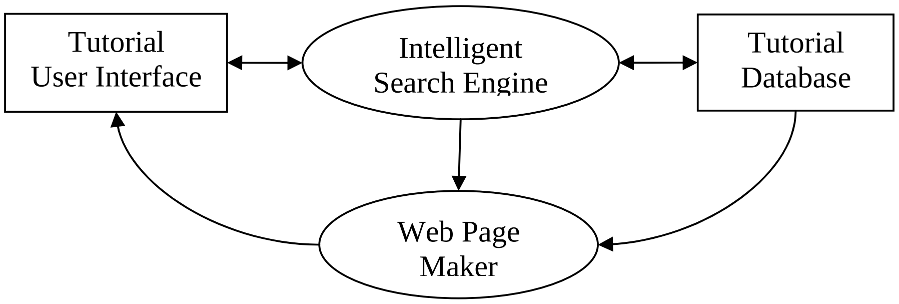


Figure 2. The Java Tutorial

4. Tutorial Examples

Experience tells us that it takes a long time for students to truly understand Java language and know the syntax well even if they have been exposed to Java for quite a while. In this section, we present sample programs and hints that give students short cuts leading to the mastery of the language.

4.1. Sample Useful Hints

We show useful hints with examples for commonly asked questions by our students, for example, "how to deal with path separators of different environments", "how to use command-line arguments", "how to use break and continue statements", "the main difference between String and StringBuffer", and so on. These hints are everywhere throughout the tutorial. The following examples are samples of these hints.

String Class:

A String object is not null-terminated. Use length method to get the length of a String object. You can't change the content of a String object; you can only make a new string by combining the existing strings:

```
String s1 = "string1";
String s2 = "string2";
String s3 = s1+ s2;      // s3 is "string1string2"
```

StringBuffer Class:

Unlike String, a StringBuffer object is mutable [6]. StringBuffer is used whenever you want to modify the string, such as inserting a character at some specific position:

```
StringBuffer s = "1357"
s.insert(2, '4');       // the new string is "13457"
```

String Comparison:

Don't use the "= =" operator to compare Strings. A statement which uses the equality operator to compare two Strings will compile but does not give the expected results because the “= =” simply determines whether the two strings’ references are referring to the same object.

```
String s1 = “one”;
String s2 = “one”;
if (s1 == s2)     // gives FALSE
```

The proper way to compare Strings is to use the “equals( )” method.

```
Example: if (s1.equals(s2))
```

To perform a comparison that ignores case differences, we can use “equalIgnoresCase()” method [11].

Character Stream versus Byte Stream:

Java input output package can be divided into 2 categories: character stream and byte stream. Character stream provide ways to read and write characters while byte stream provide ways to read and write binary data such as sound and images. The syntax of Reader/Writer (Character Stream) is similar to those of InputStream/OutputStream (Byte Stream):

```
Reader: int read(char cbuf[], int offset, int length)
InputStream: int read(byte bbuf[], int offset, int length)
```

Even though the byte stream classes provide sufficient functionality to handle most I/O operations, they cannot work directly with Unicode characters [11]. All streams are automatically opened when created. Even though a garbage collector can implicitly close the streams, it is good practice to explicitly close it after using it because you may get some unexpected input. To improve the performance of your code, you should buffer all your input and output whenever possible.

DataInputStram.readLine( ):

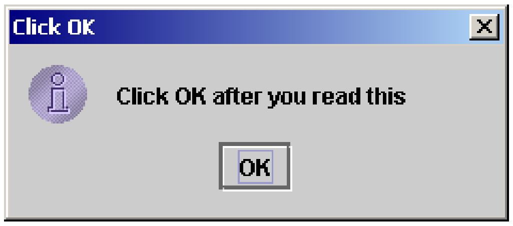
DataInputStream.readLine( ) reads the string terminator, however the string returned by readLine( ) does not include it. Use BufferedRead.readLine( ) instead. In JDK 1.1 and 1.2, if “readLine( )” sees a carriage return, it waits to see if the next character is a linefeed before returning. This is why we often have the situation in which a client or server sits there waiting for the other connection to respond after the client or server has sent a line of message with a carriage return at the end [8].

JOptionPane:

JOptionPane class provides a very convenient, useful, and simple way to create popup dialog boxes, reducing the amount of code required. Consider the following sample code taken from [4]. This 16-line code creates a simple dialog window with caption and a button as shown in Figure 3.

```
public void simpleDialog(JFrame f){
   final JDialog d = new JDialog(f, “Click OK”, true);
   d.setSize(200, 150);
   JLabel l= new JLabel(“Click OK after you read this”, JLabel.CENTER);
   d.getContentPane().setLayout(new BorderLayout());
   d.getContentPane().add(l, BorderLayout.CENTER);
     JButton b = new JButton(“OK”);
     b.addActionListener(new  ActionListener(){
               public void actionPerformed(ActionEvent ev){
```

```
                        d.setVisible(false);
                        d.dispose();
                }
        });
        JPanel p = new Jpanel();
        p.add(b);
        d.getContentPane().add(p, BorderLayout.SOUTH);
        d.setLocationRelativeTo(f);
        d.setVisible(true);
} // (sample code from [4])
```



Click OK after you read this

Figure 3. Dialog Box Generated by the “simpleDialog” Program [4]

With JOptionPane, the above code can be replaced with a single line of code:

```
    JOptionPane.showMessageDialog(null, “Click OK after you read this”,
        “Click OK”, JOptionPane.INFORMATION_MESSAGE);
```

You can also specify which icon your dialog box will display, either a standard or custom icon. Figure 4 shows the four standard icons. The code which generates Figure 3 uses the information icon.

Icons provided by JOptionPane
(Java look and feel shown)

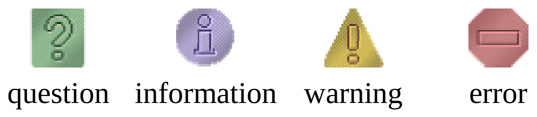


Figure 4. Icons Provided by JOptionPane [1]

Although it is very convenient to use JOptionPane.showInputDialog for getting user input without dealing with the complicated InputStream, there is a catch. The input taken from the created dialog box is always of type “String”. In this case, Integer.parseInt( ) and Double.parseDouble( ) are very handy for converting the data into the format desired by arithmetic calculation [1, 2, 4, 6]. For example:

```
    String num = JOptionPane.showInputDialog("Enter a number:");
    double result = Double.parseDouble(num);
    System.out.println(Math.sqrt(result));
    System.out.println(Math.pow(result,2);
```

JOptionPane can be used within an application or an applet. Further more, it can be placed inside a regular Swing container. Normally, a browser will catch an exception; therefore an applet can’t show an exception. Using JOptionPane is the best way around this issue [1]. For example,

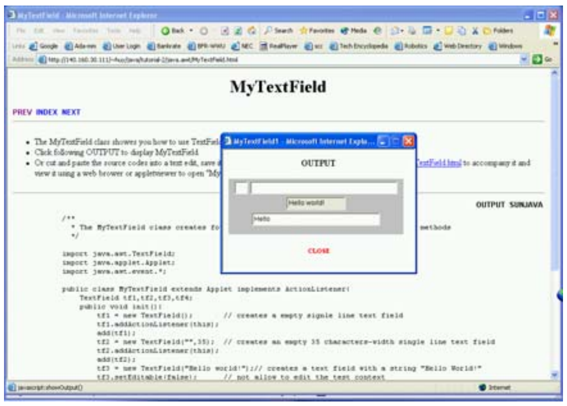
```
try{…}
catch(IOException e){
        JOptionPane.showMessageDialog(null,a,"Exception!!",
        JOptionPane.WARNING_MESSAGE); ...
```

4.2. Sample Tutorial Contents

The Java tutorial is embedded into Web pages. Each Web page contains the name of the Java class, the tutorial example code that shows how to use that particular Java class and five buttons as shown in Figure 5. Three of the five buttons are used for navigation, one for displaying the output of the example code, and one for invoking SunJava definition of that particular class. Students can see the output of the example code immediately after they click on the "Output" button. If they want to see the definition of this class, simply push the "SunJava" button, and the tutorial will bring up the exact page that explains this class. As we have motioned earlier in this paper, an AI search engine will be implemented to search the tutorial knowledgebase for required tutorial pages using keywords entered by students. The following are a few screen shots of Web pages that contain the Java tutorial contents.

MyTextField Applet:

In this example, we show how to use MyTextField class to create four text fields by using different constructor methods. The following screen shot (Figure 5, left) shows the output of the sample program after the "Output" button is clicked. From this example, students not only see the program but also see the output immediately. If they want to change the dimensions of the dialog box, they can easily copy and paste the program and then modify it. School teachers may ask students to use this example combined with other examples shown in this tutorial to create a new program that will deal with a more complicated task.



MyTextField

PREV INDEX NEXT

- The MyTextField class showes you how to use TextFiel
- Click following OUTPUT to display MyTextField
- Or cut and paste the source codes into a text edit, save i TextField.html to accompany it and view it using a web brower or appletviewer to open "My

OUTPUT

Hello world!

Hello

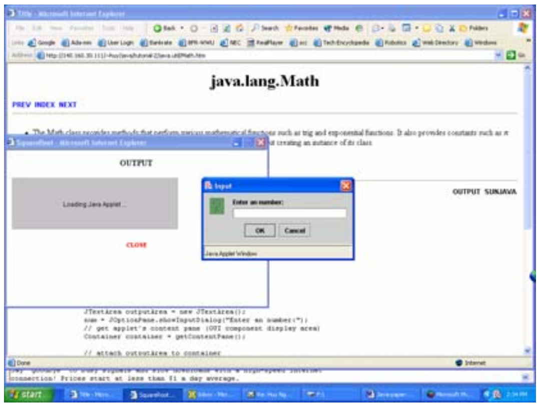
CLOSE

OUTPUT SUNJAVA

```
/**
 * The MyTextField class creates fo                                     methods
 */

import java.awt.TextField;
import java.applet.Applet;
import java.awt.event.*;

public class MyTextField extends Applet implements ActionListener{
    TextField tf1,tf2,tf3,tf4;
    public void init(){
        tf1 = new TextField();        // creates a empty signle line text field
        tf1.addActionListener(this);
        add(tf1);
        tf2 = new TextField("",35);  // creates an empty 35 characters-width single line text field
        tf2.addActionListener(this);
        add(tf2);
        tf3 = new TextField("Hello world!");// creates a text field with a string "Hello World!"
        tf3.setEditable(false);      // not allow to edit the text context
```



java.lang.Math

PREV INDEX NEXT

- The Math class provides methods that perform various mathematical functions such as trig and exponential functions. It also provides constants such as π t creating an instance of its class.

OUTPUT

Loading Java Applet ...

CLOSE

Input

Enter an number:

OK Cancel

Java Applet Window

OUTPUT SUNJAVA

```
JTextArea outputArea = new JTextArea();
num = JOptionPane.showInputDialog("Enter an number:");
// get applet's content pane (GUI component display area)
Container container = getContentPane();

// attach outputArea to container
```

Figure 5. Java Tutorial Screen Shots: TextField Applet (left) and Math (right)

Java.Lang.Math:

The Math class provides methods that perform various mathematical functions such as trigonometry and exponential functions. It also provides constants such as *π* and *e.* Since these constants and methods are static, they can be used without creating an instance of its class. The following example (Figure 5, right) shows how to use Math.sqrt( ), Math.ceil( ), Math.pow( ), and Math.exp( ).

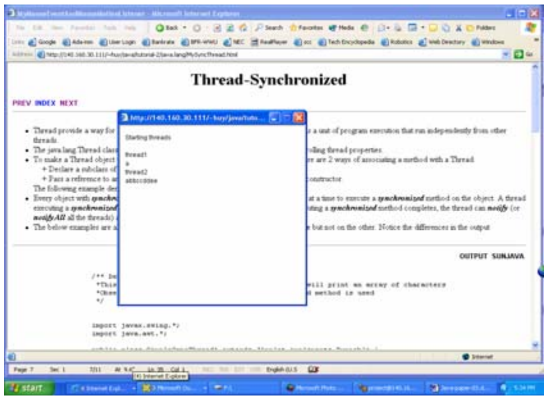
Thread:

To perform and control more than one task at the same time, we use the Thread class. There are two basic methods to associate a method with a Thread:

1) Declare a subclass of Thread that defines a run( ) method [7,18,19].
2) Pass a reference to an object that implements the Runnable interface to a Thread constructor [7]. This is used in a situation in which a class has already inherited from a different class. For example:

```
public class MyApplet extends Applet implements Runnable {...}
```

The example shown in Figure 6 (left) demonstrates both methods to create the threads. Each thread prints an array of characters. Students can see the output for each thread by clicking on the "Output" button.



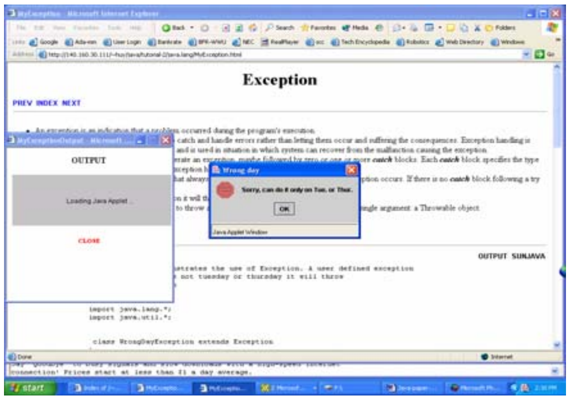


Figure 6. Java Tutorial Screen Shot: Thread (left) and Exception (right)

Every object with synchronized methods is a monitor. The monitor allows one thread at a time to execute a synchronized method on the object. A thread executing a synchronized method may wait if it cannot proceed. When a thread executing a synchronized method completes, the thread can notify (or notifyAll) a waiting thread to get ready to execute.

Exception:

Java provides the Exception class to detect potential problems that can cause a program to malfunction or crash. It is entirely up to the programmer to handle the abnormal event by giving appropriate commands upon the exceptions that are caught. For example, when a Socket is open, the programmer must try to catch an UnknownHostException [10] as shown in the following code segment:

```
try {     socket = new Socket(host, i);
          output+= ("\nServer on port " + i +" of " + host);
          socket.close ();
}catch(UnknownHostException e)
{   break;  }catch(IOException e) {}
```

You can handle the exceptions thrown by the Java API using try and catch as in the example above, or you can create and throw your own exception demonstrated in the tutorial example shown in Figure 6 (right). This example demonstrates the use of Exception defined by a programmer: If today is not Tuesday or Thursday it will throw an exception.

5. The Software

The Java tutorial is written in Java and HTML script. So far we have created a total of 160 Web pages, and each Web page has at least one embedded Java code example and an applet to show the result(s). We closely follow the Sun Java class index and organize the tutorial pages into folders which correspond to the Sun Java Packages (see Figure 7).

The "Main Page" is the front page of this tutorial. Currently it contains only choices to the next page in sequence. In the future, we will have the AI search engine to navigate students' requests. The "Main Page" is connected to a set of HTML files that contains the tutorial contents. It will call related applets according to the current tutorial page contents.

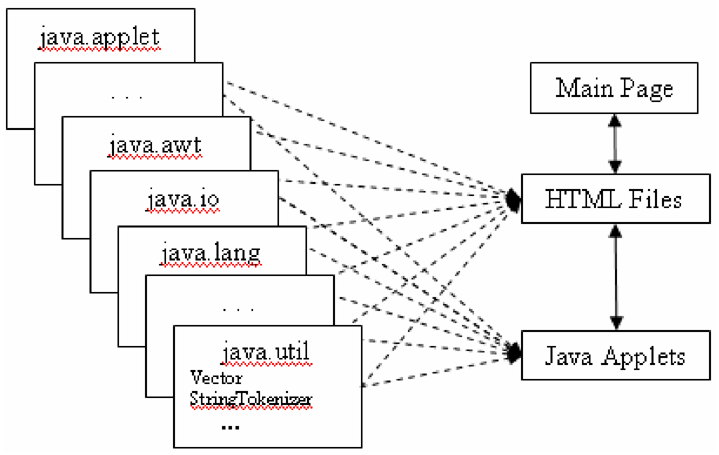


Figure 7. Java Tutorial Software Organization Chart

6. Conclusions and Future Research

We have presented the first version of the example oriented on-line Java tutorial for university students. We believe this Java tutorial will help students learn Java more efficiently while reducing the frustration during the learning process. This tutorial is also a great teaching tool for university professors. Small and simple examples shown in this tutorial can be used as teaching examples or building blocks for homework assignments.

Previously, we have just finished the first draft of the knowledge base of this tutorial, and currently, we are working on the first version of the AI search engine. We will test this tutorial in real class rooms and labs to get feed back from our students and professors. We also plan to finish the implementation of the rest part of the tutorial knowledgebase and connect the AI search engine with the knowledgebase so that students not only can go though the entire tutorial by selecting the navigation buttons, but also can enter key words to see the related tutorial examples, hints, and outputs.

References:

1. Sun Microsystems, *Java Tutorial*, URL: http://java.sun.com/docs/books/tutorial/, November 3, 2003.
2. Harold E. R., *The Java Developer's Resource*, Prentice Hall, 1997, 608 pages, on-line examples URL: http://www.ibiblio.org/javafaq/javatutorial.html.
3. Deitel and Deitel, *Java™, How to program*, 5th edition, Deitel & Associate Inc. and Prentice Hall, New Jersey 2003, 1447 pages.

4. Eckstein R., Loy M., Wood D., *Java Swing*, O'Reilly & Associate Inc., Ca. 1999, 1227 pages.
5. Oaks S., *Java Security*, 2nd Edition, O'Reilly & Associate Inc., Ca 1999, 599 pages.
6. Grand M., Knudsen J., *Java Fundamental classes reference*, O'Reilly & Associate Inc., Ca. 1997, 1089 pages.
7. Oaks S., Wong H., *Java Threads*, 2nd Edition, O'Reilly & Associate Inc., Ca 1999, 319 pages.
8. Harold E. R. *Java I/O*, O'Reilly & Associate Inc., Ca 1999, 568 pages.
9. Harold E. R, *Java Network Programming*, O'Reilly & Associate Inc., Ca 1997, 422 pages.
10. Steelman A., *Murach's beginning* Java2, Mike Murach & Associate Inc., Ca 2002, 712 pages.
11. Schildt H., *Java 2-The Complete Reference*, 5th edition, McGraw-Hill/Osborne, Ca 2002, 1154 pages.
12. Mancoridis S., "Introduction to the Java Programming Language", CS575 Software Design, http://www.cs.drexel.edu/~spiros/teaching/CS575/slides/java.pdf, 2001.
13. Reynolds M., "Java Programming … From the Grounds Up", in Java Tutorial on Webpage: http://www.webdeveloper.com/java/java_programming_grounds_up.html, 2004.
14. Enterprises F., "Introduction to Java Programming", http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/java/, 1996.
15. Gilbert H. , "Learning Java", http://www.yale.edu/pclt/java/default.htm, 1996.
16. Golding B., "Java For Students", http://www.javaforstudents.co.uk/ , 2004.
17. Mukhi V., Kotecha S., and Tripathi S., "Shlurrrpp ..... Java", http://www.vijaymukhi.com/vmis/java.html, 2004.
18. Ahmad F., Panangaden P. "A Little Book on Java, http://crypto.cs.mcgill.ca/~crepeau/CS250/little.pdf, August 30, 2001.
19. Biddle R., Tempero E., "Java Pitfalls for Beginners", http://www.mcs.vuw.ac.nz/~robert/Directory/JavaPitfalls.html, Computer Science Victoria University of Wellington, 1999.