

Java Music Systems: JMSL and JSyn

Nick Didkovsky (didkovn@mail.rockefeller.edu)
Class #1 — Java Overview

Some of these comments taken from Robert Rowe's lecture notes, with permission.

- A compiler changes the code of a high-level language (such as C or Pascal) into low-level machine codes for a particular platform. Therefore a single C program can be compiled to run on Macintosh, IBM-compatible, or UNIX-based computers. Programmers must use great care, however, not to let OS-specific commands (such as system calls) proliferate in their code because this will have to be rewritten each time the program is moved to a different operating system.
- Java is a language specification that was developed by Sun Microsystems for use in distributed systems. It is designed to be simple, object-oriented, small, portable, and secure. Originally Java was developed to run on embedded systems — small computers that run in devices devoted to other things, like your refrigerator. However it has scaled up gracefully to be a very powerful and elegant programming language that runs efficiently on a number of platforms.
- Java is designed to be simpler than languages such as C, C++, or Lisp. It is, however, a full blown programming language. Inherent in Java's design is the goal of minimizing the kinds of problems that slow down the programmer's productivity. Typical programming problems that are exceedingly time consuming to track down, such as memory leaks, lost pointers, and machine-level crashes that occur when developing in other languages such as C are eliminated by the fundamental design of Java.
- Interpreted languages (such as Lisp) are changed into machine instructions as they are run. This eliminates the compilation step and encourages a more interactive form of programming. Java is somewhat in between compiled and interpreted languages, in that Java programs are compiled into "bytecode", that then is interpreted on the host platform. This means that any computers running Java programs must have implemented a virtual machine that is able to take Java bytecodes and interpret them as low-level instructions on the host computer. Because of this approach Java programs will run with no modification on any computer that is Java-ready. In fact, because Java is now becoming so widespread, a number of compilers are available that translate Java directly into machine code, rather than bytecode for the virtual machine. While this makes the resulting applications more efficient, they are no longer portable.
- Object orientation is a concept in computer science that refers to the encapsulation of data and methods. An object contains within itself the data it needs to function and the methods that can operate on that data. Both methods and data can be either *private*, in which case only the object itself can use them, or *public*, meaning that other objects can access the data or execute the methods. Inheritance allows some classes to inherit data and methods from other classes. Thus programmers can use most of a generic object and specialize it in a subclass adding methods and data unique to their project.
- An object is an instantiated class, that is, an object is one concrete instance of the generic class definition. One class can generate many objects in a single application. Each of these objects maintains its own *state*, that is, its own set of values for the variables defined in the class.
- The traditional starting point for learning programming languages (at least since the C programming language gained popularity) is to make a program that prints out "Hello, World" on the computer screen. The Java program below declares a *class* that has one main function, in which is an instruction that will cause "Hello, World" to be printed. As mentioned above, a class is a combination of methods and data. This encourages good program design, but also (importantly for Java) makes it easier to guarantee the security of the application.
- The *public* modifier of the class and main function indicates that they can be accessed from any other class in the program. If the class and that function had been identified as private members they could only be accessed by objects of the class *TrivialApplication*. The statement

System.out.println calls a method "println()" that belongs to the class variable "out" of the class "System". Concatenating names like this is how the language identifies data and methods nested within other language constructs. The code below is a complete program — it can be compiled by a Java compiler and executed on any computer running a Java interpreter.

```
public class TrivialApplication {  
  
    public static void main(String args[]) {  
        System.out.println( "Hello World!" );  
    }  
  
}
```

- A Java *applet* is a program that can be added to a Web page and run by a web browser that is Java-enabled. A Java *application* is a standalone program that can be run in a Java virtual machine. Such applications do not require an operating system and so are designed for the network computer approach to multiuser systems.

- The HTML code to implement a web page containing the user's applet is shown below:

```
<HTML>  
<HEAD>  
<TITLE>My Java HTML page</TITLE>  
</HEAD>  
<BODY>  
<applet code="mypackage.Calculate.class"  
        codebase ="classes"  
        width=600  
        height=400 >  
You need a Java-enabled browser to view this applet.  
</applet>  
</BODY>  
</HTML>
```

Notice the polite message between the <applet> and </applet> tags. Browsers that do **not** understand Java will ignore the applet tags and print the "You need a..." message between them. Browsers that **do** understand Java are designed to ignore any text between the tags.

- A number of Java IDE's (Integrated Development Environments) are on the market, including Eclipse, IDEA by IntelliJ, and NetBeans.

Eclipse and IDEA are very powerful packages which offer such modern tools as refactoring and continuous error checking. "Refactoring" includes such powerful operations as method extraction (highlight a chunk of inner code and extract it into its own method), smart renaming of variables, package names, and class names: as opposed to a global text search and replace, renaming an identifier in these packages knows the Java context of the identifier and only renames those which match. Refactoring is a hot topic in object oriented programming these days, and having some automated refactoring tools available in an IDE is an extraordinarily powerful resource.

Development Environment recommendations:

Mac OS X and Windows, use Eclipse. It is freely downloadable from <http://www.eclipse.org>

Java's AWT Jump Start

Java's Abstract Windowing Toolkit or "awt", is a Java package that provides the programmer with GUI components, windows, layout, and event handling. By event handling we mean software response to user actions, like clicking a button or moving a slider.

Java went through a major design change in its event handling when it jumped from version 1.0.2 to 1.1x. It's earlier version was a bit clunky, but unfortunately proliferated on the Web as developers and companies rushed to launch Java applets on the Internet. In this course, we will stick to the 1.1 event model.

Layouts

A Layout is a Java class that handles how components are arranged in a Container such as a Panel or a Frame. Simple layouts include:

FlowLayout, which adds components left to right, wrapping to the next row

BorderLayout, which divides the container into North, South, East, West, and Center

GridLayout which adds components in evenly spaced rows and columns. You can use ad hoc Panels liberally within a layout, to add components in a subarrangement (for example, a scrollbar and a label to display the scrollbar's value). Components like buttons, labels, etc are add()'ed to layouts.

Use the `setLayout()` method on a container to govern the layout style. For example:

```
setLayout(new GridLayout(1,2)); // one row, two columns
Panel p = new Panel();
p.setLayout(new BorderLayout());
p.add(BorderLayout.North, new Label("Ollie"));
p.add(BorderLayout.South, new Label("Parrot Jungle, 5 miles ahead"));
p.add(new Label("egocentric")); // leave out a location and
// BorderLayout assumes "Center"

add(p); // add this panel to the Layout of this class
```

Label

A `java.awt.Label` is simply a rectangular component that displays some text (String). Its text can be changed with its `setText(String s)` method. You can create a Label and add it to a layout like so:

```
Label myLabel;
...
myLabel = new Label("This will be displayed");
add(myLabel);
...
myLabel.setText("This is my new message");
```

Note that a Label's text can be updated at runtime with `setText()`. You can change it anytime you like - it is not permanent. So it can serve as a way to make messages appear in your GUI or show progress of some process.

Button

A `java.awt.Button` is a component that can be clicked and fires a response. When clicked, it posts an `ActionEvent` to any `ActionListener` who cares to listen. The event is caught and handled by the method:

```
public void actionPerformed(ActionEvent e);
```

...which must be defined by any class which has registered itself as an `ActionListener` and adds itself to the button's list of listeners.

Below is a typical implementation of `actionPerformed()` which first checks which component caused the event, then calls the appropriate user-defined method:

```
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == myButton) handleMyButton;
    if (source == otherButton) handleOtherButton;
}
```

Below is a complete working example of an Applet that contains a button and changes its background color when the button is clicked.

```

package com.didkovsky.javamusic;
import java.awt.*;
import java.awt.event.*;

/** Switch between two background colors every time a button is pressed */

public class ColorChangingApplet extends java.applet.Applet implements ActionListener {

    Button myButton;
    boolean coolColor;

    public void init() {
        myButton = new Button("Color Change");
        add(myButton);
        coolColor = false;
        myButton.addActionListener(this);
    }

    void handleColorChange() {
        coolColor = !coolColor;          // switch state from true to false
                                        // or from false to true

        if (coolColor) {
            setBackground(Color.pink);
        } else {
            setBackground(Color.white);
        }
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == myButton) handleColorChange() ;
    }
}

```

Below is the HTML document that runs it (note the dot reference to the package `com.didkovsky.javamusic` and the specification of where the classes are located):

```

<HTML>
<HEAD>
<TITLE>ColorChangingApplet</TITLE>
</HEAD>
<BODY>
<applet code="com.didkovsky.javamusic.ColorChangingApplet.class"
        codebase ="classes"
        width=320
        height=200>
You need a Java-enabled browser to view this applet.
</applet>
</BODY>
</HTML>

```

Homework Assignment:

Create an Applet with a background colored anything but white, that includes the following AWT components and interactions:

- A Label that says "Hello I am <your names>'s first Applet"
- A TextArea that initially displays the text "This is a TextArea."
- A Button which, when clicked, clears the TextArea and prints the current time in milliseconds (see Java's `System` class documentation).
- A TextField into which the user can type a String.
- A Button which, when clicked, pulls the String from the above TextField, converts it to an integer, clears the TextArea, and prints both the integer and the integer multiplied by 2 in the TextArea. An algorithm for converting a String to an integer follows:

```
int value = (new Integer(someString)).intValue();
```

Of course someString has to be a String consisting of digits. Don't expect to convert the String "Hello" to an int with this algorithm!!!

Make this applet part of a package that includes your name (for example, johnsmith.javamusic), and post it on your website. Email me the URL of the HTML page that runs it (didkovn@mail.rockefeller.edu). I suggest your website have a directory called `JavaMusicSystems` which contains the HTML work for this class, which would include HTML pages which run your Java classes and other media like soundfiles, etc. It should contain a "classes" directory into which all your .class files go.

EXTRA CREDIT:

Create an applet containing a TextField into which you can type a base frequency, another TextField into which you can type the number of intervals in an octave, and another that takes the pitch you want to calculate. Add a button which, when clicked, prints the frequency of this step in a label.

For example, for 12 tone equal tempered music (Western European), type in 27.50 as the base frequency (this is A0, the lowest note on a piano, see Dodge/Jerse p. 37), 12 as the number of intervals per octave, and 2 to calculate the frequency of B0 (that is 2 steps above the base frequency). Your applet should calculate and display 30.87 Hz as the resulting frequency

The formula for calculating frequency from a base pitch and an interval number is:

$$\text{Pitch} = \text{basefreq} * 2^{i/n}$$

Where i = step and n = intervals/octave

Example: basefreq of A0 is 27.50

To calculate A#, which is one semitone higher, the formula would be:

$$\text{Pitch} = 27.50 * 2^{1/12} = 29.135$$