Today's Agenda

> To look at your homework & go over coding questions

> To look at a simple Java program

> To use javac to compile a program and java to run a program

> To use the classpath and library path options

> To show how to use Processing libraries directly from Java programs

> To write a bash script to simplify compilation

> To look at the Eclipse IDE

> To introduce the controlP5 GUI library for Processing

> To introduce the Swing GUI library for Java

Simple Java Program

```
/* Filename = Simple.java */

public class Simple {
  public static void main(String[] args) {
    System.out.println("Hello 201B!");
  }
}

> javac Simple.java

compiles the file Simple.java into java bytecode stored in the file Simple.class

> java Simple

runs the Simple program which prints out "Hello 201B!" to the console (to stdout)
```

Java Processing/PApplet wrapper

Why would we want to use Processing libraries from within Java?

> To use an editor we are more familiar with (although the Processing IDE lets you use an external editor, you still have to run it from Processing)

> To take advantage of the latest version of Java. The Processing compiler is still using Java 1.4 syntax, about 4 or 5 years old, which doesn't handle Generics or some of the nice syntax changes introduced in Java 1.5.

> To be able to develop with an IDE with more functionality, which can be very useful for managing and debugging larger projects.

> To be able to create regular classes. In Processing all classes are inner classes of the main sketch class.

Java Processing/PApplet wrapper

```
/* File MyProcessingSketch.java */

import processing.core.*; //import the main Processing libraries

public class MyProcessingSketch {
    public static void main(String args[]) {
     PApplet.main(args); //call the main Processing method to initialize the Processing
    environment
    }

    public void setup() { //overrides the setup method in PApplet
     size(200,200);
     //etc...
    }

    public void draw() { //overrides the draw method in PApplet
     //draw stuff...
    }
}
```

Java Processing/PApplet wrapper

To compile & run you will need to point your classpath to Processing's core.jar

```
> javac -cp locationOfLibs/core.jar MyProcessingSketch.java
//creates a file called MyProcessingSketch.class

> java -cp locationOfLibs/core.jar MyProcessingSketch
//runs the MyProcessingSketch program
```

To include other Processing libraries, such as controlP5 or minim, you simply add them to the classpath as well; and make sure to import them into your Java code.

```
> java -cp lib/core.jar:lib/minim.jar:lib/controlP5.jar MySketch2.java
> javac -cp lib/core.jar:lib/minim.jar:lib/controlP5.jar MySketch2
```

Some libraries (e.g. OpenGL) also require *native* runtime libraries, which are specified using the *-Djava.library.path* flag.

```
> java -cp lib/core.jar:lib/opengl.jar:lib/jogl.jar:lib/gluegen-rt.jar MySketch3.jar
> java -cp lib/core.jar:lib/opengl.jar:lib/jogl.jar:lib/gluegen-rt.jar
    -Djava.library.path=lib/opengl-natives MySketch3.jar
```

A simple bash helper script

```
//special first line that indicates that this is a bash script
#!/bin/bash

//lines that begin with the # are comments ignored my the script
#basic usage: ./runme.sh theSketch
#fullscreen : ./runme.sh theSketch --present

//variables are declared using the = operator
### set directory locations
P5_CP=lib/core.jar:lib/opengl.jar:lib/jogl.jar:lib/gluegen-rt.jar
P5_NATIVES=-Djava.library.path=lib/opengl-natives

//any shell command can be called from within the script
//variables are indicated by prefixing a $
//a $ followed by a number indicates an argument passed in from stdin
### compile the files
javac -cp ./$1:$P5_CP $1/*.java

### run the files ($2 can be set to '--present' for fullscreen mode, otherwise it should
    not be defined)
java -cp ./$1:$P5_CP $P5_NATIVES $1 $2 $1
```

A simple bash helper script

By convention, bash shell scripts end with .sh

To run your script, you must first make it executable using the **chmod** command:

```
> chmod +x myscript.sh
```

And then you can run your new unix command **myscript.sh** from the command line:

```
> myscript.sh MyProcessingSketch --present
```

Basic vim usage

```
Vim is the standard editor in bash. You do not have to use it if you don't want to!

gvim/MacVim is a nice GUI version of vi/vim. All of the below instructions apply to it as
    well.

to open a new java file:
> vi JavaFile.java

this will create an empty file and initially place you in "command mode."

> press "i" (for insert)
this will place you in "insert mode."

> type in your code...

> press "ESC" (to escape back to command mode)

> type ":wq" (":" = instruction, "w" = write, "q" = quit)
this will save your file and exit vi.

> or type ":q!" if you want to exit without saving your file.
```

Basic vim usage

to re-open your file:

> vi JavaFile.java

this will re-open your file and initially place you in "command mode."

> type "h" (left), "l" (right), "j" (down), and "k" (up) to navigate around your text.

> type "i" to enter insert mode to add more text

> in command mode ("ESC"), type "x" to delete text

> to save and quit, go back to command mode and then... type ":wq"

There are TONS of other commands and tweaks that you can use to edit text easily. Some of them are quite cryptic, so I'll let you figure them out on your own if you want to use vi (or if there is enough interest I can talk about it more in a future class).

Some of the tweaks let you use conventional keyboard commands to navigate, copy & paste, save, quit, etc. But your shell may not support them, so good to know the basic ones as well.

Eclipse

I'll walk you through this...

GUIs

Graphical User Interface libraries allow you to add a set of controls to your programs.

Java -> AWT widgets, Swing library, JavaFX, SWT

Processing -> built in, controlP5, other libraries (which I haven't tried)

Cross platform -> Qt / Qt-Jambi

Game input libraries (for other devices) -> JInput, SDL bindings, JMonkeyEngine, etc

we'll look just at mouse and keyboard inputs for now.

controlP5 GUI library for Processing

```
//slider example

import controlP5.*;

ControlP5 controlP5;
int sliderValue = 100;

void setup() {
  size(400,200);
  controlP5 = new ControlP5(this);
  controlP5.addSlider("sliderValue", /*range*/ 0,255,128, /*location*/ 120,120,100,30);
}

void draw() {
  background(0);
  fill(sliderValue);
  rect(0,0,width,100);
}
```

Swing GUI Framework for Java

```
import javax.swing.*;
import java.awt.event.*;
public class Buttons extends JPanel implements ActionListener {
  public Buttons() {
    JButton b1 = new JButton("Press me!");
    b1.addActionListener(this);
    add(b1);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("you pressed the button!");
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        JFrame frame = new JFrame("Buttons!");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new Buttons());
        frame.pack();
        frame.setVisible(true);
      }
    });
  }
}
```