Computer Science 252
Problem Solving with Java
The College of Saint Rose
Fall 2015

Topic Notes: Programming Without Objectdraw

A common criticism among students who take classes that make use of pedagogical tools such as Objectdraw is that "we're not learning real Java". As we wrap up, we will see that you have been doing real Java all along, and even those things that used Objectdraw are very similar to the things you'd need to do to develop similar programs with only Java's primitives and its standard libraries.

Applications

We have emphasized Java applets – graphical programs – over Java applications – keyboard/console programs – for most of this semester. However, you have seen a good number of Java applications between earlier courses and some of our examples this semester. Java applications are those that we execute using a `main` method, and those with which we interact using the keyboard, screen, and files, using things like `Scanner`s, `println`, `PrintWriter`s, and others.

One complication you might have noticed arises from the fact that the `main` method in a Java application must be declared with this method signature:

```
public static void main(String args[])
```

The qualifier `static` here means that your `main` method here can only access variables which are with local or are also declared as `static` and call only methods that have the `static` qualifier. The reason for this is that our regular ("non-`static`") variables – the instance variables – are only created when an instance of the class is constructed with `new`. So it wouldn't make much sense for a `main` method to access instance variables, as none have been created!

However, a class with a `main` method might still have instance variables and non-`static` methods, but these are only accessible once an instance of the object has been created with `new`.

See Example: Matrix2D

Further, the `static` keyword can be applied to the kinds of variables we have been calling instance variables, but when we do so, they instead become *class variables*.

See Example: StaticStuff

See Example: StaticCount

Java Applets

We have already seen some examples of GUI-based Java programs, or Java Applets, which are not Objectdraw programs. When we have a class that "`extends WindowController`" we know that a window will be created that contains a `DrawingCanvas` that is available to us in the methods of the `WindowController` class as `canvas`. The program starts by excuting the `begin` method if one is specifed.

We later saw how we could add Java Swing GUI components. Finally in the ATM concurrency examples, we saw that some programs don't have a canvas at all. The main differences to point out here:

Our class "`extends JApplet`" instead of `WindowController`. It starts by executing an `init` method instead of a `begin` method. There are no "on mouse" methods here, as those are also specific to the `WindowController` class. Here, we interact with our program through the `actionPerformed` method, which was added as an `ActionListener` to the button.

In addition to `ActionListener`, we have seen `KeyListener` and `ChangeListener`.

We can also add listeners for mouse events with the `MouseListner` and `MouseMotionListener` interfaces. See BDM Section D.2. for more.

Standard Java Graphics

Obviously, there are Java constructs to support drawing of graphical objects in a `JApplet` program. After all, an Objectdraw program is really a `JApplet` with a lot of "training wheels" attached to shield you from some of the messy details. Nowhere is this sheilding more valuable than in how the graphics shapes are drawn on the canvas.

We can draw all of our standard shapes and much more, but as you can see in BDM Appendix D.4, there is significantly more code to write to make things happen.

See Example: StdGraphicsDemo

Java's `Thread` class

Finally, the ATM examples also show that the `ActiveObject` class we used to handle threads is very similar to Java's standard `Thread` class. The main differences:

- The class should extend `Thread` instead of `ActiveObject`.
- Instead of `pause` to have the thread wait, we call `sleep`. However, since `sleep` can throw an `InterruptedException`, we need a little extra code to catch that.