Lecture 8
Java is Pointerless?

(C) Princeton University
Hints on Programming

• Don’t wait until the last day to start.
• Compile on your local machine. Use elmo, zoe, etc. only if you must -- they are busy machines.
• Develop incrementally. Start with a working program. Make small changes. Check that the program still works.
Examples of incremental development

• Come prepared, having read the assignment, having read the textbook, having listened in lecture, having prepared the outline below.
• Copy working version of Integra.java from orf201.
• Add a method func() to evaluate a function. For starters, let the function be \( f(x) = x^2 \).
• Call func() from integrate() with a few choice values and print out the values computed to make sure func() works properly.
• Write code in paint() to draw the function defined by func().
• Write a method rect() to compute the area using the rectangle rule. Put a call to this method in integrate(). Use System.out.println to print out the answer. Run with lowx = 0, high x = 1, n = 1. Compare this answer against a hand calculation. Try some other simple cases.
• Make a new method called trap(). Copy the code in rect() to trap(). Modify the code just copied so that it implements the trapezoidal rule. Test as above.
• Add code to paint() to illustrate the approximating rectangles. Test with small values of n.
• Add code to paint() to illustrate the approximating trapezoids. Test with small values of n.
• Change function implemented in func() to the complicated exponential function given in the assignment.
Memory Layout: Variables and Arrays

Declare:
```java
int n;
double[] x;
```

Initialize Variable:
```java
n = 12;
```

Initialize Array:
```java
x = new double[n];
```

Initialize Array Elements:
```java
for (j=0; j<n; j++) {
    x[j] = j*j;
}
```

Look mom, it’s a pointer!
Memory Layout: Class Variables

Declare:

```java
Zip z;
```

Instantiate Class Variable:

```java
z = new Zip();
```

Initialize Components:

```java
z.zip = 90210;
z.lat = 34.09;
z.lon = 118.41;
```
Memory Layout: Arrays of Class Variables

Declare:

```csharp
    Zip[] zlist;
```

Initialize Array Variable:

```csharp
    zlist = new Zip[100];
```

Initialize Array Elements:

```csharp
    for (j=0; j<n; j++) {
        zlist[j] = new Zip();
    }
```

Initialize Components:

```csharp
    for (j=0; j<100; j++) {
        zlist[j].zip = Console.in.ReadInt();
        zlist[j].lat = Console.in.ReadDouble();
        zlist[j].lon = Console.in.ReadDouble();
    }
```
Java Has Pointers

This makes an instance of an integer.

These only make pointers to objects of the type mentioned.

```java
int n;
double[] x;
Zip z;
Zip[] zlist;
```
What does \texttt{new} do?

\texttt{new} makes an \textit{instance} and gives a \textit{pointer} to it.

\begin{verbatim}
x = new double[12];
z = new Zip();
zlist = new Zip[10];
\end{verbatim}

\texttt{x} is now an array of doubles and using \texttt{x[j]} is now allowed.

\texttt{z} is now an instance of \texttt{Zip}. It is now possible to put things into the individual fields.

\begin{verbatim}
for (int j=0; j<10; j++) {
    zlist[j] = new Zip();
}
\end{verbatim}

\texttt{zlist} is now an array of \texttt{Zips}. But each of the elements is just a pointer that still needs to be told where to point:

Now, finally, the individual fields of \texttt{zlist[j]} can be used.
ZipFinder2

Found a better zipcode database:  zlla0.dat

Changed binsearch to return the negative of a nearby zipcode if the requested zipcode is not found.

At the end of binsearch, replaced return -1 with:

```java
if (high < 1) { high = 1; } return -high;
```

Changed the while loop in main() to print and plot even in the "not found" case:

```java
if (j >= 0) {
    myzip = zips[j];
    Console.out.println("Found it in "+myzip.lat+" "+myzip.lon);
} else {
    Console.out.println("Not found");
    myzip = zips[-j];
    Console.out.println("How about "+myzip.zip+": "+myzip.lat+" "+myzip.lon+"?");
}
```

Improved graphics somewhat.  See source for details.
ZipsFinder

To illustrate performance difference between `brutesearch()` and `binsearch()`, created ZipsFinder that looks up 1000 random zipcodes.

Made an array of zips to plot (as class globals):

```java
static Zip[] myzip = new Zip[1000];
```

In `main`, added a loop to instantiate the 1000 array elements:

```java
for (i=0; i<1000; i++) {
    myzip[i] = new Zip();
}
```

Changed `paint()` to show all the 1000 sought zip locations in yellow:

```java
for (i=0; i<1000; i++) {
    GL.color(Color.yellow);
    GL.pnt2(myzip[i].x, myzip[i].y);
}
```