Mr G's Java Jive

#10: Working with Real Numbers

So far all the calculations we've done in our programs have dealt with **integer** values. Now it's time to start working with **real numbers**. That's what this handout is all about.

Real, Floating Point, Numbers

As you probably remember from your many math classes, there are generally two types of numbers: **integer** and **real**. **Real** numbers are the set of all numbers To most people it means numbers with a decimal point, but **integers** are an important **subset** of the set of **real** numbers. It's important to understand that **integers** are part of the set of **reals** because this means that wherever a **real** is requested, you can give an **integer**.

However, the reverse is not true. Because **integers** are a **subset** of **reals**, it's more restrictive. **Integers** are the reals that have **nothing** after the decimal point, and actually **don't need** the decimal point at all. If you give a **real** where an integer is requested, at best you'll have your number **truncated** to just the integer portion. At worst, you'll get a **type mismatch error** (remember those).

Another name for a **real** number is a **floating point** number. That's because based on how many positions you have available to you on your pocket calculator, the decimal point can **float** around to wherever works best in order to show the significant digits to the **left** of it. This term **floating point** number is important, because the most basic **real** type that Java supports is called **float**.

Double Your Pleasure, Double Your Precision

While **float** is the most basic **real** type that Java supports, it's not the most useful. Most of the mathematical methods we'll want to eventually use from Java's many **helper classes** (especially the **Math** class) assume that you're using the **double** type, which has more precision than a plain old **float**. So with that being said, any time we need to work with real numbers, we'll use the **double** type. It'll save us aggravation in the long run.

The Geo1 Program

In this program, you're going to ask the user for the **dimensions of a rectangle**. Then once you have those dimensions, you're going to tell the user its **area** and **perimeter**. Start with the following skeleton:

```
//Geo1
//a program to figure out the area and perimeter of a rectangle
//8.10.06 Clark Kent

public class Geo1
{//start class
   public static void main()
   {//start main
      //variables

      //get input

      //do calculations

      //give output

   }//end main
}//end class
```

Some Initial Input

Now that we have our skeleton, let's try to let the user know what's going on. Then we'll compile it and see if it runs.

```
public static void main()
   {//start main
      //variables
        String name;

      //get input
        System.out.print("Hello. What's your name? ");
        name=gatling.getLine();
        System.out.println("\nWelcome to the Geo1 program, "+name+"!");
        System.out.println("Its job is to figure out the area and perimeter");
        System.out.println("of a rectangle that you give it.");

      //do calculations

      //give output

   }//end main
```

By the way, did you notice that **\n** at the beginning of our first **println** statement? That tells Java to start another whole new line before printing the rest of output to the screen. In this case we did this so that there's a nice break between the program asking for the user's name and telling the user what it's going to do.

Getting Our Double Values

So far, so good. Of course, the program doesn't really do anything yet, but we'll take care of that right now by adding the following code.

```
//variables
  String name;
  double height, width, area, perimeter;

//get input
  System.out.print("Hello. What's your name? ");
  name=gatling.getLine();
  System.out.println("\nWelcome to the Geo1 program, "+name+"!");
  System.out.println("Its job is to figure out the area and perimeter ");
  System.out.print("of a rectangle that you give it.");
  System.out.print("\nHow wide is your rectangle? ");
  width=gatling.getDouble();
  System.out.print("\nThanks. Now how tall is your rectangle? ");
  height=gatling.getDouble();
```

This should compile with no problem. It won't do anything new, since we haven't had it run any calculations, but it should compile with no problems. By the way, did you notice those **\n**s at the beginning of the lines where we asked for the dimensions of the rectangle? Why do you think we did that?

We'll add the calculations and the final output on the next page.

Doing the Math

The code below shows the changes you have to make in order to get Geo1 to actually do the math and then output the results:

```
//do calculations
  area=height*width;
  perimeter=2*(height+width);

//give output
  System.out.print("\nYour rectangle has an area of "+area+", ");
  System.out.println("and a perimeter of "+perimeter+".");
  System.out.println("\nSee you later, "+name+"!");
```

One very important thing you have to pay attention to here is the fact that unlike in algebra multiplication is not implied. What I mean by this is that for the **perimeter** calculations, I had to specifically write out **2*(height+width)**. Simply placing the **2** next to the parentheses for **2(height+width)** wouldn't work, and would give you an error message.

OK, time to compile and run! Handout #11 will take a look at some **number formatting** issues.

This page intentionally left **almost** blank