Mr G's Java Jive

#13: Choices and Local Methods

So far all of the programs we've written have just done one thing, and have just had one method – **main** – in them. Now we're going to get a little more sophisticated and write one that gives you a choice between two things.

The RootChoices Program

For the purposes of showing you how to do this, **RootChoices** will be a relatively simple program. Then you can use what you've learned here to give your users their own set of choices in other programs. We will, however, use this as a chance to play with some of the methods in the **Math** class. Let's start off with this basic skeleton:

```
//Choices
//a program to let the user choose between three kinds of numeric roots
//8.11.06 Clark Kent

import java.text.*;  //needed to do number formats

public class RootChoices
{//start class
   //======= parking space 1

   //======= parking space 2

   //======= parking space 3

   //======= main parking space
   public static void main()
   {//start main
      //variables

      //get input

      //make the choice

      //give output

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

This looks pretty similar to things we've done before, except that now, for the first time in a **standalone class** we've included the same kind of **parking spaces** that we've put in our **helper class** of **Mine**. You should also notice that **main** is in the very last parking space, and that it has a slightly different layout too. Finally you should notice that we've imported **java.text.*** at the top of the program, even before we've started the class. This lets us create the **number formats** we'll need for this.

The Main Method

As usual, we'll start off with just enough for us to make sure that the program actually runs. So make the changes found on the next page:

```
{//start main
   //variables
     String name;
     int choice;

   //get input
     System.out.print("\nHello. What's your name? ");
     name=gatling.getLine();
     System.out.print("\nWelcome, "+name+". ");
     System.out.println("This program gives you four choices. They are:");
     System.out.println("\t[1] The Square Root of a Number");
     System.out.println("\t[2] The Cube Root of a Number");
     System.out.println("\t[3] Any other Root of a Number");
     System.out.println("\t[4] Quitting and Getting Out of Here!");


   //make the choice

   //give output
     System.out.println("\nThanks for stopping by. Come back again!");
}//end main
```

OK, compile and run. It shouldn't do anything fancier than just let you know that it's there and it works. We'll take care of the fancy stuff next.

Our Three Choices: square, cube, and other

Each valid choice the user makes is going to either send them to one of three **local methods** or right out of the program. Our three local methods will be **square**, **cube**, and **other**, and we'll set up simple **stub** versions of them right now just so that we have something we can test. Check out the changes below:

```
public class RootChoices
{//start class
   //======= parking space 1
     public static void square()
     {//start square
         System.out.println("This is square.");
     }//end square
   //======= parking space 2
     public static void cube()
     {//start cube
        System.out.println("This is cube.");
     }//end cube
   //======= parking space 3
     public static void other()
     {//start other
        System.out.println("This is other.");
     }//end other
   //======= main parking space
     public static void main()
```

These three methods are declared as **void** because their only job is to display information on the screen and not **return** anything. But they can't even really do that yet, since we haven't <u>used</u> them. We'll take care of that with a few changes on the next page.

```
//make the choice
  System.out.print("\nEnter the number of the choice you want: ");
  choice=gatling.getInt();
  if (choice==1)
    {square();}
  if (choice==2)
    {cube();}
  if (choice==3)
    {other();}
  if (choice==4)
    {System.out.println("\nI'm sorry you didn't like any of our choices.");}
```

Now compile and run it. Impressive isn't it? Well, OK, it still doesn't do much, but you can see that it has potential. There is one little problem we need to deal with though – **idiots**. You know just as sure as I do that some turkey will come along and enter **17** as one of the choices. It won't ruin the program, but we should let the user know that it's unacceptable. Check out the following additions:

```
//give output
  if (choice<1 || choice>4)
  {//start if
     System.out.println("\nYou are an idiot who can't follow directions!");
     System.out.println("Feel free to come back when you get a clue.");
  }//end if
else
  {System.out.println("\nThanks for stopping by. Come back again!");}
```

You're probably wondering about the **||** symbols in the **if** statement. Those two symbols (found at **shift-blackslash**) are how you say **or** in Java, and an **or** is true if either part of the statement is true. So this statement is saying if **choice is less than 1 or greater than 4**, then do what comes in the set of curly brackets.

Our **else** statement is simply the old output to the program. We've put it in the **else** so that the users only gets the polite response if they're not total idiots.

Compile and run it with both good and bad data, and then we'll start working on our first local method.

The square Method

Take a look at the code below and make those changes to your square method:

```
public static void square()
{//start square
   System.out.println("This is square.");
   //variables
     double num, root;
     String nstring, rstring;

   //number formats
     DecimalFormat d3o = new DecimalFormat("0.###");

   //get info
     System.out.print("\tEnter the number you want the square root of: ");
     num=gatling.getDouble();

   //calculations and output
     root=Math.sqrt(num);
     nstring=d3o.format(num);
     rstring=d3o.format(root);

     System.out.println("The square root of "+nstring+" is "+rstring+".");
}//end square
```

Now let's look at a few things that might be worth noting in that code:

1. The Strings **nstring** and **rstring**. We'll be using those to get the **formatted** forms of both the initial number and the square root.
2. **DecimalFormat d3o**… We'll need to create the format in each method we're using it in, so be prepared to do this two more times. In addition, notice that we're defined this one **directly**, and as one with **optional** decimal places. There's no point in having **2.5** show up as **2.500**.
3. **nstring=d3o.format(num)**. This puts our initial number into a formatted text string based on the **d3o** number format.

Now compile it and run it!

The cube Method

This is going to be really simple. It's pretty much the same as the **square** method, except that this time you'll be using **Math.cbrt** instead of **Math.sqrt**. You'll also want to change the word **square** to **cube** (this happens **twice**). In fact, this is so easy that I'll let you do this yourselves, and save my energy for explaining the next method.

The other Method

Sometimes a simple square or cube root isn't really what you want. Sometimes you need a 4^{th} root. Or, as I mentioned earlier in the case of piano tuning, you might want a 12^{th} root. This is easily done using the Math.pow method. Take a look at the example below to see how we'll use this in our **other** method. Just delete the one line you already have there, and copy this whole thing as is:

```
public static void other()
{//start other
   //variables
     double num, power, root;
     String nstring, pstring, rstring;

   //number formats
     DecimalFormat d3o = new DecimalFormat("0.###");
     DecimalFormat frac= new DecimalFormat("1/0");

   //get info
     System.out.print("\tEnter the number you want the root of: ");
     num=gatling.getDouble();
     System.out.print("\tNow enter the whole number root you want of it: ");
     power=gatling.getInt();

   //calculations and output
     root=Math.pow(num,(1/power));
     nstring=d3o.format(num);
     rstring=d3o.format(root);
     pstring=frac.format(power);

     System.out.println(nstring+" to the "+pstring+" power is "+rstring+".");
}//end other
```

As with **cube**, there are some things we have to look at there that are worth noting.

1. We've added a new double variable of **power** and a String object of **pstring**. The **formatted** value of **power** will eventually go into **pstring**.

2. We've created a new number format called **frac** with a pattern of **"1/0"**. This will format a number to look like a **fraction** in the form of **1/n**. You'll see why we want this in a bit.
3. Even though **power** is defined as a **double**, we're using **gatling.getInt** to get the value of it. That's because even though we'll use it as a double, we want to force the user to enter a **whole number**.
4. In the calculation for **root**, remember that the **nth root** of a number is equal to that number to the **1/n power**. For example, the **square root of two** is the same as $\mathbf{2^{1/2}}$.
5. As mentioned in #2, the String **pstring** is the power formatted look like a fraction by using the **frac** format.

OK, compile it, run it, check it out.

A Few Final Words

So now you have a simple program that gives you four choices of things to do, and the kicks idiots out of the program. One of the really important things you've learned here, in addition to how to make choices run from other **internal methods**, is how to keep your **main method** slim and trim by doing so.

Keeping your **main method**, and in fact, all of your methods, as simple as possible is a very important part of programming in Java or any other language. If you can possibly make the body of your code simpler and easier to follow by farming out other jobs to little methods that you can test one at a time, then by all means do so.

Pretty soon you'll find that when we write programs, the only purpose of the **main method** will be to repeatedly call the methods that really do the work. But before we there, we have to learn to count – again and again and again. You'll learn about that in handout #15.

But first we have to take a look at complex comparisons in handout #14.

This page intentionally left **almost** blank