Mr G's Java Jive

#17: Random Numbers

One of the things Java is really great at dealing with is **random numbers**. Why would you want to work with random numbers? Because they're practically indispensable when it comes to doing statistical modeling. You also need them in order to make any kind of decent game.

A Flip of the Coin

We'll find out how to create and use random numbers by writing a simple program to flip a coin.

Random numbers are created by using a **Random object**. And in order to create a Random object, you'll need to import **java.util.Random**.

Check it out, and notice that we're using the guts of BigLoop so we can run this program over and over again.

```
//CoinToss
//a program to flip a coin
//2.24.07 Clark Kent

import java.text.*;  //needed for number formats
import java.util.Random; //needed for random numbers

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

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

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

   //======== main parking space
     public static void main()
     {//start main
        //variables
          char ans = 'y';

        //big loop
          while (ans=='y')
          {//start while
             //insert method call below

             //ask to do over
               System.out.print("\nRun again (y/n)? ");
               ans=gatling.getAns();
          }//end while

        //ending
          System.out.println("\nWe're finally done here. See ya!");

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

As usual, compile and run this to see if it works so far.

Coinmain

In the **RootLoop** program we created a new method called **rootmain()** that was used to do all of the work, and all that **main()** did was to keep calling **rootmain()** over and over until the user finally decided to quit.

In this program, we're going to create a method called **coinmain()** that serves the exact same function. This is where all the action takes place, and the only job **main()** has is to keep calling **coinmain()**.

Check out the code below for a very simple starter version of **coinmain()**.

```
public class CoinToss
{//start class
   //======== parking space 1
     public static void coinmain()
     {//start coinmain
        System.out.println("\tThis is coinmain");
     }//end coinmain
   //======== parking space 2
```

All this does for now is announce to the user that it exists. We'll come back to this later. What's that **\t** doing in the **println** statement? That's how you tell Java that you want the text to **tab in** a few characters. You'll see the results when you run the program.

But in order to see this, you'll have to actually use **coinmain()**. Check out the code below for the changes you'll need to make.

```
//big loop
          while (ans=='y')
          {//start while
             //insert method call below
               coinmain();

             //ask to do over
```

Make sure it compiles and runs, and then we can move on.

Random Behavior

A coin toss is the simplest example of random behavior there is. That's because instead of millions of possible choices, you only have two: **heads** or **tails**. However, as simple as this is in theory, there's one complicating little factor. Java can't easily randomly pick heads or tails. All Java can do is pick randomly from a **spread of numbers** that represent our choices. After that, it's up to us to assign a value of heads or tails to the different possible choices in the spread.

But first we'll create our Random object and an int variable to use with it. Make the following changes to **coinmain()**:

```
public static void coinmain()
{//start coinmain
   System.out.println("\tThis is coinmain");
   //variables
     Random myRandom = new Random();
     int flipnum;

   //do stuff
     flipnum=myRandom.nextInt(2);
     System.out.println(flipnum);
}//end coinmain
```

The line **Random myRandom = new Random()** creates a new Random object. The expression **myRandom.nextInt(2)** means that we want to pick the next number from a **spread** of **two integers**. Those two integers are 0 and 1. This is because counting in Java (as in most programming languages) actually starts at zero. If you think about how **decades** of numbers work (0-9, 10-19, 20-29, etc), this actually makes a lot of sense.

When you compile and run the program, you should get an almost even number of 0s and 1s. Which ones are heads? Which ones are tails? I don't know. That's for you to determine, and it's the perfect opportunity for either a class or a homework assignment.

A Roll of the Dice

The CoinToss program was fairly simple because it only involved two choices that really had no inherent numerical value. But suppose you needed to write a program for something that did – say a set of dice? With dice, not only do you have six possibilities for each die, but the choices and values have to match. Then you have the complicating issue of having to do this for two dice.

But not to worry, it's not that hard, and it gives us the opportunity to learn something else about random numbers.

Create a new program, based on **BigLoop**, and call it **RollDice**. In this program, you'll need to create a new method called **rollmain()** (are you getting the hang of this?). The **rollmain()** method should look like the code shown below:

```
public class RollDice
{//start class
   //======== parking space 1
     public static void rollmain()
     {//start rollmain
        //variables
          Random myRandom = new Random();
          int die1, die2;

        //do stuff
          die1=myRandom.nextInt(6)+1;
          die2=myRandom.nextInt(6)+1;
          System.out.println("\t"+die1+", "+die2);

     }//end rollmain
   //======== parking space 2
```

Here, something new has happened. Our **myRandom.nextInt** expression, not only has a spread, but has a number added to it. What's up with that?

The answer is really quite elementary. If there are six sides to a die, then it makes perfect sense that you'd want the spread to be 6. However, with that spread of 6, the numbers generated would from 0-5. In order to get the numbers to run from 1-6, you have to add 1 to each one. Hence the expression of **myRandom.nextInt(6)+1**.

The Fancier Spread

Sometimes you need something even a little fancier than what we just did with the dice. Suppose you need to pick a random number between 100 and 200. In this case you'd want to use the expression **myRandom.nextInt(101)+100**.

Why? Because the spread between 100 and 200, **including the numbers at each end**, is 101. But if you just set up a spread of 101, you'll only get numbers from 0-100. That's why you have to add 100 to each number.

A slightly trickier example is wanting to pick a random number between –50 and 50. In this case the spread is 101 again, but now you're going to **subtract 50** from each of the numbers generated. This means that your expression should be **myRandom.nextInt(101)-50**.

The Key to Random Numbers

Does all of this have you confused? Not to worry. The little chart below will make it all much easier for you to understand. I hope.

Expression	Result
myRandom.nextInt()	Generates a random integer within the **entire range of positive and negative integer values**. This probably isn't what you want.
myRandom.nextInt(spread)	Generates a random integer within the spread given. Possible values run from 0 to spread-1. The value of spread has to be a **positive integer**.
myRandom.nextInt(spread)+startpoint	Generates a random number from startpoint to startpoint+(spread-1). Once again, the value of spread has to be a **positive integer**. However the value of startpoint can be any positive or negative integer.

Easier Random Numbers

The way that Java handles creating random numbers is very powerful and flexible, but it's also very confusing to a beginner. Why can't Java have a simple method that allows you to enter the two numbers you want it to pick a number between, without you having to figure out the **spread** and the **startpoint**? One of my students asked the same question, and the result is **gatling.randomInt(int x, int y)**. Try it out.

Random Letters

I learn a lot from my students, and it eventually shows up in these handouts. One of those students needed to generate random letters of the alphabet. He wrote a really great line of code to do that, but I decided that it would be a lot easier if there were a simple method you could call on to get the same results. That method is **gatling.randomChar()**. Try that one out too.

More Information

Both of those random methods will become very useful in the future, and if you're so inclined, you can "take a look under the hood" to see how they work.

That's all I'm going to mention about random numbers in this handout. If you want to find out a lot more, check out the official page at **http://java.sun.com/j2se/1.5.0/docs/api/java/util/Random.html**. This goes into detail about more random variations than you probably care about right now – especially since practically everything you'll need to do can be accomplished by using one of the **gatling.random** methods.

Next up: dates.