Class Libraries and Packages

There are many, prewritten Java classes for you to use, just like the String class. Groups of these classes that are related are put into packages. For example, the String class is included in the java.lang package. Normally, when you use a prewritten java class, you must import the corresponding class. However, the java.lang package is automatically included to use for every java program. That is why no import statement appears in the code we just looked at.

However, let's say for example that you wanted to create a Random object, which will help you generate random numbers. This class appears in the java.util package. In order to use this class you would have to include one of the two following statements in your code:

```
import java.util.*;
```

OR

```
import java.util.Random;
```

The first statement imports the whole java.util package which contains the Random class AND other classes, whereas the second statement JUST imports the Random class itself.

The general rule of thumb that I use is if I am only using one class from a package, I will only import that class. Otherwise I will import the package.

Before we look at the Random class, let’s take a look at the Math class. It allows us to use many standard mathematical functions.

Math Class

All of the methods in the Math class are static methods. This means that there is no object associated with the methods. (Note that in the String class each method IS associated with an object.) Whereas to call a String method you have to call them on a String object, Math methods get called on the class. Thus, for any Math method you simply call it with "Math" followed by a dot, followed by the name of the method.

Invoking Class Methods (Static)

Sometimes methods inside of a class are NOT specific to an object of the class, but simply pertain to the class itself. In order to call these methods, you do NOT need to call them on an object of the class. Rather, you may call them simply my specifying the class in which they reside.

For example, here are the prototypes of some methods from the Math class, (which is part of the java.lang package.)

```
// Returns the absolute value of num.
static int abs (int num);

// Returns the cosine of angle, where angle is expressed in
// radians.
static double cos(double angle);

// Returns the greatest integer less than or equal to num.
static double ceil(double num);

// Returns e (the base of the natural log) raised to the
// exponent power.
static double exp(double power);

// Returns num raised to the exponent power.
static double pow(double num, double power);

// Returns a random double uniformly distributed in between
// 0.0 and 1.0.
static double random();

// Returns the square root of num.
static double sqrt(double num);
```

Since these methods are static they do not need to be called on an object of the class.

Instead, they are called by the class, as follows:

Classname.method(<parameters>)

Here is an example:

```
System.out.println("The square root of 15 is
"+Math.sqrt(15));
```

All of these methods do NOT operate on an object, but rather carry out some generic task.

Incidentally, the Math class has two constants defined:

E – the double value that is closer than any other to e, the base of the natural logarithm

PI – The double value that is closer than any other to pi, the ration of the circumference of a circle to its diameter.

These can be referred to in code as Math.E and Math.PI, respectively.

Let's use the math class to solve the following two problems:

(1) Calculate the amount of money earned after accruing simple interest for several years.
(2) Calculate the amount of money earned after accruing continuously compounding interest for several years.

The formula for simple interest is simply

Money = $P(1 + r)^t$, where P represents the amount of money initially invested, r represents the interest rate as a decimal and t represents the number of years the money is invested.

In our program, we'll prompt the user for those three pieces of information and then we'll tell them how much money they've earned.

```
import java.util.*;

public class Earnings {

    public static void main(String[] args) {

        Scanner stdin = new Scanner(System.in);

        System.out.println("Enter the principle.");
        double money = stdin.nextDouble();
        System.out.println("Enter the rate.");
        double rate = stdin.nextDouble();
        System.out.println("Enter the number of years.");
        double numyears = stdin.nextDouble();

        double total = money*Math.pow(1+rate, numyears);

        System.out.println("You earned "+(total-money)+".");
    }
}
```

The key in this example is to make the appropriate method call. We want to take the quantity (1+rate) and raise it to the power numyears. That's why the method call to the math class is:

```
Math.pow(1+rate, numyears)
```

Then we want to take this answer and multiply it by the original amount of money invested. To get our final earnings, we have to subtract the original amount of money out of the total all together. Note that this implementation just prints out that value without ever storing it in a particular variable.

The formula for continuously compounded interest is

Money = Pe^{rt}.

We just have to change the total line in our previous program to implement continuously compounded interest. It should be changed to the following:

```
double total = money*Math.exp(rate*numyears);
```

They key here is that we want to raise e to the power rate*numyears. Thus, we must simply pass this latter value as a parameter to the exp method.

The Random Class

Here are some commonly used methods in the Random class:

```
//constructor
Random();

// Returns a random real number uniformly distributed in
// between 0.0 and 1.0.
float nextFloat();

// Returns a random integer in between -2^31 and 2^31 – 1.
int nextInt();

// Returns a random integer in between 0 and max-1.
int nextInt(int max);
```

For most programs you need to generate random numbers, creating a Random object and making calls to the two other methods listed above should be sufficient. Generally in most programs you only want to create one instance of a Random object:

Random gen = new Random();

Once you have done this, for the rest of your program you can simply call the nextFloat() or nextInt() method on the gen object. Each call will return a random float in between 0 and 1 or a random int in between -2^{31} and $2^{31} - 1$.

If you need to generate random numbers within some method, you can either create a Random object in that method OR you can simply pass a random number generator in as a parameter to that method.

The standard example used to illustrate the user of the Random class is a game where the computer generates a random number in between 1 and 100 and the user keeps on guessing until they get the number. (After each guess, the user is told whether their guess is too low or too high.)

Since we are only generating one random number for this program, the segment of code that will achieve this is as follows:

```
Random gen = new Random();
int secret = Math.abs(gen.nextInt()%100) + 1;
```

Note that the nextInt method doesn't automatically give us a random number in between 1 and 100. Instead, when we mod this by 100 and take the absolute value of it, that gives us a number in the range 0 to 99, inclusive. By adding 1 to this, we get a range from 1 to 100, inclusive.

Incidentally, since there's another method called nextInt in the Random class that automatically generates a random integer in between 0 and some limit, we can do the following also:

```
int secret = gen.nextInt(100) + 1;
```

Once the secret number is generated, we successively ask the user for their guess and then respond with whether or not that guess was too high or too low. The code for this program in its entirety is included in the list of sample programs.

Another example that illustrates how to generate multiple random numbers (using only ONE Random object) is allowing the user to play a game of Craps. In this game, the user rolls to fair six-sided dice. If the sum is 7 or 11, they win. If the sum is 2, 3, or 12, they lose. If the number is not one of these, the game continues. This first sum rolled is known as the player's point. From there, the player continues to roll the pair of dice until they get either their point or 7. If they get the point first, they win. If they get 7 first, they lose.

The key idea here is that we instantiate one Random object:

```
Random r = new Random();
```

Once we do that, we can call the nextInt method on this same Random object as many times as we want. For our initial roll, we have:

```
int die1 = Math.abs(r.nextInt())%6 + 1;
int die2 = Math.abs(r.nextInt())%6 + 1;
```

From that point on, if the game continues, we can have the lines

```
die1 = Math.abs(r.nextInt())%6 + 1;
die2 = Math.abs(r.nextInt())%6 + 1;
```

inside of a loop that runs until the point or seven is hit.

The full example is included in the sample programs.

One way in which the nextFloat method could be used would be if we wanted to simulate a baseball program. Say we know that a player's batting average is .257. (This means they get a hit 25.7% of the time.) Then we can simulate them hitting as follows (using the same random object r):

```
double value = r.nextFloat();

if (value <= .257)
    System.out.println("You got a hit!");
else
    System.out.pritnln("You are out.");
```

Other examples of programs that can be written using random numbers:

(1) A Price is Right Program, where the price is generated as a random number and two users try to guess as close to that number without going over.

(2) Any sort of game where the user plays against an opponent who moves or chooses randomly. (This is very wide open whereas the last idea was very specific.)