

These exercises focus on the *Random* class defined in the Java Standard Class Library.

Introduction and Background

In many situations a program needs to generate a random number in a certain range. The standard library class “*Random*” lets the programmer create an object of type *Random* and then use that object to generate a stream of random numbers (one at a time). The following statement declares the variable *generator* to be an object of type *Random* and instantiates it with the *new* operator.

```
Random generator = new Random();
```

The *generator* object can be used to generate either integer or floating point random numbers using either the *nextInt* method (either with no parameter or with a single integer parameter) or *nextFloat* (or *nextDouble*) methods, respectively. The integer returned by *nextInt()* can be any integer (positive or negative) whereas the number returned by *nextInt(n)* is a random integer in the range 0 to (n-1). The numbers returned by *nextFloat()* or *nextDouble()* are floating-point (decimal) numbers between 0 and 1 (up to but not including 1). Most often the goal of a program is to generate a random integer in some particular range, say 30 to 99 (inclusive). There are several ways to do this:

- **Using *nextInt()*:** If we use this method, we must use the % operator to reduce the range of values. For example,

```
Math.abs(generator.nextInt()) % 70
```

will return numbers between 0 and 69 (because those are the only possible remainders when an integer is divided by 70 - note that the absolute value of the integer is first taken using the *abs* method from the *Math* class). In general, using % N will give numbers in the range 0 to N - 1. Next the numbers must be shifted to the desired range by adding the appropriate number. So, the expression

```
Math.abs(generator.nextInt()) % 70 + 30
```

will generate numbers between 30 and 99.

- **Using *nextInt(int n)*:** The expression

```
generator.nextInt(70)
```

will return numbers between 0 and 69 (inclusive). Next the numbers must be shifted to the desired range by adding the appropriate number. So, the expression

```
generator.nextInt(70) + 30
```

will generate numbers between 30 and 99.

- **Using *nextFloat()*:** In this case, we must multiply the result of *nextFloat* to expand the range—

for example,

```
generator.nextFloat() * 70
```

returns a floating point number between 0 and 70 (up to but not including 70). To get the integer part of the number we use an explicit “cast” (short for “typecast”) to convert the result to an int:

```
(int) (generator.nextFloat() * 70)
```

The result of this is an integer between 0 and 69, so

```
(int) (generator.nextFloat() * 70) + 30
```

shifts the numbers by 30, resulting in numbers between 30 and 99.

The method *nextFloat* can be replaced by *nextDouble* to get double-precision floating point numbers rather than single-precision.

Exercises

1. Lucky Numbers

Create a new BlueJ project for today's in-class work. Download the source file `LuckyNumbers.java` from the course website and save it the project folder for the new project you just created. From within BlueJ, use the Edit-->Add Class From File... menu item to import the `LuckyNumbers` class into your project. Open the editor for the class and fill in the blanks in the source code to generate the random numbers as described in the program comments. Compile the class and test the methods; debug if necessary.

2. Dice

Now create a new class called “Dice” (use the Edit-->New Class... menu item in BlueJ). This class will represent a pair of standard six-sided dice. It should have two integer fields “die1” and “die2”. The constructor should take no parameters and it should initialize the fields to zero. Provide simple accessor methods `getDie1()` and `getDie2()`. Your class should also have a method called “rollDice” that does the following:

1. Generates a random integer between 1 and 6 and assigns it to die1
2. Generates a random integer between 1 and 6 and assigns it to die2
3. Prints a message showing the result of the roll. For example, if the values of die1 and die2 happen to be 3 and 4, your method should print something like the following:

```
(3 and 4) ----> 7
```

3. Dice Game (optional)

If you finished the first two exercises, here's a challenge for you: create a new class that represents a dice game of your choosing (craps is a good one, but you might know others). The design of the class is up to you. Your new class should probably use the `Dice` class from the previous exercise, although you may need to add additional methods to the `Dice` class. The methods of the `Dice` class, however, should be generic: `Dice` objects should be able to be used in many different sorts of dice games.