CSE1007 Java Programming In-Lab Activity L3+L4 on 23-08-2021

Sharadindu Adhikari | 19BCE2105

Prepare the report on the following:

1. How java can intricate with developer's Operating System with an example code discuss in details
2. How do generate random numbers using Java and list out the impartments of it.
3. Discuss about timing events to calculate total execution time of java code.

Answers:

Question 1.

Before the advent of Windows and Macintosh things were done predominantly with the command line. We can build applications for Windows, or Mac and Linux for that matter, that strictly use the command line and that is how the user interacts with the application. Many of the courses here on Treehouse rely on the command line for interaction. It works, but in this day and age of Windows and mobile devices, isn't thought of as very user friendly.

Enter the world of GUI, or Graphical User Interface. This will allow a user to, for example, use their mouse and have more visually stimulating interactions with the program. One can certainly program all of that stuff by oneself, mouse clicks, windows, tabs, forms, etc. but that *can* get rather complicated. Packages like JavaFX allow developers to easily create a graphical interface for Java applications that will run on devices that use the JVM (Java Virtual Machine).

Console applications run strictly in the console. 😉

We would have interaction similar to

```
Input your name:
Sharad

Hello, Sharad, I'm a console app.
```

If we start through the Treehouse Java courses they predominantly demonstrate console based applications instead of a Windows like user face or web based interface. It is my understanding that Java and web application courses are due in the near future here at Treehouse.

Question 2.

In Java programming, we are often required to generate random numbers (within a specific range of type integer, float, double, long, and Boolean) while we develop applications. Many applications have the feature to generate numbers randomly, such as to verify applications using OTP.

There are three methods to generate random numbers in Java:

Method 1: Using random class

To use the Random Class to generate random numbers, follow the steps below:

1. Import the class `java.util.Random`
2. Make the instance of the class Random, i.e., `Random rand = new Random()`
3. Invoke one of the following methods of rand object:
    - `nextInt(upperbound)` generates random numbers in the range 0 to `upperbound-1`.
    - `nextFloat()` generates a float between 0.0 and 1.0.
    - `nextDouble()` generates a double between 0.0 and 1.0.

```
import java.util.Random;
class GenerateRandom {
    public static void main( String args[] ) {
      Random rand = new Random(); //instance of random class
      int upperbound = 25;
        //generate random values from 0-24
      int int_random = rand.nextInt(upperbound);
      double double_random=rand.nextDouble();
      float float_random=rand.nextFloat();

      System.out.println("Random integer value from 0 to" + (upperbound-
1) + " : "+ int_random);
      System.out.println("Random float value between 0.0 and 1.0 : "+float_random);
      System.out.println("Random double value between 0.0 and 1.0 : "+double_random)
;
    }
}
```

```
Output
Random integer value from 0 to24 : 21
Random float value between 0.0 and 1.0 : 0.91133624
Random double value between 0.0 and 1.0 : 0.47951718643784025
```

Method 2: Using Math.random

For generating random numbers within a range using `Math.random()`, follow the steps below:

1. Declare the minimum value of the range
2. Declare the maximum value of the range
3. Use the formula `Math.floor(Math.random()*(max-min+1)+min)` to generate values with the `min` and the `max` value inclusive.

This method, however, can only be used if we need an integer or float random value.

```
class GenerateRandom {
    public static void main( String args[] ) {
      int min = 50;
      int max = 100;

      //Generate random int value from 50 to 100
```

```
      System.out.println("Random value in int from "+min+" to "+max+ ":");
      int random_int = (int)Math.floor(Math.random()*(max-min+1)+min);
      System.out.println(random_int);
    }
}
```

Output

```
Random value in int from 50 to 100:
55
```

Method 3: Use ThreadLocalRandom

To generate random numbers using the class `ThreadLocalRandom`, follow the steps below:

1. Import the class `java.util.concurrent.ThreadLocalRandom`
2. Call the method
    - To generate random number of type int `ThreadLocalRandom.current().nextInt()`
    - To generate random number of type double `ThreadLocalRandom.current().nextDouble()`
    - To generate random number of type boolean `ThreadLocalRandom.current().nextBoolean()`

```
import java.util.concurrent.ThreadLocalRandom;
class GenerateRandom {
    public static void main( String args[] ) {
      // Generate random integers
      int int_random = ThreadLocalRandom.current().nextInt();

      // Print random integers
      System.out.println("Random Integers: " + int_random);

      // Generate Random doubles
      double double_rand = ThreadLocalRandom.current().nextDouble();

      // Print random doubles
      System.out.println("Random Doubles: " + double_rand);

      // Generate random booleans
      boolean boolean_rand = ThreadLocalRandom.current().nextBoolean();

      // Print random booleans
      System.out.println("Random Booleans: " + boolean_rand);
    }
}
```

Output

```
Random Integers: 1199423179
Random Doubles: 0.5762662573361096
Random Booleans: true
```

Question 3.

There are two ways to measure elapsed execution time in Java either by using System.currentTimeinMillis()or by using System.nanoTime(). These two methods can be used to measure elapsed or execution time between two method calls or events in Java. Calculating elapsed time is one of the first things Java programmer do to find out how many seconds or millisecond a method is taking to execute or how much time a particular code block is taking. Most Java programmers are familiar with System.currentTimeInMillis()which is there from the beginning while a new version of more precise time measurement utility System.nanoTime is introduced in Java 1.5, along with several new features in a language like Generics, Enum types, autoboxing and variable arguments or varargs.

You can use any of them to measure the execution time of the method in Java. Though its better to use System.nanoTime() for more precise measurement of time intervals. In this Java programming tutorial, we will see a simple Java program to measure execution time by using System.nanoTime() and Spring framework's StopWatch utility class. This article is in continuation of my post on covering fundamental Java concepts like How to compare String in Java, How to write equals method in Java correctly, and 4 ways to loop HashMap in Java. If you haven't read them already you may find them useful.

Java Program example to measure execution time in Java:

Here is a **code example for measuring the elapsed time between two code block**s using the System.nanoTime, Many open-source Java libraries like Apache commons-lang, Google commons, and Spring also provide StopWatch utility class which can be used to *measure elapsed time in Java*.

The StopWatch improves readability to minimize calculation error while calculating elapsed execution time but beware that StopWatch is not thread-safe and should not be shared in a multi-threading environment and its documentation clearly says that this is more suitable in development and test environment for basic performance measurement rather performing time calculation in a production environment.

```
import org.springframework.util.StopWatch;

/**
 * Simple Java Program to measure elapsed execution time in Java
 * This Java Program shows two ways for measuring time in Java, by using System.nanoTime
() which was
 * added in Java 5 and StopWatch which is a utility class from Spring Framework.
 */
public class MeasureTimeExampleJava {



    public static void main(String args[]) {

        //measuring elapsed time using System.nanoTime
        long startTime = System.nanoTime();
        for(int i=0; i< 1000000; i++){
            Object obj = new Object();
        }
        long elapsedTime = System.nanoTime() - startTime;

        System.out.println("Total execution time to create 1000K objects in Java in mill
is: "
                + elapsedTime/1000000);
```

```
        //measuring elapsed time using Spring StopWatch
        StopWatch watch = new StopWatch();
        watch.start();
        for(int i=0; i< 1000000; i++){
            Object obj = new Object();
        }
        watch.stop();
        System.out.println("Total execution time to create 1000K objects in Java using S
topWatch in millis: "
                + watch.getTotalTimeMillis());
    }

}
```

Output:

```
Total execution time to create 1000K objects in Java in millis: 18
Total execution time to create 1000K objects in Java using StopWatch in millis: 15
```
