Methods in Java – Basics

As the complexity of a task increases, it becomes more and more difficult to see the entire solution at once. It is often useful to break the problem into manageable pieces. After solving each subproblem, the solutions can be combined into a solution to the entire problem.

This is known as *modular programming*, where the program is broken into separate parts, or *modules*, which are then assembled into an overall solution. In Java, the primary structure for modularity is the *method*.

Modular programming offers some distinct advantages:

- Break the problem into smaller pieces, making them easier to understand and solve.
- Multiple programmers can work on the same overall problem by solving its subproblems.
- Modules can be reused in other parts of the program.
- Some modules can be reused in other programs.
- A module is easier to test in detail than a larger program solving a complicated problem.

Methods as Modular Programming

Methods should already be familiar. Many first programs use the `println` method to produce output. The String class has built-in methods such as `equals` and `compareTo`, and the Math class has numerous useful methods for mathematical operations. So far, the only method we have written is the `main` method, required in all programs, but now we will begin writing our own methods.

Example 1 – Suppose we are writing a program that requires the identical address information to be output at regular intervals.

```
Ahmed's Video Wonderland
  123 Roehampton Avenue
    613-555-1212
```

The following code can be used to output this information whenever we need it by calling the associated method. This is the *method definition*.

```
public static void printHeading ()
{
  System.out.println();
  System.out.println("Allan's Video Wonderland");
  System.out.println("   123 Roehampton Avenue");
  System.out.println("      613-555-1212");
}
```

This code very closely resembles our previous programs, except that instead of a main method, this method is called `printHeading`. Also notice that the parentheses following `printHeading` are

empty.

To use this method, the simplest option is to call, or *invoke*, the `printHeading` method from within the main method of the same class. When the main method executes, it finds the reference to the `printHeading` method, and executes the code from `printHeading`.

```
class Sample
{
  public static void printHeading ()
  {
       System.out.println();
       System.out.println("Ahmed's Video Wonderland");
       System.out.println("   123 Roehampton Avenue");
       System.out.println("      613-555-1212");
  }

  public static void main (String[] args)
  {
       printHeading();
       printHeading();
       printHeading();
  }
}
```

Note that we need to include the parentheses and semicolon with the call to `printHeading` in the main method. This lets Java know that `printHeading` is a method, and not simply a variable.

Example 2 – The `printRoot` method will ask the user for a value and then find and print its square root. If the number is less than zero, the method prints an error message.

```
public static void printRoot ()
{
      System.out.println("Please give a non-negative value");
      double x = In.getDouble();

      if (x >= 0)
      {
            System.out.println("Square root is " + Math.sqrt(x));
      }
      else
      {
            System.out.println("Square root of a negative is not allowed");
      }
}
```

Exercises

1. Explain the difference between a method *definition* and a method *invocation*.

2. In the following program, the executable statements are numbered. Use these numbers to indicate the order in which the statements are executed.

```
class Song
{
      public static void printChorus ()
      {
            System.out.println();
            System.out.println("Ee-igh, ee-igh, oh!");            //1
            System.out.println();
      }

      public static void main (String[] args)
      {
            System.out.println("Old McDonald had a farm");        //2
            printChorus();
            System.out.println("And on that farm he had a pig");  //3
            printChorus();
      }
}
```

3. Write a method that will simulate the results of rolling a single fair die by printing a random integer value in the range 1 to 6. Write a program to test your method. Sample output from a call to the method could be:

```
You rolled a 3!
```

4. Write a method that will simulate the results of rolling two fair dice by printing two random integer values in the range 1 to 6 along with their total. Write a program to test your method. Sample output from a call to the method could be:

```
4 and 3: a total of 7
```

Solutions

1. Explain the difference between a method *definition* and a method *invocation*.

   The method definition is the code that tells the compiler what the method will do. It could have many lines of code. The method invocation occurs when we use that method from the main method, which only requires a single line of code.

2. In the following program, the executable statements are numbered. Use these numbers to indicate the order in which the statements are executed.

   2, 1, 3, 1

```
class Song
{
      public static void printChorus ()
      {
            System.out.println();
            System.out.println("Ee-igh, ee-igh, oh!");            //1
            System.out.println();
      }

      public static void main (String[] args)
      {
            System.out.println("Old McDonald had a farm");        //2
            printChorus();
            System.out.println("And on that farm he had a pig");  //3
            printChorus();
      }
}
```

3. Write a method that will simulate the results of rolling a single fair die by printing a random integer value in the range 1 to 6. Sample output from a call to the method could be:

```
You rolled a 3!
```

```
class Methods1Exercise3
{
  public static void main(String [] args)
  {
    printSingleRoll();
  }

  public static void printSingleRoll()
  {
    int roll = (int)(6*Math.random()) + 1;
    System.out.println("You rolled a " + roll);
  }
}
```

4. Write a method that will simulate the results of rolling two fair dice by printing two random integer values in the range 1 to 6 along with their total. Sample output from a call to the method could be:

```
4 and 3 - a total of 7
```

```
class Methods1Exercise4
{
  public static void main(String [] args)
  {
    printRollTwoDice();
  }

  public static void printRollTwoDice()
  {
    int roll1 = (int)(6*Math.random()) + 1;
    int roll2 = (int)(6*Math.random()) + 1;
    System.out.print(roll1 + " and " + roll2);
    System.out.println(" - a total of " + (roll1 + roll2));
  }
}
```