Methods in Java – Passing Parameters

To make methods more flexible, we often want to give them different values to use in their processing and calculations. Consider the following method, which outputs any character a specified number of times on the same line.

Example 1 – Printing a character and specified number of times using parameters

```
public static void printRow (char c, int n)
{
  for (int i = 1; i <= n; i++)
  {
    System.out.print(c);
  }
}
```

In the definition of the method `printRow`, the `char c` and `int n` are called the *parameter list*. They identify the variables `c` and `n` as *parameters* of the method. This method might be called by

```
printRow('*', 10);
```

which will result in ten asterisk characters output on the same line. The character `'*'` and the number 10 are the arguments of the method, and their values are *passed* to the parameters of the method.

In this case, the arguments of the method are *constants*, but it is also valid to use *variables* or *expressions* (mathematical or logical) as arguments for a method.

When the argument to a method is a *variable*, it is important to understand that a copy of the value is passed to the method. Thus the original value contained in the variable does not change, regardless of what happens to the parameter inside the method.

Example 2 – Passing a Variable as a Parameter to a Method

Suppose we have a method that, among its other statements, increments the input parameter by one. It might look like the following:

```
public static void sample (int n)
{
   ...          // other useful code
   n++;
   ...          // other useful code
}
```

Suppose we were to call this method by writing

```
int n = 3;
sample(n);
```

The argument of the method call is the variable, `n`, containing the value of 3. As part of the *parameter passing* process, a copy is made of the value of `n` (3), and it is this copy that is received by the actual method. So even though the method changes the value of `n` to 4 inside the method, the actual value of the variable `n`, in the calling code, is unchanged.

Exercises

1. A student wrote the following method to exchange the values of two integer variables.

```
public static void swap (int m, int n)
{
 int temp = m;
 m = n;
 n = temp;
}
```

He then tested the method with the following:

```
i = 7;
j = 3;
swap(i, j);
System.out.println("i = " + i + " and j = " + j);
```

What did the fragment output? Explain.

2. Write a method that will simulate the results of rolling N fair dice, where N is an integer parameter provided to the method. For example, given the following signature for the method,

```
public static void printRollDice(int numDice)
```

then calling the method with a '5',

```
printRollDice(5);
```

might produce the output

```
Roll 1 is a 3.
Roll 2 is a 6.
Roll 3 is a 2.
Roll 4 is a 3.
Roll 5 is a 5.
```

3. a) Complete the definition of the following method, `printRect`, so that it prints a filled rectangular pattern, using the character `c`, that is `width` characters wide and `height` characters high.

```
public static void printRect (char c, int width, int height)
```

For example, the following call to `printRect`

```
printRect('*', 6, 4);
```

should produce the following output:

```
******
******
******
******
```

b) Copy and modify your original method so that it prints an *open* rectangular pattern with blanks in the interior, and change the name of the method to `printRectHollow`, so that a call such as

```
printRectHollow('*', 6, 4);
```

should produce the following output:

```
******
*    *
*    *
******
```

Solutions

1. A student wrote the following method to exchange the values of two integer variables.

```
public static void swap (int m, int n)
{
 int temp = m;
 m = n;
 n = temp;
}
```

He then tested the method with the following:

```
i = 7;
j = 3;
swap(i, j);
System.out.println("i = " + i + " and j = " + j);
```

What did the fragment output? Explain.

The output will be:

```
i = 7 and j = 3
```

These are the original values, and they are unchanged by the method. When a parameter is passed to a method, a copy is made, so the original values are unchanged, regardless of what happens inside the method.

2. Write a method that will simulate the results of rolling N fair dice, where N is an integer parameter provided to the method. For example, given the following signature for the method,

```
class MethodsPassingParmsEx2
{
  public static void main (String [] args)
  {
    printRollDice(5);
  }

  public static void printRollDice(int numRolls)
  {
    int random = 0;
    for (int i = 1; i <= numRolls; i++)
    {
      random = (int)(6 * Math.random()) + 1;
      System.out.println("Roll " + i + " is " + random);
    }
  }
}
```

3. a) Complete the definition of the following method, `printRect`, so that it prints a filled rectangular pattern, using the character `c`, that is `width` characters wide and `height` characters high.

`public static void printRect (char c, int width, int height)`
For example, the following call to `printRect`

```
printRect('*', 6, 4);
```

should produce the following output:

```
******
******
******
******
```

```
public static void printRect(char symbol, int width, int height)
{
  // print one row at a time (start with height)
  for (int h = 1; h <= height; h++)
  {
    // print all symbols in this row
    for (int w = 1; w <= width; w++)
    {
      // print the next symbol for this line
      System.out.print(symbol);
    }
    // reached end of row, go to next line
    System.out.println();
  }
}
```

b) Copy and modify your original method so that it prints an *open* rectangular pattern with blanks in the interior, and change the name of the method to `printRectHollow`, so that a call such as

```
printRectHollow('*', 6, 4);
```

should produce the following output:

```
******
*    *
*    *
******
```

```
public static void printRectHollow(char symbol, int width, int
height)
{
  // print one row at a time (start with height)
  for (int h = 1; h <= height; h++)
  {
    // print all symbols in this row
    for (int w = 1; w <= width; w++)
    {
      // if we are on an edge, print the symbol
      // otherwise, print a blank space
      if (h==1 || h==height || w==1 || w==width)
        System.out.print(symbol);
      else
        System.out.print(' ');
    }
    // reached end of row, go to next line
    System.out.println();
  }
}
```