Methods in Java – Return Values

Methods can be divided into two groups – *commands* and *queries*. In many programming languages, these are known as *procedures* (commands) and *functions* (queries).

A *command* is a method that performs a task and then returns control to the point where they were called. A *query* is a method that is used to calculate or determine some value, and that value is then returned to the point where the method was called.

Example 1 – The following method calculates values of the function whose defining equation is

$$f(x)=\begin{cases} x^2 & \text{if } x<0 \\ x^3 & \text{if } x\geq 0 \end{cases}$$

```
public static double f (double x)
{
  double y;
  if (x < 0)
  {
    y = x * x;
  }
  else
  {
    y = x * x * x;
  }
  return y;
}
```

Two notes about the differences in this method as compared to previous methods:

1. The keyword `void` has been replaced by the keyword `double`. Previous methods returned nothing (`void`), while this one returns a `double`.
2. The method contains the statement `return y;` rather than simply ending. This indicates that the method should finish at this point, and send the value `y` back to the point where the method was called.

The return statement does not need to be at the end of the method. In fact, there can be more than one return statement (if appropriate logic is used). For example, an alternative to the above code might be

```
public static double f (double x)
{
  double y;
  if (x < 0)
  {
    return x * x;
  }
  else
  {
    return x * x * x;
  }
}
```

We have already used examples of this type of method from the Math class, such as `Math.sqrt` or `Math.random`.

Boolean Methods

Through the use of *boolean methods*, we can greatly improve the clarity of our code through the use of *queries* that return either true or false. Many boolean expressions can be quite complicated, and to package that logic into a clearly identified method greatly improves the readability of code.

For example, the code fragment

```
if (13 <= age && age <=19)
```

could be replaced by

```
if (isTeen(age))
```

where `isTeen` is a `boolean` method.

```
public static boolean isTeen (int n)
{
      if (13 <= n && n <= 19)
      {
            return true;
      }
      else
      {
            return false;
      }
}
```

A more condensed version of the method is also possible, while the clarity of the method name is preserved.

```
public static boolean isTeen (int n)
{
      return 13 <= n && n <= 19;
}
```

Exercises

1. a) Write a method `randomNumber` with no parameters that returns a random `int` value between 1 and 6.

   b) Copy and modify your method `randomNumber` to have one int parameter, `high`, that returns a random integer value between 1 and `high`.

   c) Copy and modify your method randomNumber to have two `int` parameters, `min` and `max`, that returns a random integer value between `min` and `max`.

   d) Copy and modify your method randomNumber to have three int parameters, `min` `max`, and `step`, that returns a random integer value between `min` and `max` in increments of `step`.

   For example, a call to `randomNumber(10, 25, 5)` could return the values `10`, `15`, `20`, or `25`.

2. Consider this method then answer the questions that follow.

```
public static int mystery ( double a, double b)
{
      int value = 0;
      if (a < b)
      {
         value = -1;
      }
      if (a > b)
      {
         value = 1;
      }
      return value;
}
```

   a) What is the identifier of the method?

   b) What are its parameters?

   c) What type of value is returned by the method?

   d) Rewrite the method using a nested `if` structure.

   e) Rewrite the method using multiple `return` statements.

3. Assuming that the method `f` has been defined as it was in Example 1, state, with reasons, which of the following statements is invalid.

   a) `System.out.println(f(-7));`

   b) `double x = f(-7);`

   c) `double x = System.out.println(f(-7));`

   d) `double x = -7;  f(x);`

4. Write a method `largest` that returns the value of the largest of its three `double` parameters.

5. Write a method `gcd` that returns the value of the greatest common divisor of its two `int` parameters.

6. Write a boolean method `isDivisible` with two int parameters. The method should return `true` if and only if the first parameter value is exactly divisible by the second.

7. Write a boolean method `isPrime` that returns `true` if and only if its `int` parameter is a prime number.
8. Write a boolean method `isLetter` that returns `true` if and only if its single `char` parameter is a letter of the alphabet (either upper or lower case).

Solutions

1. a) Write a method `randomNumber` with no parameters that returns a random `int` value between 1 and 6.

```
public static int genRandNumber()
{
  int number;
  number = (int)(6 * Math.random() ) + 1;
  return number;
}
```

b) Copy and modify your method `randomNumber` to have one int parameter, `high`, that returns a random integer value between 1 and `high`.

```
public static int genRandNumber(int high)
{
  int number;
  number = (int)(high * Math.random() ) + 1;
  return number;
}
```

c) Copy and modify your method randomNumber to have two `int` parameters, `min` and `max`, that returns a random integer value between `min` and `max`.

```
public static int genRandNumber(int min, int max)
{
  int number;
  number = (int)((max - min + 1) * Math.random() ) + min;
  return number;
}
```

d) Copy and modify your method randomNumber to have three int parameters, `min` `max`, and `step`, that returns a random integer value between `min` and `max` in increments of `step`.

For example, a call to `randomNumber(10, 25, 5)` could return the values `10`, `15`, `20`, or `25`.

```
public static int genRandNumber(int min, int max, int step)
{
  // determine how many possible choices we have
  int numChoices = ((max - min) / step) + 1;

  int number;
  number = step*((int)(numChoices * Math.random() )) + min;
  return number;
}
```

2. Consider this method then answer the questions that follow.

```
public static int mystery ( double a, double b)
{
      int value = 0;
      if (a < b)
      {
         value = -1;
      }
      if (a > b)
      {
         value = 1;
      }
      return value;
}
```

a) What is the identifier of the method?

The identifier, or name, of the method is `mystery`.

b) What are its parameters?

The parameters are `a` and `b`, which are both of type `double`.

c) What type of value is returned by the method?

The method returns an integer (`int`) value.

d) Rewrite the method using a nested `if` structure.

```
public static int mystery ( double a, double b)
{
  int value = 0;
  if (a < b)
  {
    value = -1;
  }
  else
  {
    if (a > b)
    {
      value = 1;
    }
  }
  return value;
}
```

e) Rewrite the method using multiple `return` statements.

```
public static int mystery ( double a, double b)
{
  if (a < b)
  {
    return -1;
  }
  else
  {
    if (a > b)
    {
      return 1;
    }
    else
    {
      return 0;
    }
  }
}
```

3. Assuming that the method `f` has been defined as it was in Example 1, state, with reasons, which of the following statements is invalid.

a) `System.out.println(f(-7));`
This is valid.

b) `double x = f(-7);`
This is valid.

c) `double x = System.out.println(f(-7));`
This is invalid, since the `println` statement does not return a value to be stored in x.

d) `double x = -7;  f(x);`
This is valid.

4. Write a method `largest` that returns the value of the largest of its three `double` parameters.

```
public static double largest(double a, double b, double c)
{
  double largest;
  if (a > b && a > c)
    largest = a;
  else if (b > c)
    largest = b;
  else
    largest = c;

  return largest;
}
```

5. Write a method `gcd` that returns the value of the greatest common divisor of its two `int` parameters.

   Note: This solution uses a simple approach, but it is not particularly efficient.

```
public static int gcd(int a, int b)
{
  // the greatest common divisor is the largest number
  // that will divide evenly into both numbers (i.e.,
  // there is no remainder) - use modulo for remainder
  int gcd = 0;
  for (int i = 1; i <= a; i++)
  {
    if (a % i == 0 && b % i == 0)
    {
      gcd = i;
    }
  }
  return gcd;
}
```

6. Write a boolean method `isDivisible` with two int parameters. The method should return `true` if and only if the first parameter value is exactly divisible by the second.

```
public static boolean isDivisible(int a, int b)
{
  if (a % b == 0)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

7. Write a boolean method `isPrime` that returns `true` if and only if its `int` parameter is a prime number.

```
public static boolean isPrime(int number)
{
  // a prime number is divisible only by 1 and itself
  // don't need to test these numbers
  for (int i = 2; i < number; i++)
  {
    // use the isDivisible method from previous exercise
    if (isDivisible(number, i))
    {
      // a single divisible number means we're done
      // so return a false right away
      return false;
    }
  }
  // if we made it here, no divisible values were found
  // so the number must be prime
  return true;
}
```

8. Write a boolean method `isLetter` that returns `true` if and only if its single `char` parameter is a letter of the alphabet (either upper or lower case).

```
public static boolean isLetter(char letter)
{
  if (('A' <= letter && letter <= 'Z') ||
      ('a' <= letter && letter <= 'z'))
  {
    return true;
  }
  else
  {
    return false;
  }
}
```