An Introduction to Using Methods

Methods

Every Java application has at least one class that has a method named main. If you execute a Java class the JVM automatically looks for a method named main and passes control to the first statement in main.

In practice most classes comprise many methods. By using methods you can write modular code where an algorithm is broken down into its component parts. As an example of the modular approach we use the Sieve of Eratosthenes – an algorithm for finding prime numbers that you might have been introduced to in school. The sieve is discussed below and coded in Listing 1.

Example - Sieve of Eratosthenes

The result of applying the Sieve is a list of *prime* numbers. Recall: a prime number is an integer greater than 1 that has no positive divisors other than 1 and itself. For instance, 2, 3, 5, 7, 11, and 13 are the first 6 primes.

Context: To find prime numbers between 2 and some known limit. A list of all integers i ranging from 2 to the limit is created such that $list_i$ is i. The algorithm *crosses off* multiples of prime numbers and when the algorithm completes the integers that have not been crossed off are primes. Crossing-off multiples begins with 2; then the next non-crossed off integer in the list has its multiples crossed off, and so on. Crossing off an integer is done by setting the entry in the list corresponding to the integer to 0.

Sieve of Eratosthenes:

1. next ← 2 // begin crossing-off multiples starting with 2
2. for next ranging from 2 to limit in increments of 1
   a. if $list_{next}$ is not crossed-off // cross-off all multiples of next
      i. p ← 2*next
      ii. while p <= n
         1. $list_p$ ← 0 // cross-off this multiple
         2. p ← p+next // next multiple

Let's look at applying the algorithm for n=25

The algorithm begins with (note the list includes 0 and 1 in order that $list_i$ = i):

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

When step 2 executes for the first time (next is 2) the list becomes

```
0 1 2 3 0 5 0 7 0 9  0 11  0 13  0 15  0 17  0 19  0 21  0 23  0 25
```

where all the even numbers larger than 2 are replaced by 0.

When step 2 executes again (next is 3) the list becomes

```
0 1 2 3 0 5 0 7 0 0  0 11  0 13  0  0  0 17  0 19  0  0  0 23  0 25
```

where all the numbers that are multiples of 3 are replaced by 0.

This process continues and the list eventually becomes:

```
0 1 2 3 0 5 0 7 0 0  0 11  0 13  0  0  0 17  0 19  0  0  0 23  0  0
```

From the above, starting at 2, we easily see primes less than or equal to 25 are:

```
2 3 5 7 11 13 17 19 23
```

Listing 1: Sieve of Eratosthenes

```
import javax.swing.JOptionPane;
import java.util.ArrayList;

public class Sieve1
{
    public static void main(String[] args){
        // get limit from user
        String limitAsString = JOptionPane.showInputDialog("Enter upper limit:");
        int limit = Integer.parseInt(limitAsString);
        // set up the list of integers
        ArrayList<Integer> list = new ArrayList<>();
        for (int i=0; i<=limit; i++){
            list.add(i);
        }
        // apply the sieve technique
        for (int next=2; next<limit; next++){
            if (list.get(next)!=0){
                // cross off multiple of next
                int p = 2*next;
                while (p <= limit){
                    // cross off this element
                    list.set(p, 0);
                    p+=next;
                }
            }
        }
        // display primes
        ArrayList<Integer> result = new ArrayList<>();
        for (int i=2; i<limit; i++){
            if (list.get(i)!=0) result.add(i);
        }
        JOptionPane.showMessageDialog(null,"primes < "+limit+" are "+result);
    }
}
```

In Listing 1 you should note:

1. As always we have coded the algorithm using a single method named main.
2. We have used JOptionPane to obtain the limit for the sieve from the user.
3. The ArrayList named `list` is initialized so that it has elements starting at `0`. This is done so the element at the i^{th} position is `i`.

Using Methods

A programmer will often use many methods when coding an algorithm. This is done to improve readability and to provide reusable pieces of code. We will, over the next two examples, re-write the code to better indicate the overall structure of the program:

1. get limit from user
2. apply the sieve
3. display the results

A book we recommend to Java programmers is Clean Code: A Handbook of Agile Software Craftsmanship by Robert C Martin[1]. As an example for using methods to create a self-documenting and readable program the book uses the Sieve of Eratosthenes. In that example Robert Martin codes the algorithm using 8 methods. To illustrate the use of methods here we will employ 3 methods.

Using a value-returning method

A value-returning method is one that returns a value to the point where the method was invoked. Let us begin by placing the code for getting the limit from the user in a method. For this we will use a *value-returning* method that begins:

```
public static int getLimitFromUser(){
```

There are two things you should note about the above method header:

1. Instead of `void` this method specifies `int`. In this way we are stating the method will return an `int` value.
2. The method is named `getLimitFromUser`.

To illustrate the use of this value-returning method we have re-coded the sieve as shown in Listing 2. In this listing you should note:

1. There are two methods, one named `main` and the other named `getLimitFromUser`.
2. The method `getLimitFromUser` contains the code to prompt the user for the limit.
3. The last statement in `getLimitFromUser` is a *return* statement – this return is executed when `getLimitFromUser` ends causing a value to be returned to the point where the method was invoked.
4. The first statement in `main` is
   ```
   int limit = getLimitFromUser();
   ```
   This statement calls `getLimitFromUser` which executes and when it completes it returns an `int` value which is assigned to `limit`.

[1] Robert C. Martin; *Clean Code: A Handbook of Agile Software Craftsmanship*; Prentice Hall; 2008; ISBN-13: 978-0132350884.

Listing 2: **Sieve of Eratosthenes with value-returning method getLimitFromUser**

```
import javax.swing.JOptionPane;
import java.util.ArrayList;

public class Sieve2
{
    public static void main(String[] args){
        int limit = getLimitFromUser();
        // set up the list of integers
        ArrayList<Integer> list = new ArrayList<>();
        for (int i=0; i<=limit; i++){
            list.add(i);
        }
        // apply the sieve technique
        for (int next=2; next<limit; next++){
            if (list.get(next)!=0){
                // cross off multiple of next
                int p = 2*next;
                while (p <= limit){
                    // cross off this element
                    list.set(p, 0);
                    p+=next;
                }
            }
        }
        // display primes
        ArrayList<Integer> result = new ArrayList<>();
        for (int i=2; i<limit; i++){
            if (list.get(i)!=0) result.add(i);
        }
        JOptionPane.showMessageDialog(null,"primes < "+limit+" are "+result);
    }
    public static int getLimitFromUser(){
        String limitAsString = JOptionPane.showInputDialog("Enter upper limit:");
        int number = Integer.parseInt(limitAsString);
        return number;
    }
}
```

We say that Java has two types of methods:

- `void` methods that do not return a value.
- Value-returning methods have their type declared in the method header. The return statement must specify a value of the type defined in the header.

Parameters

A method can be defined with or without parameters. We have seen, but never made use of, the parameter specified for a main method. By convention the parameter for a main method is an array of `Strings` – such arrays are covered much later in these notes.

In our next version of the Sieve we have coded two more methods that begin with the headers:

```
public static ArrayList<Integer> applySieve(int upperlimit){
public static void displayResults(ArrayList<Integer> listOfPrimes){
```

The above headers indicate:

1. The method `applySieve` has one parameter named `upperlimit` of type `int`. When `applySieve` is called we say the calling method must pass in an `int` value.
2. The method `applySieve` declares its type to be `ArrayList<Integer>` and so it must return a value which is an `ArrayList` of `Integers`.
3. The method `displayResults` has one parameter named `listOfPrimes` which is an `ArrayList` of `Integers`. When `displayResults` is called we say the calling method must pass in an `ArrayList` of `Integers`.
4. The method header for `displayResults` uses the keyword `void` and so `displayResults` must not return a value.

In Listing 3 you will see a class that has 4 methods. The `main` method is very simple:

```
public static void main(String[] args){
    int  limit = getLimitFromUser();
    ArrayList<Integer> result = applySieve(limit);
    displayResults(result);
}
```

One can say it clearly states what it does: get a limit from a user, apply the sieve to get a result and then display that result. Note each statement in `main` invokes another method.

In Listing 3 you should observe:

1. `getLimitFromUser` is the same as in Listing 2. This method does one simple thing: it interacts with the user to get a limit for the Sieve of Eratosthenes.
2. `displayResults` is a void method, and so it does not return any value.
3. `displayResults` has a parameter that is used by a calling method to pass an `ArrayList` into the method. The method uses the name `listOfPrimes` to refer to that ArrayList. `displayResults` does one simple thing: it displays the list of primes.
4. `applySieve` is value-returning. The header specifies the type that it returns to a calling method. The last statement in the method returns the `ArrayList` containing the list of primes.
5. `applySieve` has a parameter that is used by a calling method to pass an `int` into the method. This method uses the name `upperLimit` to refer to that `int` value.

The hope with Listing 3 is that, overall, the program is easier to understand. Someone reading this code should easily grasp its structure and order of execution. Beginning at main, the reader knows the other methods are called in sequence. The main method makes it clear how the program is organized.

Listing 3: Sieve of Eratosthenes using 3 methods in addition to main.

```
import javax.swing.JOptionPane;
import java.util.ArrayList;

public class Sieve3
{
    public static void main(String[] args){
        int  limit = getLimitFromUser();
        ArrayList<Integer> result = applySieve(limit);
        displayResults(result);
    }

    public static void displayResults(ArrayList<Integer> listOfPrimes){
        JOptionPane.showMessageDialog(null,"list of primes "+listOfPrimes);
    }

    public static int getLimitFromUser(){
        String limitAsString = JOptionPane.showInputDialog("Enter upper limit:");
        int number = Integer.parseInt(limitAsString);
        return number;
    }

    public static ArrayList<Integer> applySieve(int upperlimit){
        // set up the list of integers
        ArrayList<Integer> list = new ArrayList<>();
        for (int i=0; i<=upperlimit; i++){
            list.add(i);
        }
        // apply the sieve technique
        for (int next=2; next<upperlimit; next++){
            if (list.get(next)!=0){
                // cross off multiple of next
                int p = 2*next;
                while (p <= upperlimit){
                    // cross off this element
                    list.set(p, 0);
                    p+=next;
                }
            }
        }
        // the list of primes to return
        ArrayList<Integer> result = new ArrayList<>();
        for (int i=2; i<upperlimit; i++){
            if (list.get(i)!=0) result.add(i);
        }
        return result;
    }
}
```

Summary

This chapter is an introduction to using methods in Java applications. The chapter on designing classes will complete this topic. Do note that as you have been learning Java you have been introduced to many methods that have been developed by programmers that have provided our Java framework. For example, the `Math` method `max` is a value-returning method with two parameters. To use max we write code such as `Math.max(i,j)` where `i` and `j` are passed in to its parameters. The `Random` class has a value-returning method used to get a random value. To use `nextInt` we write code such as `g.nextInt(6)` where the value `6` is passed in to its parameter. The System class has a method `println` that has one parameter of type `String`; `println` does not return a value.

Methods are used by programmers to produce readable code. Methods can be used to break complex software into its component parts. Well designed components can be read and understood without having to know everything about the software as a whole. Robert Martin`s purpose in writing *Clean Code* is to help its readers become better programmers. Each method must be given a name that indicates exactly the purpose of the method. To some programmers this way of naming methods along with good names for variables replaces the need for most comments.

Methods can be categorized as value-returning or `void`. A value-returning method declares its type in the header and must have a `return` statement that returns a value of the requisite type.

Methods may be designed with no parameters, or with parameters of specific types. A method with more than one parameter separates their declarations with commas. For instance:

```
public static void myMethod (int a, double x, String myName){
```

When the above method is called the calling method must include three values in the order indicated; for example:

```
myMethod (25, 26.3, "George");
```

Exercises

1. Consider the Sieve examples in this chapter. Write another version that uses methods to break up the complexity of `applySieve`.

2. Consider Example 2 in the Character section of Chapter 4. In this example a scanner object is used to get a line from the user, then the characters are examined one-by-one, and then the sum of the numeric value of those characters is displayed. Rewrite the program to use the main method:

```
public static void main(String[] args){
    String line = getLineFromUser();
    int s = analyzeLine( line );
    displaySum( s );
}
```

3. Consider Example 3 in the Character section of Chapter 4. In this example the user is prompted for a student number, the student number is analyzed for validity and a message "valid" or "invalid" is displayed.

```
public static void main(String[] args){
    String number = getStudentNumber();
    boolean valid = analyzeNumber( number );
    displayValidity( valid );
}
```