2 Data Declaration

In our look at the three programs HelloWorld, EchoArgs, and Gauss in the first lesson, we encountered several kinds of data. We begin the second lesson by reviewing the process of declaring the variables used by a program.

2.1 Primitive Data Types

In order to store information to use in its computations, a program must set aside space to save the information and, at the same time, identify how the information is to be interpreted. In Java, as in many other computer languages, named variables are used to store data. In order to use a variable, a program must first declare each variable, that is, identify the name of the variable and the kind of information to be held there. The declaration of variables in Java has the form:

```
identifier name;
```

Here the identifier indicates what kind of data is to be stored and the name is used to refer to the data later in the program. You can declare several variables in one line by including several names (separated by commas):

```
identifier name1, name2, name3;
```

Thus, for example, the statement

```
int sally, fred;
```

announces that the program will use two integer variables with the names sally and fred.

The simplest data can be described by one of Java's primitive data types. We have already seen five primitive data types that can serve as identifiers in a declaration.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>An integer between $-2^{31}$ and $2^{31} - 1$.</td>
</tr>
<tr>
<td>long</td>
<td>An integer between $-2^{63}$ and $2^{63} - 1$.</td>
</tr>
<tr>
<td>double</td>
<td>A floating point (real) number roughly between $-1.7 \times 10^{308}$ and $1.7 \times 10^{308}$, stored with roughly 14 significant digits.</td>
</tr>
<tr>
<td>boolean</td>
<td>A truth value: true or false.</td>
</tr>
<tr>
<td>char</td>
<td>A 16 bit unicode character.</td>
</tr>
</tbody>
</table>
When a variable is declared with a primitive data type, not only is the name and interpretation of the data decided, but the physical space in which the data is to stored is identified. Think of it this way: the declaration `int sam` sets aside enough space in computer memory to save one integer and labels that location with the name `sam`, so you know where to find the data when you need it.

The `unicode` character set includes not only the standard western alphabet, but also cyrillic, chinese, arabic, sanscrit, hebrew, and almost every other conceivable alphabet. This makes it possible to write Java programs in Russian or Chinese!

Any variable can be initialized to hold a value at the same time it is declared. Thus, for example, you can write:

```java
int i = 0;
long sum = 123;
double pi = 3.141592653;
boolean ILikeMike = true;
char initial = 'W';
```

Each of the primitive data types has an associated `Class` that has some `methods` associated with that type. Thus, for example, the `Class Integer` is associated with the data type `int` and provides several useful `methods`. Here are a few examples:

```java
Integer.MIN_VALUE . . is -2147483648, the smallest int
Integer.MAX_VALUE . . is 2147483647, the largest int
Integer.parseInt("135"); . . turns the String "135" into an int
Integer.toString(123); . . . turns the Integer 123 into a String
```

Notice that the `methods` of a `Class` are invoked by specifying the `Class` name (or the name of an object from that class) followed by a period and the name of the method.

It is common to use the `Integer` method `parseInt` to identify an argument to a program. If, for example you wanted to change the program `Gauss` from the first laboratory assignment to add up the integers from 1 to a number specified via an argument, you might write:
/**
   06/03/2020 Walter Carlip Program "Gauss" to add the
   first n integers: n specified as the "zeroth" argument.
*/
public class Gauss {
   public static void main(String[] args) {
      int sum = 0;
      for(int i = 0; i <= Integer.parseInt(args[0]); i++){
         sum += i;
         System.out.println(sum);
      }
   }
}

Notice that the 100 in the first version of Gauss has been changed to

   Integer.parseInt(args[0])

The String that is typed in as the argument must be converted to an
Integer in order to compare it to the int i. Similarly, a program that adds
two integers might look for the integers in the programs argument string and
add them:

   x = Integer.parseInt(args[0]);
   y = Integer.parseInt(args[1]);
   z = x + y;

2.2 Arrays

Each of the primitive datatypes can be combined to form arrays, or lists
of elements. An array is declared by putting square brackets ([ ]) after the
identifier. The general format for defining an array is:

   identifier[] name;

Thus, an array of integers named bill would be declared by:

   int[] bill;
There is one tricky aspect to declaring arrays: declaring the array does not actually allocate the space. Why not? The reason is simple: you have not specified how many items are in the list. A list of five integers takes up less space than a list of one hundred integers. You can allocate the space for storage by using the command new. Thus, the command

```java
int[] bill = new int[5];
```
makes the variable bill point to an array of 5 integers or, more accurately, to sufficient space to store 5 integers. You can also initialize an array along with the declaration by listing the values it is to hold link this:

```java
int[] bill = new int[] {7, 11 13, 92, 101};
```
A short-hand way to write this that works in some situations is:

```java
int[] bill = {7, 11 13, 92, 101};
```

You reference the individual elements in an array by their index. The first element has index 0 (e.g., bill[0], the second has index 1 (e.g., bill[1], etc. You can also identify the total number of items in the list, i.e., the length of the array. Thus, for example, bill.length will be equal to 5, the number of integers in the array we declared above. Of course, the last element of the array is bill[bill.length-1]. (Why length-1 for the last one? Because 0 is the index for the first one.)

### 2.3 Strings

In lesson one we also introduced the class String. A String is simply a list of characters. You can define a variable to hold a string by declaring it:

```java
String harry;
```
You can initialize a String by writing the characters that define the String between double quotes:

```java
String harry = "My name is Harry!";
```
If you were to execute the command `System.out.write(harry)`, after declaring and initializing the String variable `harry` as above, the sentence “My name is Harry!” would be typed out on your terminal screen.

Since String is the name of a Class, it can have data (fields) and actions (methods). In fact, there are several useful methods in the String class. For example, you can convert other types to String:
int i = 125;
String numeral = String.valueOf(i);

will take the number 125 stored in the integer variable i and convert it to the String of characters 125.

You can also identify specific characters within a string and obtain it’s length. Here are some examples:

String MyName = "George Washington";
MyName.length() ....... represents the integer 17
MyName.charAt(4) ....... represents the character g
MyName.endsWith("ton") .... represents the boolean true

As with the Class Integer discussed above, the methods (e.g., length(), charAt()) are invoked by following the String with a period and the method name.

### 2.4 Operations

In lesson one we introduced several operations that could be performed with integers, including ++ (increment by one) and += (add to). There are quite a few more. Here are some, with their meaning:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Increase by one</td>
</tr>
<tr>
<td>--</td>
<td>Decrease by one</td>
</tr>
<tr>
<td>+, -, *</td>
<td>Respectively: add, subtract, multiply, two numbers</td>
</tr>
<tr>
<td>/, %</td>
<td>Respectively: quotient and remainder when dividing integers.</td>
</tr>
<tr>
<td>+=, -=, *=, /=, %=</td>
<td>Operate and replace with result.</td>
</tr>
<tr>
<td>&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td>==, !=</td>
<td>Equal to, unequal to (result is a boolean value).</td>
</tr>
<tr>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td>Less than, greater than, less than or equal to, greater than or equal to (result is a boolean value).</td>
</tr>
</tbody>
</table>

There are also a number of methods that belong to the Class Math that perform useful computations. Some examples are:

Math.sin(double) Math.tan(double) Math.random()
Math.sqrt(double) Math.max(double,double)

The last of these, Math.random() returns a pseudorandom number between 0 and 1. The Class Math is one of many built in libraries of methods in Java.
2.5 Example: An Array of Fibonacci Numbers

The Fibonacci numbers are an interesting sequence of numbers that occur in interesting ways in nature. The Fibonacci sequence begins with the integers 0 and 1, and each successive number is the sum of the two previous. Here’s a program that computes an array of Fibonacci numbers. Notice that this class has two methods, the main method and the Fibonacci method.

```java
/**
 * 06/03/2020 Walter Carlip Compute an array of Fibonacci Numbers
 */

public class FibonacciArray {
    public static int[] Fibonacci(int n) {
        int[] MyFibonacci = new int[n];
        MyFibonacci[0] = 0;
        MyFibonacci[1] = 1;
        for (int i = 2; i < MyFibonacci.length; i++) {
            MyFibonacci[i] = MyFibonacci[i - 1] + MyFibonacci[i - 2];
        }
        return (MyFibonacci);
    }

    public static void main(String[] args) {
        int[] firsttest, secondtest;
        firsttest = Fibonacci(10);
        secondtest = Fibonacci(50);
        for (int i = 0; i < firsttest.length; i++) {
            System.out.print(firsttest[i] + " ");
        }
        System.out.println();
        for (int i = 0; i < secondtest.length; i++) {
            System.out.print(secondtest[i] + " ");
        }
    }
}
```

Notice that most of the work is done in the Fibonacci method. The main method is only there to demonstrate and test the workings of the Fibonacci method.
2.6 Example: Division Algorithm

The division algorithm is an extremely important algorithm in mathematics: it justifies the concept of “quotient” and “remainder”. Here are two short programs that implement the division algorithm. They are essentially the same, with small differences in the writing style.

```java
/**
 * 06/03/2020 Walter Carlip Apply Division Algorithm to two integers
 */
public class DivAlg {
    public static void main(String[] args){
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int r = a;
        int q = 0;
        for( ; b <= r; ){
            System.out.print(r + " ");
            r = r - b;
            q = q + 1;
        }
        System.out.println();
        System.out.println("Quotient: "+ q);
        System.out.println("Remainder: "+ r);
    }
}
/**
 * 06/03/2020 Walter Carlip Apply Division Algorithm to two integers
 */
public class DivAlg {
    public static void main(String[] args){
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int r, q;
        for( r = a, q = 0; b <= r; r -= b, q++ ){
            System.out.print(r + " ");
        }
        System.out.println();
        System.out.println("Quotient: "+ q);
        System.out.println("Remainder: "+ r);
    }
}
```
Summary

- Java programs save data by declaring and initializing variables, and then changing the data with operations.

- There are five important primitive data types `int`, `long`, `double`, `boolean`, `char` and these have associated Classes which have methods that are useful for manipulating data.

- Primitive data can be arranged in lists called arrays.

- Lists of characters belong to the Class String, which has several useful methods.

- There are quite a few operations that can be used with primitive data types to perform computations and other more sophisticated operations that occur as methods in the Class Math.