Introduction to Computing with Java

Walter Carlip

June, 2020

These are notes from a course offered during the summer of 2020 as part of the University of Chicago Young Scholars Program. Copyright © 2020 by Walter Carlip and the University of Chicago Young Scholars Program.
Lesson 1: Introduction

In this course we will study programming in the computer language Java. In this day and age, when most programs that we need have already been written, you may wonder “Why study programming?” Of course, one reason is to achieve basic computer literacy: no matter what use you make of computers, it helps to understand just what makes them “tick”. Even when using a commercial program it helps to have a good idea what kinds of things a program can do (and what it is unreasonable to expect a program to do), and to develop some intuition what is going wrong when things go wrong (as they surely will).

On the other hand, we have several other important objectives. As a mathematician, my main interest in computers lies in their usefulness in studying mathematics. In the course of my research I have observed many reasons to write computer programs. In this course we will be using computers to aid in our study of number theory. Number theory is the study of the properties of the integers. We will be using computers to study the integers in several ways:

- We can use computers to perform experiments with numbers and other mathematical objects. By observing these experiments we can formulate conjectures, which we can then try to prove.

- We can use computers to test the truth of possible theorems. Every mathematical theorem starts out as a conjecture. It is often worthwhile to try out a conjecture before attempting to prove it. In doing so, we may discover it is not true, or we may gain insight into why it is true.

- We can use the computer to generate examples that help us understand theorems that we have already proven. A theorem is only interesting if it really tells you something, that is, if it actually applies in concrete situations. Thus, for example, a well-known theorem asserts that there are an infinite number of primes, but most of us are only familiar with one and two digit primes. If we really want to understand primes, we need to be able to find larger and more interesting primes. In fact, the ability to find very large primes is important in the field of cryptology and has vital application in data security applications for internet commerce.

Copyright © 2019, 2020 by Walter Carlip and the University of Chicago Young Scholars Program
• We can use computers to help us understand theorems. In order to write a program to exhibit the workings of a theorem, you have to be able to explain the theorem to a computer. To do this you have to understand the theorem very well — and be able to program.

• We can use computers to perform specialized computations rapidly.

• We will also examine some ways in which number theory can be used in computer science, for example, to devise (almost) unbreakable codes.

Computers are useful in the study of integers because they are capable of performing complicated calculations at very high speed. This is useful when you want to study a large list of numbers. It would be an easy task for a computer to check whether the first 1000 integers have some property, while it would take a long time to check each one by hand. The speed of computers is also useful when you need to do one extremely complicated calculation. For example, it is easy to find by hand all the ways that the number 35 can be written as the product of two integers: \(1 \times 35\) and \(5 \times 7\). Imagine, however, trying to figure out how many ways the number 1,234,567,892,134,456,884 can be written as a product of two integers!

As we study the nitty-gritty of programming, we will focus on mathematical experimentation, and gain insight into some interesting mathematics.

1.1 Java

In order for a computer to perform a calculation for us, we must tell the computer what calculation to perform. To do this we must be able to express precisely what we want the computer to do. This is done by giving the computer a program written in a computer language.

A computer language provides a precise way of instructing the computer what to do. A program is a series of instructions, written in a computer language, that tells a computer how to perform a particular task.

We will be using the computer language Java. Java is a relatively new language, developed by James Gosling at Sun Microsystems (now owned by Oracle), and introduced in late 1995. In the intervening years it has become one of the most important modern computer languages. According to Sun’s documentation Java is a “simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance,
multithreaded, dynamic language” . . . and that’s why it’s been so highly acclaimed! That is a long list of technical terms, but they do, indeed, describe Java well.

The most important of these is that Java is portable. That means the same Java program written in our lab on Macintosh machines can run on IBM PC’s, on Sun Workstations, Linux, and on virtually any other machine. This capability is usually referred to as “Write Once, Run Anywhere”. The reason for this is, in part, that Java programs are interpreted: Java programs are executed by another program, called a Java Virtual Machine (JVM). Java programs can run on any machine that has a JVM—and, in fact, most web browsers have a built in JVM.

The second most important of these characteristics is that it is “Object-Oriented”. This involves a sophisticated way of thinking about programming that we will discuss throughout this class. In particular, the primary elements of a Java program are its “objects”, which are described by Java Classes. Objects in Java have their own properties (or data), which are called Fields and their own actions, which are called Methods. As you work with Java, keep in mind that every thing it does is a Method associated to some object or Class.

**Getting Started**

The fundamental idea of a computer program is to instruct the computer how to perform a task. Computer languages provide a formal way to tell a computer exactly what to do, and consequently, discussions of computer programming often share a lot with discussions of grammar: they are extremely dull. On the other hand, programming itself is fun; you will discover that computers really do what they are told, and it is easy to feel a sense of accomplishment when your programs succeed. As with developing language skills, it is easier to learn programming by actual programming than by memorizing grammatical rules. Consequently, we will study Java primarily by studying examples and by writing programs.

The process of creating a Java program is simple: you write your Java program using a text editor; you run a Java compiler to turn your program into universal byte code, which can be interpreted by the Java Virtual Machine; and you hand the new program to the virtual machine for interpretation.

Most years we have created and run our Java programs on machines that run the Macintosh operating system, using an Integrated Development
**Environment** called DrJava, which was developed at Rice University. Since DrJava is free and easily downloadable from the internet, you are free to do that if you wish.

We also have the option of using a separate editor (there are several available for Macintosh users), compiler (javac), and interpreter (Java) run directly from a terminal session.

This year, however, because of our “social distancing”, we will experiment with a new approach—using an online development environment available with a web browser at https://www.jdoodle.com/online-java-compiler/. (Instructions on how to proceed appear in the first lab handout.)

**Hello World**

Tradition has it that everyone’s first program in any new language should be HelloWorld, a simple program that does nothing more than write out the words “Hello World!” to the terminal screen. In Java, Hello World looks like this:

```java
/**
 06/05/2020 Walter Carlip  My First Java Program! (Not!)
*/
public class HelloWorld{
    public static void main(String[] args) {
        System.out.println( "Hello World!" );
    }
}
```

This is the simplest possible Java program. Let’s take it apart.

The first three lines are comments: their only purpose is to describe and document the program and changes to the program. Comments do not actually do anything when the program is run. However, it is important to include ample comments in your programs that explain in plain English what you intend the program to do. In Java, any text written between the symbols /* and */ are considered to be comments and are ignored by the Java interpreter. You can also put individual comments at the end of any line following the symbols // . Thus, for example, one might change the fourth line to

```java
public class HelloWorld{ // Start the HelloWorld Class here.
```
The fourth line introduces the class, **HelloWorld**, that you are writing. This is a **public** class, meaning that it could be used by other classes if it were part of a large project. The definition of the class is contained between the first open brace, {, and the last close brace, }.

The fifth line introduces a **method** for the class. The term **method** is Java’s way of saying “procedure” or “process”. A class’s **methods** define what things the class can do. The **HelloWorld** class has only one method, its **main** method. The main method of a class defines what the class should do when it is started-up by the interpreter (the Java Virtual Machine). The definition of the method is contained between the second open brace, {, and the second-to-last close brace, }.

The sixth line describes what the main method of the **HelloWorld** class actually **does**. It asks a method, **println**, which belongs to another class, **System.out**, to “print” its argument (the quoted characters within the parentheses) to the system console. (The system console is the window in which you run your Java program.) Again the main method is **public** (other classes may use it) and it has no **return value** (the return value is **void**).

The next line has a single brace indicating the end of the main method, and the last line a single brace indicating the end of the **HelloWorld** class. The body of every **class** and every **method** is always enclosed between pairs of braces, {).

**Strings and Arguments**

The **HelloWorld** program is a very simple program indeed: it merely types a fixed series of characters out to the screen. (This series of characters is the program’s **output**.) It is a boring program, because it does not respond or interact with the user in any way. In later lessons we will see how to make programs interact dynamically with users, but for now we will settle for writing a program that takes **input** (called its **arguments**) when it starts up.

The Java language includes several ways to store information within a program. At the most primitive level, a computer only sees ones and zeros. It follows that programmers must tell the computer how to **interpret** its data, and tell it how much space to allocate to save that data. In most computer languages this is done by describing the data’s **type**. Java has eight built-in **types** of data that it can store. In this course we will encounter five of these primitive types:
int Used to store relatively small integers (limited by the amount of space allocated for an integer).

long Used to store larger integers (using more space than an int).

boolean Used to store truth values (true or false).

double Used to store double precision floating point numbers with about 14 or 15 significant digits of accuracy.

char Used to store text characters.

More complicated kinds of data can be created by defining new Classes. Thus, for example, one might wish to be able to handle very, very large integers (we will see how to do this later using a class called BigInteger); or, perhaps, complex numbers or Cartesian coordinates; or geometric figures; or words, sentences, and paragraphs. An important feature of Java is it’s ability to handle complicated user-defined objects that may be designed to handle special applications.

One of the most important examples of a built-in Java class is the class String. A String is nothing more than a list of characters. In the HelloWorld program, for example, the words in quotes (“Hello World!”) form a string. The String class also has useful methods that we will use to manipulate Strings.

Another way to obtain more complicated data is to use arrays. An array is simply a list of objects of the same type. In a Java program square brackets, [ ], are used to indicate an array. Thus, for example, an array of integers would appear in a Java program as int[]. The integers in the array are identified by their index, i.e., their position in the array, starting with 0. If mylist is the array 2, 3, 5, 7, 9, 11, 13, 15, then mylist[0] is 2, mylist[1] is 3, mylist[5] is 11.

The reason for this discussion of Strings and arrays is that the first line of the main method of any class

    public static void main(String[] args) {

tells us that the input to the main program is an array of Strings with the name args.

If the main method of a Java program is invoked with the input argument This is a test of my new Java program, then the program will see an array containing nine Strings. Consider the following program:
/**
  * 06/04/2020 Walter Carlip EchoArgs: Echo back input arguments
  */
  public class EchoArgs {
      public static void main(String[] args) {
          for(int i = 0; i < args.length ; i++){
              System.out.print(args[i]);
              System.out.print(" ");
          }
          System.out.println();
      }
  }

This program has several new elements.

1. In its sixth line, it creates and uses an integer, i. The integer variable i is created in the sixth line of the program and given the value 0. The equals sign in the statement i = 0, is the assignment operator; it instructs the machine to give the variable i the value 0. (Think of it this way: space for an integer is allocated and given the name i when you write int i; when you say i = 0, the value 0 is placed in that location.)

2. There is a comparison performed: i < args.length. This comparison has a boolean value: true or false depending on whether the number stored in location i is less than the length of the argument array. Of course, args.length contains the length of the argument array, i.e., the number of words passed to the program in the variable args.

3. There is also a mysterious statement i++. This is Java’s way of increasing the value saved in the variable i by one. Thus, if the location i contains the integer 17, after the command i++ is performed it will contain the integer 18. There are similar operations i-- and subtle variations ++i and --i, which we may encounter later. In the object-oriented way of thinking, the operation ++ is telling the object i to increase itself by one.

These three elements are contained in the argument of a for loop. The for loop causes the statements within braces following it to be repeated many times. The three statements in parentheses following for indicate:
• The step to be taken before the loop begins (set the index counter \( i \) equal to zero).

• The test to see if the looping is done (continue as long as \( i \) is less than the number of input words).

• Any steps required to be taken before repeating the loop (increment \( i \) to the next index).

The statements between the braces following the \texttt{for} statement contain the \texttt{block} of statements to be executed each time around. Thus, each time around the \( i \)-th argument is printed.

\textbf{Gauss’s Trick}

We will complete this lesson by looking at a program that adds the first 100 numbers: a feat (allegedly) performed in his head by the mathematician Gauss while he was in elementary school.

\begin{verbatim}
/**
   06/08/2020 Walter Carlip  Program "Gauss" to add the first hundred integers.
*/
public class Gauss {
    public static void main(String[] args) {
        int sum = 0;
        for(int i = 0; i <= 100; i++){
            sum += i;
            System.out.println(sum);
        }
    }
}
\end{verbatim}

The only new elements here are the declaration and initialization of the integer variable \texttt{sum} in the third line; the new comparison <= within the \texttt{for} loop; and the new assignment operator \texttt{+=}. Here \texttt{sum += i} means “replace \texttt{sum} by its old value plus the value stored in the variable \texttt{i}”. Thus \texttt{+=} combines addition (+) with assignment (=). This line could just as easily be written \texttt{sum = sum + i}.
Summary

- **Java** programs are made up of **Classes** that contain **Methods** (what the class does) and **Fields** (the class’s data).

- When a class is run, execution begins with its **main** method, which always begins: `public static void main(String[] args)`.

- Data can be stored in **primitive variables** with types `int`, `long`, `double`, `boolean`, or `char` (or a few others we have not yet seen).

- Primitive data can be put into lists called **arrays** that are indexed starting with their zeroth term.

- The class **String** describes lists of characters.

- The input to the **main** method is an array of **Strings**.

- The **for** statement can be used to repeat statements in a loop.

- The operations `< <= > >= ==` are used for comparison of numbers and result in a **boolean** result (true or false).

- The operations `= +=` are assignment operators: the first simply assigns the value on the right to the variable on the left, the second **adds** the value on the right to the variable on the left.

- The system command `System.out.println` prints its argument String on the terminal screen.