

AP Computer Science Manual

Brandon Waller | brandonw@live.com

August 4, 2017

Contents

1 Purpose	2
2 Introduction	2
2.1 Running Java Programs	2
2.1.1 BlueJ	2
2.1.2 Linux/Unix Terminal	2
2.1.3 The .java file	3
3 Java Basics	3
3.1 Introduction	3
3.1.1 bits	3
3.2 Data Types	4
3.2.1 Primitives	4
3.2.2 Declaring and Converting Primitives	4
3.2.3 Objects	5
3.2.4 Declaring Objects	5
3.2.5 Arrays and the ArrayList	5
3.3 Operations	6
3.3.1 System.out.print(String)	6
3.3.2 Mathematical Operations	7
3.3.3 Modulo	8
3.4 Control Structure	8
3.4.1 Boolean Operations	8
3.4.2 If, If-Else, Else	9
3.4.3 Loops	10
3.4.4 Nested Loops	11
3.5 Questions	12
4 Classes and Objects	13
4.1 Why use objects?	14

1 Purpose

Reading and understanding should guarantee at least a 4 on the ap exam and a 90% chance of getting a 5. This is also an excuse for me to practice \LaTeX .

2 Introduction

to score well on the AP exam, you will need to have a grasp of the basics of Java, a programming language that is rarely used any more, but the college board for some reason won't switch to Python.

2.1 Running Java Programs

note: Skip this section until you are ready to run Java programs

To run Java programs, you will either need to install a Java IDE (Integrated Development Environment) on PC/Linux/Mac , or use a terminal compiler on Linux or Mac.

2.1.1 BlueJ

When I took the class, we all used BlueJ, a free Java IDE. You can download it from this [Link](#). you may also need to download a [JDK](#) (Java Development Kit)

2.1.2 Linux/Unix Terminal

Run the following commands in the Terminal to run and compile .Java files:

- 1: sudo apt-get install openjdk-8-jre
- 2: sudo apt-get install default-jdk

To run and compile a file with the name **helloworld.java**:

- 1: javac helloworld.java
- 2: java helloworld

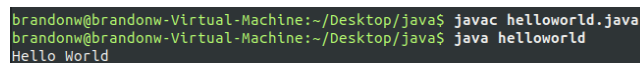
A terminal window with a dark background. The prompt is 'brandonw@brandonw-Virtual-Machine:~/Desktop/java\$'. The first command is 'javac helloworld.java'. The second command is 'java helloworld'. The output is 'Hello World'.

Figure 1: Running a .java file from the terminal

Additionally, you may want to learn how to use a terminal text editor such as vim or emacs to edit files.

```

1 import java.util.*;
2
3 public class helloworld
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("Hello World");
8     }

```

Figure 2: Hello World Java program

2.1.3 The .java file

Java programs are written in a .java file that is interpreted by the java compiler to execute the instructions. Features of this file

- 1: line 1: importing java.util.*, a file that include additional Java functions
- 2: line 3: class declaration, public class followed by the file name
- 3: line 5: The main method. The set of instructions that are ran when by the executable after compiling
- 4: line 7: Java functions

There are many more things that go into .java files. These are just a few.

3 Java Basics

3.1 Introduction

Java is an object-oriented programming language developed by Sun Microsystems in 1995, and is the main language for Android app development. This section will cover the basics of Java including data types, operations, loops, and control structures.

3.1.1 bits

In computers, data is stored in bits that can either have a value of 1 or 0. Each data type is defined by the number of bits consumed by the data type, and by the way the bits are interpreted by the compiler. For example, the number 23 in 8-bit binary is 00010111 and the number -23 is stored as 11101001. This will be discussed later, but when converted to decimal 11101001 is 233. Depending on the data type, the compiler will either decide to make that number -23 or 233.

3.2 Data Types

Data types are the classifications given to variables/data. Unlike other languages such as Python and MATLAB, java requires the programmer to explicitly state the data type for a variable, making this section more important to know when programming in Java.

3.2.1 Primitives

Primitives are the basic Java data types used to store variables in memory. They are as follows:

- 1: byte: 8-bit signed data type (value $[-127, 127]$) used for applications where memory space is limited
- 2: int: 32-bit integer value that can positive and negative value
- 3: float: 32-bit decimal value
- 4: double: 64-bit decimal value, can hold more data than a float
- 5: char: 16-bit single character
- 6: boolean: stores either true or false

3.2.2 Declaring and Converting Primitives

Declaring a primitive in Java:

```
int x = 3;
byte y = 127;
double z = -12.34567;
boolean bool = true;
char a = 65;
char b = 'A';
//both a and b have the same value. Each character is
//mapped to a number
```

The left side of the = sign is the data type followed by the variable's name. On the right, is the value of the data type. Sometimes, you will want to convert one primitive to another, for example rounding a double to whole number. This is how you would do it. This uses if statements which will be discussed later:

```
double x = 2.6;
int x_whole = (int) a; //converts a to an int by removing the portion
                        //after the decimal point
if(x - x_whole >= 0.5) //if the decimal is greater than or equal to
    0.5
    x = x_whole + 1; //round up by removing the decimal and adding 1
else
    x = (int) x; //rounding down, simply remove the decimal
```

3.2.3 Objects

Objects are other data types that are either defined by the Java language, or can be created by the user (new primitive types cannot be created by the user). Additionally, unlike primitives, functions can be called on objects, potentially modifying their values.

Some common objects defined by the Java language are

- 1: String: A group of characters
- 2: Integer
- 3: Double

note: Objects start with capital letters while primitives start with lower case

3.2.4 Declaring Objects

The syntax is slightly different for the declaration of objects

```
Integer j = 20;
String str = "the is the text for the string";
String s = new String();
s = "this is the text for String s";
```

The syntax for creating an instance for a user defined object will be a little different One of the defining features that distinguishes primitives from objects is that you can call functions on objects:

```
String s = "abcd";
System.out.println(s.length()); //output = 4
```

Because String is an object, you can call the function `.length()` on it.

3.2.5 Arrays and the ArrayList

Arrays are a group of variables that can be individually addressed and have a set size. They are useful for storing data of fixed length. Here is how you declare and use an array:

```
//Methods for declaring a new array
int[] i = new int[10];
int[] j = {1,2,3};
String[] s = new String[3];
String[] t = {"a", "b", "c"};
//Assigning values to an array
i[0] = 1;
i[1] = 2;
i[9] = 10;
```

```
s[0] = "abc";  
//j and t already have values and are both size 3  
//getting the length of an array  
s.length;  
i.length;  
//accessing data from an array  
i[1];  
t[2];  
j[1] + i[9];
```

note: Java, like any well written programming language starts its arrays at 0.

The ArrayList, unlike the array, does not have a define length. However, primitives cannot be stored in an ArrayList

```
//Basic syntax for creating an ArrayList: ArrayList<object type> name =  
    new ArrayList<object type>();  
ArrayList<String> strings = new ArrayList<String>();  
ArrayList<Integer> integers = new ArrayList<Integer>();  
//adding data to ArrayList  
strings.add("abc");//add "abc" to index 0  
strings.add("def");//add "def" to index 1  
strings.add("ghi");  
integers.add(1);  
integers.add(2);  
integers.add(3);  
integers.add(0,4);//add 4 to index 0, shifting each element in the  
    ArrayList forward  
//accessing the data in an ArrayList  
strings.get(0);//abc  
strings.get(2);//ghi  
integers.get(3);//3  
//removing from an array list  
strings.remove(0);//remove index 0 from the ArrayList and shift the rest  
    of the list back  
strings.get(0);//def
```

3.3 Operations

For the early stages of the class, you will mainly be staying within the **main()** function. This section will cover some of the basic operations you can do.

3.3.1 System.out.print(String)

System.out.print (and System.out.println()) are functions for displaying information to the console window. when using println(), subsequent text will be placed on a new line.

```
System.out.print("Hello");
System.out.print("World");
System.out.println("Hello ");
System.out.println("World");
```

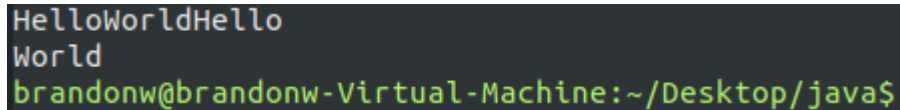


Figure 3: Console Window Output

Variables can be included in print statements. Each variable is separated by a `+`. When the variables are replaced by their value, they are included without a space.

```
String a = "a";
String b = "c";
String c = "c";
int x = 123;
System.out.println(a + b + c + x);
System.out.println("abc123");
System.out.println(a + " " + b + " " + c + x);
System.out.println("a b c 123");
```

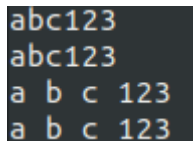


Figure 4: Console window output

Adding a `" "` between variables will put a space in the output.

3.3.2 Mathematical Operations

Java includes many mathematical operations, and many more can be added with the `java.math` package.

```
int x = 3, y = 5;
x + y; //8
x - y; //-2
x * y; //15
x / y; //0
sqrt(4); //2
sin(0); //0
```

Why is the result of x/y 0?

When two integers are divided, the result is stored as an integer. The result of dividing 3 and 5 is 0.6, which is stored as the integer 0. These are correct ways to divide integers and store the result as a double:

```
int x = 3, y = 5;
double z;
z = x/(double)y; //casting y as a double will result in a double result
z = 3/5.; z = 3/5.0;
//These implementations do not work:
z = (double)x/y; z = 3.0/5;
//The denominator must be a double for the result of division to be a
double
```

3.3.3 Modulo

The modulo is an operations similar to division, but the output is the remainder. The modulo operation is represented by the % sign. For example, $22 \% 10 = 2$ because $\frac{22}{10} = 2r2$

3.4 Control Structure

Control Structure is using boolean expressions to control if/when code is ran.

3.4.1 Boolean Operations

The boolean operations are as follows:

> : Greater than
>= : Greater than or equal to
< : Less than
<= : Less than or equal to
== : Equal to
!= : Not equal to
! : Not/Negation. Converts a boolean to the opposite value.
&& : and. Comparison of two booleans. True if and only if both conditions are true.
|| : or. Comparison of two booleans. If either is true, the or operation results in true

These operations are used to compare values and other booleans and result in a boolean. Here are some examples:

```
true == false; //false
true == true; //true
false == false; //true
```



```
1 >= 0; //true
1 > 1; //false
!true; //false
!false; //true
!(true || false); //false
!(!(true)||false) && true; //true
```

3.4.2 If, If-Else, Else

These three operations are the main three for controlling functions using boolean conditions. The basic syntax for this type of control is:

```
if(/*condition*/)
    //do something
//optional
else if(/*some other condition*/)
    //do something else
.....
else
    //If none of the conditions are true, run this code
```

In this type of statement, only one set of code will run regardless of how many are true. For example:

```
int x = 8;
if(x == 10)
    System.out.println("The value of x is 10");
else if(x % 2 == 0)
    System.out.println("x is an even number");
else if(x >= 5)
    System.out.println("x is greater than or equal to 5");
else
    System.out.println("None of the conditons are true");
```

The result of this code will be **x is an even number**. Although it is true that $x \geq 5$, that line will not be executed because the rest of the if-else statement is passed after one of the conditions is true. A way to implement this function so that all the applicable lines are executed is:

```
int x = 8;
if(x == 10)
    System.out.println("The value of x is 10");
if(x % 2 == 0)
    System.out.println("x is an even number");
if(x >= 5)
    System.out.println("x is greater than or equal to 5");
if(!(x == 10 || x % 2 == 0 || x >= 5))
    System.out.println("None of the conditons are true");
```

If the last line was kept as **else** then the code would be executed if $x < 5$ is true, even if x is even, or equal to 10.

3.4.3 Loops

Loops are a control structure that allows lines of code to be repeated many times, saving time when writing code, and allowing programs to be more flexible. There are two types of loops: **for** and **while** loops. They are both fundamentally the same, but have different syntax.

The first type of loop is the **for** loop. It consists of declaring a variable, a condition, and the increment. When a for loop is created, a variable is given a variable, and it is compared against the condition. If the condition is true, then the code inside the loop runs. After, the variable is changed by the increment, and the process is repeated. Here is a for loop and a description of each step:

```
for(int i = 0; i < 3; i++)
{
    System.out.print("*"); //The College Board loves printing *'s
}
```

- 1: declare the variable **i** with the value 0
- 2: evaluate $i < 3$ which results in true
- 3: output: *
- 4: evaluate: $i++$ (the $++$ and $--$ operation increase/decrease the variable by 1 respectively)
- 5: evaluate: $i < 3$ which results in true
- 6: output: **
- 7: evaluate: $i++$
- 8: evaluate: $i < 3$ which results in true
- 9: output: ***
- 10: evaluate: $i++$
- 11: evaluate: $i < 3$ which results in false ($3 < 3$ is false)

In a while loop, the variable is declared outside the loop, and the increment is inside the loop, meaning that you can change the increment and any time, not just at the end of the body of the loop. Here is an equivalent implementation using a while loop:

```
int i = 0;
while(i < 3)
{
    System.out.println("*");
    i++;
}
```

3.4.4 Nested Loops

Loops can appear inside other loops:

```
for(int i = 0; i < 3; i++)
{
    for(int j = i; j < 3; j++)
    {
        System.out.print("*")
    }
    System.out.println();
}
```

Output:

```

    * * *
      **
        *
```

Loops with arrays and ArrayLists

```
ArrayList<Integer> integers = new ArrayList<Integer>();
int[] ints = new int[10];
//Using a for loop to set values for in ints and integers
for(int i = 0; i < ints.length; i++)
{
    integers.add(i);
    ints[i] = i;
}
//integers and ints have the values 0...9
//increase each value by 1
for(int i = 0; i < integers.size(); i++)
{
    integers.set(i, integers.get(i) + 1); //set the element at index i to
    the value at index i + 1
    ints[i] += 1;
}
/*
**There is also the for-each loop that can access the data within an
array list or array, but cannot change it
*/
```

```

for(int x : ints)
    System.out.print(x);
//this is equivalent to the following
for(int i = 0; i < ints.length; i++)
{
    int x = ints[i];
    System.out.print(x);
}

```

3.5 Questions

Create the following pattern by editing the code provided:

```

* * * *
* * * *
    * *
    * *
* * * *
* * * *

```

```

int i = 0;
while(i < 6)
{
    for(int j = 0; /*implement condition*/; j++)
    {
        /*
         Implement body
        */
    }
    i++;
}

```

Evaluate the following given that $t = \text{true}$ and $f = \text{false}$:

- 1: $!(t \parallel f) == f$
- 2: $!(t \ \&\& \ f) == t$
- 3: $((f == f) != (5 > 4)) \parallel (3 < 5)$
- 4: $(f == (!f \parallel (5 < (8 \% 3))))$
- 5: $5/3. == 5/3$
- 6: $5./3 == 5/3$

Create a for loop that will reverse the order of an array before: $i = [1,2,3,4,5]$ after: $i = [5,4,3,2,1]$

```
//code for swapping data without creating a new variable
int[] x = new int[2];
x[0] = 1;x[2] = 2;
x[0] = x[0] + x[1];//3
x[1] = x[0] - x[1];//3 - 2 = 1, x[1] is now equal to the old value of
    x[0]
x[0] = x[0] - x[1];//3 - 1 = 2, x[0] is now equal to the old value of
    x[1]
```

4 Classes and Objects

Classes allow you to create your own object types that can be changed by user defined functions. Each .java file can define one object. Here is a .java file for an on object called **Obj**:

```
import java.util.*;

public class Obj
{
    private int x,y;//private variables
    public obj(int x, int y)//constructor
    {
        this.x = x; this.y = y;
    }
    //get methods
    public int getX()
    {
        return x;
    }
    public int getY()
    {
        return y;
    }
    //set methods
    public void setX(int x)
    {
        this.x = x;
    }
    public void setY(int y)
    {
        this.y = y;
    }
    //main
    public static void main(String[] args)
    {
        Obj o = new Obj(3, 5);
        o.getX();//3
    }
}
```

```
        o.getY();//5  
        o.setX(10);  
        o.getX();//10  
    }  
}
```

components of this class:

- 1: Private variables: Variables that cannot be accessed by normal means. Methods can be written to access private variables
- 2: Constructor: A function that is called when creating a new instance of the object. An object may have many different constructors
- 3: Get methods: Methods that return the value of private variables
- 4: Set methods: Methods that set the value of private variables

4.1 Why use objects?

- 1: Code reuse: Several instances of the same object can be created to save time and make more organized code.
- 2: Variable scope: Variables outside of objects cannot have their value changed by other functions. Variables inside objects can through set and other methods
- 3: Control: Set ways for accessing the data within an object