The Vanilla Java Programming Language

The Smarty Party 11366

Version .2

1. Basics

What is Java?

Java is a compiled programming language that is supposed to be "write once, run anywhere". To elaborate on that Java code is supposed to be able to run on any computer with Java installed regardless of architecture or external libraries installed. Based off C++, Java is an object oriented programming language with C like syntax but with a few changes, such as thoroughly integrating data and functions than C++ and less low level capabilities than either C or C++. Java has found use in the business world as a modern com- piled language for client-server development on and offline as well as mobile app development. This paper will address the basics of Java Programming. You will learn all about data types, variables, arrays, opera- tors, and controlling the flow of a program to name a few items. There will be exercises at the end of every section along with sample code. I highly recommend typing in the code you see by hand and altering it a little bit to gain a better understanding of how it works. Test and experiment, breaking programs doesn't cost money and is the only way to understand how the language works intimately.

Installation

The process of installing Java varies by operation system and your use case, the easiest way to install Java only by far is Linux. On Linux you simply need to search and install the newest stable version of Java using your package manager. For demonstrative purposes I will be provide the line to install Java 7 on Debian as of November 11 2018.

```
sudo apt-get install openjdk-7-install
```

On Windows to install Java you would need to navigate to the Java site or OpenJDK website and install the version suitable for your purposes, the same process applies on MacOS. If you intend to write and compile the code from the same screen there are plenty of IDES, interactive development environ- ments, that include Java installs with them. If you intend to write Java from Android studio, Eclipse, or Net beans then separate Java installs are redundant.

Importance of Simplicity

The main purpose of writing code in a legible manner is not for the computer, the computer is going to process and reorganise your code anyways when its being compiled, the main purpose of writing clear code is for yourself the people who may read your code later. General rules to take when formatting is to make your variable names descriptive but not too long (some examples below), incrementing variables are usually i or x unless they need to be more descriptive, tab code between brackets to show the scope of func- tions and loops.

These are rules of thumb and are not final.

Dependencies are a double edged sword in programming. You don't need to write as much code when you used someone else's libraries but by nature of dependencies you are reliant one someone else's code to make your program, this usually leads to bloated programs that take up more system resources. While modern computers can afford to handle some bloat this is a slippery slope that eventually leads to large programs that start using a gigabyte of ram.

1.1. Variables

Variables are the registers that data types describe. They are place holders for a value or state, much like variables in algebra. Variables can be assigned values as long as the new value being assigned matches the data type of the variable or the new value is being cast into the same datatype the value has. Variables in Java must obey certain rules such as A) variables cannot have spaces in their name b) variables cannot be only numbers for example int 8 is not value C) variable names must be unique from other variables or else you are simply changing the value of one variable not making a new one and D) variable names cannot be reserved keywords like "while" or "boolean" because Java has predefined behavior for these words and that behavior isn't being a variable. There are many different data types of ranging possible values and they are as follows: boolean, byte, short, char, int, long, float, double. These data types are arranged from lowest to highest possible value there are some variable examples listed below.

Data Types Name data held Version minimum value maximum value boolean boolean Java 7 and 8 false true byte numerical Java 7 and 8 -128 127 short numerical Java 7 and 8 -32768 32767 int numerical Java 7 -2^31 2^31-1 int numerical Java 8 0 2^31-1 char character Java 7 and 8 \u0000 \uffff long numerical Java 7 and 8 -2^63-1 2^62 float numerical Java 7 and 8 -126.1022 127.1022 double numerical Java 7 and 8 -1022.16382 1023.13383

Boolean Booleans are simple true false values. By default when you declare a boolean in Java the boolean is "false", you can assign it the true value. Booleans can either be "true" or "false" an example of two boolean can be seen below. Booleans are the basis for logic in computation and are seen everywhere in functions but especially in the "Controlling Flow" section of this document.

Byte

A byte indicates the variable holds 128 bits or exactly one byte. A byte is the smallest possible datatype in Java and it holds numerical value. Bytes are used for projects that need to be very conservative with their memory usage yet still need the portability Java provides. Unless you are working on a program for some embedded hardware like a kiosk chances are you wont see much of this data type.

Short A short says the variable holds a numerical value marginally larger than a byte, again its used for situations where free memory is scare and resource conservation is a high priority. One other notable detail about the short is the amount of memory allocated to it as a char so hypothetically if you typecast a short (a concept we explore in "Whats Going on beneath the Hood") you could convert the short into a character.

Char and String

The char datatype indicates that the variable holds a single Unicode character. The char datatype was expanded on by the creators of Java and the String Object was added to the standard library. The String descriptor is an object not a datatype because its a list of characters lumped together in an object instead of its own type of data. Also, unlike normal datatype when you declare a Sting the 'S' at the beginning needs to be capital.

Int

An int is a datatype that describes an integer, its the most common datatype for numbers since it isn't nearly as resource intensive as a long but still larger than either a short or a byte which are used for pro- grams that have access to very small amounts of free memory. An Int is the goldilocks of integers in Java. The amount of data an int can hold changed from Java 7 to Java 8. Minimum and maximum values from before and after the change are listed below.

Long

Long is a larger version data type that stores numbers. I can only assume its namesake is the length of its maximum value and minimum value. A long is more often or not used when creating functions and stor- ing large amounts of data.

Float

A float is a datatype that indicated the variable is going to store a number that can have decimal places. The reason a float is called a float is because it has floating point precision instead of truncating like a byte, short, int, or long would. Truncating a number means rounding up the decimal value to the largest whole number. For example if I had an int with the value 2.5 and I didn't get an error telling me to make it a float it would round up to three while if it were float it would remain a 2.5. I have seen floats used when measuring a range of value where precision is a high priority.

Double A long is to an int as a double is to a float. A double is a floating point integer like its float counter- part but its twice as large. Again like a long its intended to be used on a very large variable but this time when floating point precision is necessary.

```
class main{
    public static void main(){
        boolean state = true; byte byteMin = -128; short shortMin = -32768; int java7Min = Math.pow(-2,
        31), java8Min = 0; char character = 'c'; String sentence = "The quick brown fox jumps over the
        lazy dog." long longMin = Math.pow(-2, 63); float floatMin = -126.1022; double doubleMin =
        -1022.16382;

System.out.println("It is " + state + " that the earth is flat"); System.out.println("byte minimum value: " + min);
System.out.println("short minimum value " + shortMin); System.out.println("int minimum value in Java 7 " +
java7Min); System.out.println("int minimum value in Java 8 " + java8Min); System.out.println("the character
```

stored is: "character); System.out.println("the sentence stored is "sentence); System.out.println("long minimum value is " + longMin); System.out.println("float minimum value is " + floatMin); System.out.println("double minimum value is " + doubleMin); } }

1.2. Operators

Operators are procedures that alter sets of data stored in variables. Operators are a necessary piece of programming because data is useless if there is no way to interact with it. Operators range in function from string manipulation, to datatype manipulation, to math. Combining operators in specific routines to create sets of smaller instructions is the process of creating a function which will be explored in greater detail in the "Controlling Flow" section of his document.

State Operators

Modern programming languages rely on boolean logic to determine weather to execute blocks of code that are hidden inside of if statements or various loops. To check the state of two blocks of data, a variable or whatever else, there are operators which test the numerical and boolean values of two values and return a true or false value. Relevant state operators and examples of them being used are listed in a table at the end of the section after more detailed descriptions of the operators that may warrant a more detailed explanation of their function. All examples in the table below are resolved as true by Java.

Not Operator

The not operator or the bang is represented by an exclamation mark and to put it in simple terms it checks if something set true or inverts the value of a boolean value. The "not operator" is usually combined with other operators like the parentheses it the equal to operators simply out of convenience. In those two situation it allows the programmer to test a condition in the parentheses and act if it its false or to outright check for inequality.

And Operator

The "and operator" tests of two conditions are true. Both of the conditions tested need to resolve true for the and operator to return a true. If even one of the conditions being tested resolves to false the whole thing is false. Since the and operator or operators are so low on the evaluation priority list it shouldn't be necessary to use parentheses around the conditions being evaluated but for legibility's sake the standard remains.

Or Operator

The "or operator" test if one or two of the cases being tested are true. For the or operator to resolve as true you only need one statement to evaluate as true. All the same principals as the and operator applies.

State Operators Operator Name Operator Example not ! boolean state = false; !state; greater than < 2 > 1 less than > 1 < 2 if equal == 1 == 1 if not equal != 1 != 2 greater to or equal >= 2 >= 2 or 3 >= 2 less than or equal <= 2 <= 2 or 2 <= 3 and && (2 != 1) && (2 == 2) or || (1 == 2) || (2 == 2)

Type Casting

Type casting is when you take a variable and convert its value from one type to another most typi- cally from character to an integer or vis-versa. This is useful when you need to manipulate streams of char- acters of integers or need to

```
public class Main{
    public static void main(String args[]){
char letter = 'a'; System.out.println((int) letter); } }
```
OUTPUT: 97

Math To interact with numerical data there are operators that correspond with each of the arithmetic operations and modulo. There is an addition operator which adds numbers, a subtraction operator which subtracts two numbers, a multiplication operator which multiplies two numerical values, and finally division and modulo operators. The modulo and division operators divide numbers but modulo returns the remainder of the division problem while division returns the result of the division problem.

There are shortcuts built into the syntax for some simple math equations like incrementing or de incrementing a numerical value. To increment you place two plus signs either in front or behind the variable being incremented, the same principal applies with de incrementation. Somme other syntax shortcuts are the mathematical assignment operators. They are represented by their mathematical operator followed immediately by an equals sign e.g. "+=" or "%=". If we assume the variable "A" is an integer "A+=1" is the same as "A = A + 1".

```
class main{
    public static void main(String args[]){
        int incrDecr = 1; int shortHand
        = 1;

        int addition = 1 + 4; int subtraction = 9 - 3;
        int multiplication = 2 * 2; int division = 2 /
        1; int modulo = 21 % 6;

        System.out.println("sum of an addition problem: " + sum); System.out.println("difference
        of a subtraction problem: "
                                                    + difference); System.out.println("product of a
        multiplication problem: " + product); System.out.println("result of a division problem: " + remainder);
        System.out.println("remainder left after 21 / 6 : "
                                                        + modulo); int increment = incrDcr++; int
        decrement = incrDcr--; System.out.println("Value of incrDcr after increment: " + increment);
        System.out.println("Value of incrDcr after decrementing: " + decrement);

        shortHand+=1;
```

```
        System.out.println("shorthand after adding one: " + shortHand); shortHand = 1;

        shortHand-=1; System.out.println("shorthand after subtracting one: " + shortHand); shortHand = 1;

        shortHand*=6; System.out.println("shorthand after multiplying by five: " + shortHand); shortHand = 4;

        shortHand/=2; System.out.println("shorthand after dividing by two: " + shortHand); shortHand = 19;

        shortHand%=3; System.out.println("shorthand after getting the remainder of 15 / 3: "
                                                                + shortHand); shortHand = 1; }
}
```

OUTPUT: sum of an addition problem: 5
difference of a subtraction problem: 6 product of a multiplication problem: 4
remainder of a division problem: 2 remainder left after 21 / 6: 3 shorthand after
adding one: 1; shorthand after subtracting one: 0 shorthand after multiplying
by five: 5 shorthand after dividing by two: 3 shorthand after getting the
remainder of 15 / 3: 6

Parentheses

Parentheses are used to clarify the order in which commands will be executed on a line. Java, against common belief, doesn't execute the code the you wrote it literally in reorders the code you wrote in compile time so your code will be more efficient in run time. The parentheses explicitly states that the compiler should not rearrange the order of operations within the parentheses. Code within the parentheses will also be executed first so in the example below three will be multiplied by nine before eighth is subtracted.

Ternary Operator

This is technically an operator since it uses logic to assign value. A ternary operator uses logic to determine what value should be assigned to the variable in question. While the ternary operator can be used in place of a series of if-else statements it usually sees usage when determining what value is going to be assigned to a variable or returned at the end of a function. If the case being tested resolves as true then the value to the of the colon right is assigned, otherwise the value to the left of the colon is assigned. This lets you alter variable assignments or return values based on situations that may arise.


```
class main{
    public static void main(String args[]){
        boolean state = false; int result = (state)?1:2; System.out.println("since state is " + state + "the
        result is "
```

+ result); } }OUTPUT: since

state is false the result is 1

1.3. Arrays

An array is a collection of values all under one name. The different values are placed in a list and the way you reference the individual values is by naming the array and the specific position the item is in for example in the example below if I wanted to reference 14 I would use the numberArray name and within the square brackets type in the number that corresponds with 14, in this case one.

To create an array you need to define the array type, any datatype can have an array assigned to it. The type defines what data will be stored in the array. Then, you create the array with the "new" keyword, the amount of spaces in the array is specified after the new keyword in the square brackets. To assign a value to a slot you assign the value as you would to any variable but you need to include the slot you will be assigning the value to in the square brackets following the variable name. An example of how to assign an array can be seen below.

```
class main{
        public static void main(String args[]){
int numberArray[] = new int[10]; numberArray[1] = 14; numberArray[2] = 88;
System.out.println("value one: " + numberArray[1]); System.out.println("value two: " +
numberArray[2]); } }OUTPUT: value one: 14 value two: 88
```

1.4. Controlling Flow

ScopeScope is the range where Java sees a variable and utilises it. To begin with Java works procedurally so if a function is used in the code before its office illy declared then you will get an error because it does not exist to the Java compiler yet, you are trying to do something that it does not know. Aside from working done form the top of the file to the bottom procedurally then variables exist withing curly brackets like this: { or this: }. When code is contained within these brackets is interpreted separate from whatever cane before. You can reuse variable names because any code within these brackets is separate from anything else, Java's scope doesn't move from beyond those curly braces.


```
class main{
        public static void main(String args[]){
                int variableOne;
                        while(1){
//this is separate from the other variableOne int variableOne; } } }
```

If-Else Statements

If else statements allow you to execute code within the brackets if the condition within the parenthe- ses is satisfied. There is an order to which you call an "if statement" and "if else" statement or an "else" statement. An "if statement" is the first of the group and it can only be used once, and if else statement comes after the "if statement" and can be used as many times as necessary while the else statement can only be used once at the end. The "else statement" unlike the if or the else if statement doesn't have any paren- theses to define conditions to run the code contained with its braces, this is because the else statement runs if none of the conditions defined in the if or the if else statements was true.

```
class main{
    public static void main(String args[]){
        char letter = 'z';

        if(letter == 'a'){
System.out.println("the letter is a"); }else if(letter == 'b'){
System.out.println("the letter is b"); }else if(letter == 'c'){
System.out.println("the letter is c"); }else{System.out.println("the letter is not a, b, or c"); } } }
```

Switch Case

Switch case is like an if-else statement but different. Like an If-Else statement the switch case tests the value of a variable or variables and executes code based on the value but unlike an if-else statement the switch case only tests one case and has no scope limitations. That means that in the example below if the letter stored in the "character" variable is an 'A' then you could see A, B, and C print to the console. The "default" case like the else statement earlier executes if none of the conditions defined earlier are met. An example of this can be seen on the next page:


```
class main{
    public static void main(Strings args[]){
        char character = 'a';
        switch(character){
            a:
System.out.println("A"); b:
System.out.println("B"); c:
System.out.println("C"); default:
        System.out.println("some letter that is not A, B or C"); } } }OUTPUT: some letter that is not A, B, or C
```

For Loop

the for loop is meant for situations where there a block of code needs to be executed a predetermined amount of times. The first part of the for loop declares a variable used within the for loop to keep track of the amount of executions, in the case below 'i' and it is followed by a semi-colon to break up the expres- sions in the loop. The middle expression is to show what must be true in order for the bock of code within the for loop to run and after the semi-colon it shows the i value incrementing by one. An example can be seen below.

```
class main(){
    public static void main(String args[]){
        for(int i = 0; i != 4;i++){
            System.out.println("current value is " + i +
                                " and the max value is 100"); } } }
```

OUTPUT: current value is 1 and the max value is 4 current value is 2 and the max value is 4 current value is 3 and the max value is 4 current value is 4 and the max value is 4

While Loop

The while loop unlike the for loop is intended for when you have a case you want to test without a specific amount of repetitions in mind. Its like the for loop but a bit more flexible. To continue with the dif- ferences between the loops you don't have the differences segments of the loop you only have one condi- tion being tested within the parentheses. An example can be seen on the next page.


```
class main{
    public static void main(){
        while("apples" != "oranges"){
System.out.println("don't compare apples and oranges"); } } }
```

OUTPUT: don't compare apples and oranges

Do-While Loop

the do-while loop is identical to the while loop but it has one important difference aside from its form, it executes the code within its braces once before checking the conditions its testing for. The for and while wile loops both wont execute their code as long as the conditions in the parentheses aren't met so the do while loop. An example can be seen below:

```
class main{
    public static void main(){
        do{
```

```
System.out.println("we compared apples and oranges once"); }while("apples" != "oranges"); } }
```
OUTPUT: we compared apples to oranges

Break a break is used to stop execution of a loop early. This is usually used when a condition is met pre-emptively or to stop some code execution in the loop for safety reasons when working with moving parts.

```
class main{
    public static void main(){
        for(int i;i < 100;i++){
            if(i == 50){
break; }System.out.println("This for-loop is set to go to 100 but
                                                    will break at 50"): } } }
```
OUTPUT:

1.5. Comments

Comments are lines of code that aren't read by the Java compiler, they are typically used to write quick little descriptions for anyone glancing over the code. Its just to save the reader time by clearly stating the details of a programs implementation. There are single line comments and multi-line comments. The single line comment is denoted by two forward slashes next to each other like so //. An example of the sin- gle line comment can be seen below.


```
class main{ //this is the main function
    Public static void main(String args[]){
//this prints System.out.println("prints"); } }
```
The multi-line comment is the same thing but instead of removing one line you need to declare the beginning as well as the end by using a forward slash and a star to begin the comment and a star and slash to end the comment like so: /* commented out */. An example of this can be seen below:

```
class main(){
    public static void main(String args[]){
System.out.println("not commented out");
/*System.out.println("commented out"); System.out.println("commented
out"); */ } }
```
OUTPUT: not commented out

1.6. Functions

A function is a set of events that can be called by their name at any point in the program. These sets of instructions can have arguments passed to them in-between the parentheses when they are called as long as the arguments read defined when the functions is defined.

```
class main{
        public static void main(String args[]){
                static yell(String word, int amount){
                        for(int i = 0;i <= amount; i++){

System.out.println(word); } }yell("hello, 6"); } }OUTPUT: hello
 hello hello hello hello hello
```


1.7. Object Oriented Programming

Object oriented programming was created in the nineties as a response to the dominant way of organ- ising variables and functions of the time, procedural programming. Java is an object-oriented language which means that Java uses class hierarchies to organise data and functions. The main goal is to have reus- able code by organising functions and variables into familial hierarchies that let objects share functionality.

1.7.1. Objects

An object is collection of data and functions held under a chosen name. To declare an object you need to use the "class" keyword followed by the name of the class. The class allows people to move code from one project to another with minimal modification since every variable and function are held indepen- dently. So say a developer finds a better version of an object they are using they should hypothetically be able to replace the old object with the new object with minimal tweaking since the object's functionality is self contained.

Objects also allow for inheritance, you can "extend" an object and create a new object with features specific to their intended function. Say you have generic object called car and then ??!you extend this into a tuck. A car is going to have the bear essentials lie a motor, chassis etc. A truck is going to have a larger bed more horse power, maybe it can hold less passengers. You can extend the truck again to make a dump truck or a moving truck with features that more specific. In this situation the car is the parent class who's children are the truck, dump truck, and the moving truck. The truck's children classes are the specific truck types and the specific trucks don't have any kids. If you wanted to you could abandon trucks altogether and just make a taxi. The class hierarchy is shown below.

Car Truck

Dump Truck

Moving Truck

Taxi

Declaration Modifiers

Declaration modifiers are keywords that change the locations where data can be accessed. Declara- tion modifiers change characteristics about the variable or function being declared. Variables may need to be accessed outside of the object they are declared in or the opposite, a variable of function must remain exclusively accessed by the object and by nothing else. Declaration modifiers are responsible for adding scope modifiers that can be used when declaring a variable to explicitly state what portions of the program can access the data contained within the variable. If the scope modifier is presumed to be "protected" as opposed to "public" or "private".

This is because protected variables are seen by the object itself and any children so in the case seen above if the truck class that described the color of the truck then the dump truck and the moving truck could

access that value but the car and taxi objects cannot. This provides extra security that isn't found with a public variable since any object no matter where they are in the hierarchy can access public objects and private variables are limited to the object they are declared in.

Scope Modifiers keyword
scope public anywhere private within
the class protected package and
subclass unnamed package and
subclass

the remaining modifiers change the values returned by functions or the modifiability of a portion of the

program regardless if its visible or not. The class keyword defines a class it need to be followed by the class's name and the contents of the object. When declaring a new object it can "extend" the functionality of another object if it exists the name of the object needs to be followed by the extends keyword and the object it will be extending.

The static keyword means that the function or variable is local and cannot be altered. Regardless of if its public or private it cannot be edited after its declared, if its a function that is given the static modifier the contents of the function cannot be altered in runtime. The final keyword is used to create a variable that is for all intents and purposed public it cannot be altered or coped in any way shape or form. If an object is given the final modifier it cannot be modified and if a variable is given the final keyword it cannot be altered, it is a constant. Volatile is the opposite when you pass the volatile keyword to a variable it is expected to change often so it is going to be given priority. A void function means that nothing will be returned when the function completes there cannot be a void variable.

Declaration Modifiers keyword applicability

usage	class	class creating a class
extends	class	creating a child class
static	function, variable	unchanging instance in the class
final	function, variable	unchanging instance globally
void	function	returns no value
volatile	variable	all value that changes often

2. In Conclusion

Java is a very rich programming language but this document only addresses the basics of Java. The principals here can be applied to more specific programming projects like android app creation, to programming for FTC robotics there will be purpose specific libraries and functions you will need to learn. The documentation will be your best friend when trying to learn any new library or language.