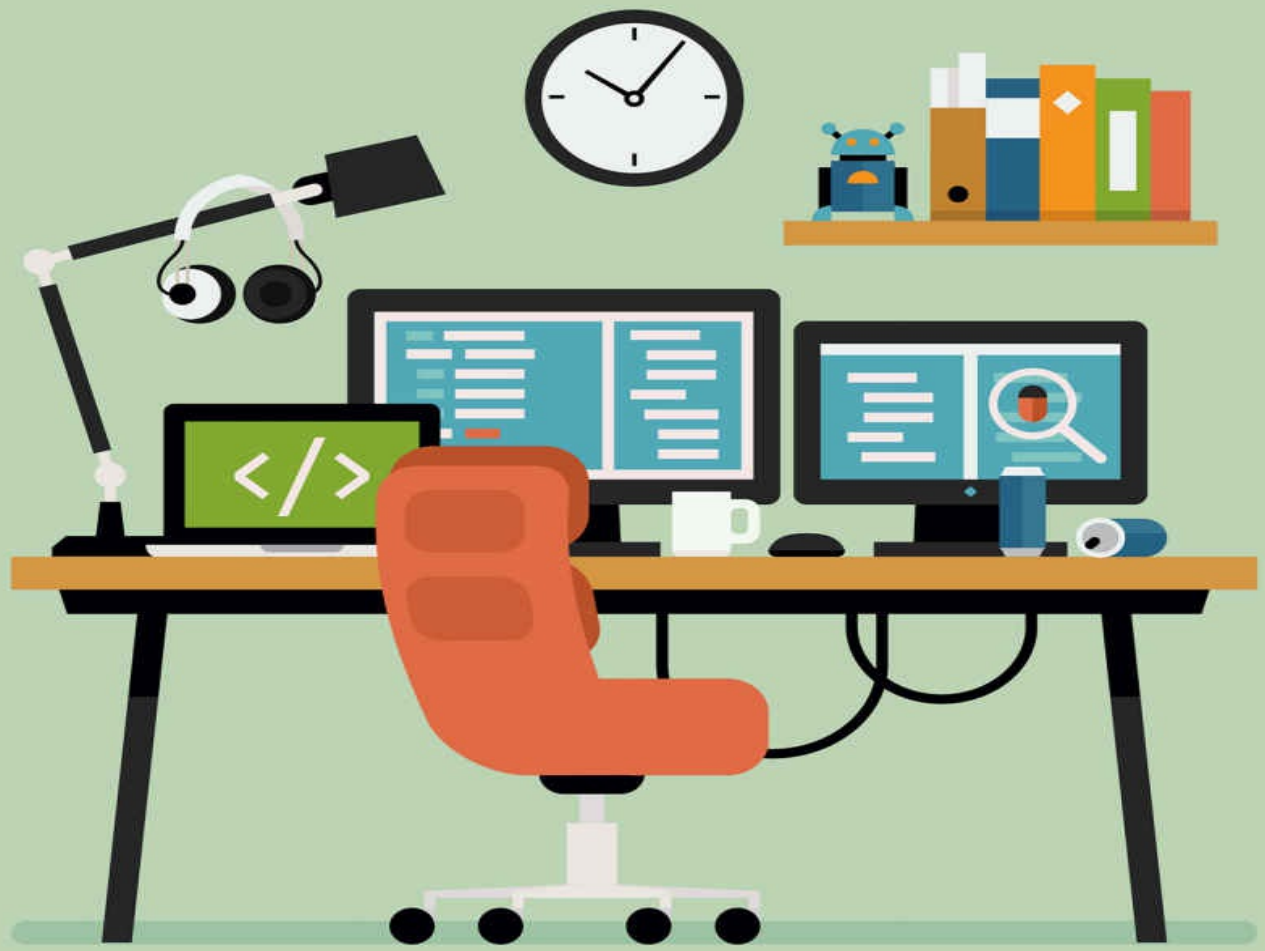
JAVA PROGRAMMING for Beginners

CRASH COURSE



MARTIN LAREDO

Martin Laredo

Java Programming for Beginners

Crash Course

Table of Contents

Creating your first programming language is easier than you think.

“The book I want to read.” — Matz, creator of the Ruby language

“I really love this book.” — Jeremy Ashkenas, creator of the CoffeeScript language

Want to create a programming language, but don't feel like going through one of those expensive and boring 1000-page books ? Well, you're not alone ...

Get it here !

Introduction

Congratulations on downloading *Java* and thank you for doing so.

The following chapters will discuss how Java works and what needs to be known so that you can have the best experience when you are working with Java.

Programming is not going to be easy and it is going to take a lot of time and patience, do not be discouraged if it does not work out the first time, it will work eventually.

I am going to tell you that Java is going to benefit you in ways that you may not ever realize. Java can help you get a new job or even advance in your job that

you currently have, so learning Java is going to be good for you on a professional and personal level.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy!

Chapter one: This is Java

I have found that the hardest thing to learn about Java is what needs to be installed before I can write the code that I want to write. If I do not have all of the programs, then I am going to be setting myself up for a headache because I am not going to be able to run the program properly.

VM

Java has a platform that is going to run by itself without the need to have any other programs assisting. Therefore, it is going to run off of any operating system that I want to work with. This is thanks to the Virtual Machine that the creators of Java made.

The virtual machine is going to aid in making sure that my code is processed correctly. Not only that, but it is not going to run Java code without having the virtual machine installed.

Oracle was the company that created Java and on their website, is the download links to any Java related programs that need to be downloaded such as Java Virtual Machine otherwise known as Java Runtime Environment.

I first need to check to see if my computer already has Java Runtime Environment because there are a lot of computers that come with Java preinstalled. To do this, I am going to go to the Oracle website and click on the box that says “Do I have Java?” Clicking on this link is going to scan my computer to let me know if I have Java or not.

In the event that I do not have it, I am going to be able to download it and get it on my computer so that I can write the code that I want to write.

Software development

After I have installed Java, it is going to run off my computer just like any other program is going to run. However, before I am able to write code, I am going to have to download the software development kit.

The kit is going to be the standard edition and it is going to download the NetBeans program. While this download is going to take up a lot of time because it is a large file, once it has been downloaded, the code is going to be able to be written into the program so that I can create my own Java programs.

The innerworkings of Java

When I am coding for Java, I am going to be using a text editor. My code is going to be called Source code and when I am saving my work, I am going to be saving it with the extension of .java.
At the point in time that I save my file, I am going to launce Javac so that I can now create my class file. Thanks to NetBeans, I am going to run my program using its own software so that I do not have to open multiple windows to enter commands and then run them to test my code. I can do everything from one window.

Chapter two: NetBeans the Program

When I first open NetBeans, I am going to need wait so that the program can fully load. Just like when it is downloaded, it is going to be slow in getting the program up and going. However, when it is loaded, I am going to go to file so that I can start a new project by selecting application.

After the Java Application has been clicked, I will select the next button so that it goes to the next section. I am going to need to name my project so that the text that is located at the bottom of window so that the text changes to match the project name.

When the file name has been set, it is going to name the class that I am going to be using. So, if I name it littlefoot.LittleFoot then the L and P are going to be lowercased for the package.

The NetBeans program also makes it to where the program that I am creating is going to automatically save to the default location, however, I can change this so I do not have to worry about not being able to find it at a later date. I am going to essentially create a folder that is going to be the name of the project that I am working on so that I can find it and work on it later. All the files for that program are going to be listed in that folder.

The IDE is going to allow me to see all of the projects that I have created and see if there are any that are still in progress. When I select the plus sign next to the projects name, I am going to see any and all folders that are tied to that

project. The folders are going to expand as well so that I can see where my code is being stored.

My code is going to be shown on the right hand side in the text area. Should I not see my code, I am going to need to double click on the project to force my code to appear so that I can work on it.

The file name and the class name are going to need to match or Java is not going to work with that program the way that it is supposed to work. I will ultimately end up getting an error when I have to compile my code so that it can run properly.

Java Comments

Once I have created a new project, I am going to see spots of text that are grayed out and are either going to have slashes or asterisks. The gray text is the comments that are ignored by the program when the code is run. I can type anything that I want into a comment and that comment is not going to be seen by anyone that is not working on the code. Most of the time, these comments explain what is going on with the code so that it can be worked on by other programmers.

Example

```
//I have created a comment

// This is where my comment starts
```

// and this is where it is going to continue because it is a long comment.

Or

/* I have made a different type of comment here
While it goes on to another line here
*/

I can also use a Javadoc comment to enter comments. These comments are going to have a forward slash followed by two asterisks while ending with a single asterisks and one slash. However, every line I create is going to need to have an asterisks in front of it so that it is deemed a comment.

Javadoc is going to use comments in the document's code so that the code is explained whenever it is turned into an HTML page that is going to be used by other people.

I am going to need to generate a Javadoc in the menu for NetBeans, but, since I have not written any code, I am not going to see anything in my window. NetBeans does generate comments automatically so that I can see an example, and I can delete these comments so that I can insert my own.

Structure Java Code

Looking at how the code is going to be structured, the name for the package is going to come first when I am looking at my code. This line is going to end with a semicolon and if I do not put that semicolon in, then the program is not going to be able to be compiled therefore, it is not going to run. Listed after the

package name is going to be the class name.

The class name is going to be set up like it is a segment of code. I have to indicate where the code is going to start and where it ends by using the curly brackets ({ }). Everything that is inside of the brackets is going to be known as a code segment that is going to be executed by Java.

As I am writing my code, I need to include the keyword of main so that Java is aware of where the code should start being executed. It is going to be much like the curly brackets. If I happen to forget to put the main keyword then the program is going to give me an error message.

Running programs

Any Java program is going be run in an output window that is different than the window that I type all of my code in. This makes it to where I do not need to have any other windows open to run my code. Everything is run inside of NetBeans.

The easiest way that I can run a program in Java is to press F6, this is going to automatically run the program so that I do not have to accidentally close my program. But, in the event that I do not want to do that, I can always go to the start menu and select the button that says “run project”.

If all else fails, I can locate a green button on my tool bar that is going to run the program same as any other method that I might choose. I am going to need to make sure that the proper code is being run though especially if I have been working with multiple coding projects. To do this, I can right click on the

project that I am wanting to be run.

Chapter three: Manipulating Java Data

Java is just like any other programming language that is going to allow me to manipulate the data that I input into it while the program stores it on the memory. This data is known as text, objects, and even amounts. I have to give any data that I use a name, and this data can always be renamed. However, the name and the value are going to be Java variables.

Java amounts

Amounts can be stored in the memory of Java by using whole numbers or floating point numbers. Floating point numbers are going to be stored with an equals sign while whole numbers are stored with the keyword int.

Example

```
Public static voice main(String [ ] args) {
        Int last_amount;
        System.out.printIn (“I wrote Java code!”);
}
```

I am going to need to communicate with the program so that I can let it know what integers I want to be saved to the memory of Java; to do this I am going to use the int keyword and place a space after it. I am going to enter what I want this variable to be named as well, but I am going to have to follow a set of rules that has been set forth by Java.

1. I cannot use keywords in my variable names. If the word turns blue in my code window, then the word is a keyword for Java.
2. I cannot use spaces. Doing this will cause an error code to be returned, but, I can use underscores and uppercases to “separate” words like I normally would with a space.
3. I cannot have a number for the first character of a variable name. If I spell the number out, then that will work, but using the actual number is not going to work.
4. Having a name that is completely lowercased is going to be different than having one that has two uppercased letters.

When I want to put an integer in after the variable name, I am going to need to use an equals sign to ensure that the variable’s value is stored properly.

Example

```
Public static void main(String [ ] args) {
Int some_number;
Some_number = 55,
System . out . printIn (“Java code”) ;
}
```

Java is going to store the number 55 to the variable some_number. If I do not like how this looks, I can put all of the code in one line instead.

Example

```
Public static void main(String [ ] args) {
        Int some_number = 55;
        System . out. printIn(“Java code”) ;
}
```

I will have manipulated the last line of the script so that it will fit into one line. When I place information between the parentheses, it is going to need to be placed inside of a set of double quotes. The plus sign is going to make sure that two halves of the expression that I create are joined together so that the name of the variable is concatenated.

After all the code is typed out the way that I want it to look, I can run it in the output window to see how it looks before I put it on a website that is going to be used by other people.

The integer is going to be some_number in this example.

Two integers can be added together so that only one amount is stored as the result for the addition that was done by Java. Each variable that I use in my code is going to need to be separated by a comma.

Example

```
Fifth_number = 90 ;
Tenth_number = 66 ;
Result = fifth_number + tenth_number
```

The result is the sum of the fifth number and the tenth number. The sum is not

going to be assigned until after Java has successfully done the math for this equation.

Example

Answer = 90+66

Since there are two variables, Java understands this and the name for these variables does not have to be used. Doing this is going to change how the code that I created looks.
Example

System . out. printIn(“sum of the two variables =” + answer)

The plain text that is between the double quotes is going to be combined whenever the program is run so that the output is given in the output window that is displayed at the bottom of my screen.

I can also change my amounts directly so that the result is changed.

Example

Result = fifth_number + tenth_number + 65 ;

The program will run this code once more, but the result is going to be different since there is an additional number that has been added into the equation.

Java has put a limit on how high or how low the numbers that are entered into my code can go. The highest positive integer that I can use is 2147483647 and the lowest negative integer that I can use is -2147483648.

Variable X's 2

A double variable is going to be the integers of extremely high or extremely low value. The highest and lowest values that I can use is going to be 17 with 307 zeros following it. Double variables can also be used for floating point values. When I am storing floating point's for an int variable, the program will underline the code that I enter as unreadable therefore I can change it before I run the program and get an error.

Example

```
Int twentieth_value, first_value, result;

Double twentieth_value, first_value, result;

twentieth_value = 20.5;
first_value = 5.5;

Result = twentieth_value + first_value ;
```

The program is going to add together these variables and give out the result for the two values as an assigned variable to the keyword result on the left side of the equals sign.

Shorts that float

A short variable is going to be the values that are smaller than other integers. These numbers are going to fall between the value of -32768 and 32767. The code that I have been using is not going to work for shorts, instead I am going to need to use the short function so that I can make sure that the values are going to fall between the set limitations.

Large values are going to be stored inside of a double value along with floating point values. A double does not have to be used if I can use a floating point. As I am storing a floating variable. I am going to need to put a letter at the end of my value amount. This value should be f because it is a floating point number.

Example

```
Float first_value, twentieth_value, result ;
First_value = 15.3f;
Twentieth_value = 95.6f;
```

Chapter four: Math and More

Arithmetic

Java just like any other programming language can be used to do math. There are going to be the basic operations that I am going to be able to do with Java and their symbols are pretty easy to understand.

Adding – plus sign
Minus – negative sign
Multiplying – asterisks
Dividing – forward slash

Precedence for operations

It does not matter what I am doing with Java and math, I have to be sure that I am putting every value in the proper place so that I know what is being done and in what order it is being done in.

Example

```
Tenth_value= 5;
Fifteenth_value = 654;
Twohundredith_value = 54;

Result = tenth_value – fifteenth_value +twohundredith_value;
```

The expression is going to be evaluated in the order that it is entered into Java. what is going to happen is that 654 is going to be subtracted from 5 giving me a result of 649 before I add 54 which makes the sum 703.

When I want something to take priority in my expression, I am going to use brackets around it. Multiplication and division are always going to be done first when it comes to the order of operations.

Variables for strings

Java is not only going to store values in the memory, it is also going to store text that has been assigned for variables. When I am wanting to store a character, I am going to need to change my keyword from int to char.

I am going to start a new project and this one is going to be used for strings so I am going it name it something like VarsStrings so that I know that it is for the variables that I can put in strings. I am also going to want to ensure that I click the box that says that the main class will be created so that I do not have to do this later or risk messing up my project.

Once I have my text box open, I am going to delete all the comments that the program automatically places inside the text box so that I now have a clean slate to deal with. To create a string, I am going to need to use the word string so that Java acknowledges that this is not just plain text before I go into detail of placing my variables names.

Example

String game_title

The value for the string is going to be assigned to a new variable through the use of an equals sign. The data that I put after the equals sign will then be stored inside of double quotes.

Example

Game_title = “Raiders” ;

I can also place my script in a single line so that I can make my script easy to look at when I am looking for potential mistakes.

String game_title = “Raiders” ;

I will then put the creator of this game in a different line so that when it appears on the site that the user is on, it is going to have the game’s title along with the creator so that they are getting credit for what they made.

Example

String creator_name = “Markus” ;

At this point I can piece my code together for how I want it to look for my users.

System. out. printIn (game_title + “ “ + creator_name) ;

What I insert into the parentheses is going to be what my variables are. The plus sign indicates that there needs to be a space between the game's title and the creator's name.

The char keywords needs to be lowercased so that just a single character is being stored on the memory. This is also going to be when I am going to need to use single quotes rather than doubles. Should I forget and use double quotes, I will get an error code from the program.

Example

String game_title = "R";

While this will be the correct input

String game_title = 'R';

This is will return an error.

Chapter five: Input and Beyond

Java contains large directories that will allow me to use them in order to ensure that my code is being properly executed. When I have to reference the directories, I am going to need to call the directory to action.

A class directory that I have found useful is the scanner class. As long as I use the keyword import, I can pull any class from the directories so that it is used in my code.

Example

Import Java.util.scanner ;

This statement needs to be put before the statement that is going to tell other programmers what class is being used.

Example

```
Import java. util. Scanner ;
Public class VariablesForStrings {
}
```

The import section of the script allows Java to know that it needs to pull the class from the listed directory. Once it has completed that task, it is going to then enable me to start placing variables into that class. When I look at it, it is

nothing more than a chunk of code that will not do anything until I have successfully created an object inside of that class.

As I create a new object, I am going to use specific code so that Java does not try and pull from another directory to create this object.

Example

```
Scanner inputFrom_user = new Scanner( System . in) ;
```

The integer variable cannot be used to set up the variable along with the string variable function. I am going to have to use inputFrom_user specifically to ensure that it is running the program correctly.

My keyword is new so that a new object is created from the class and when that object is created, it is going to fall between a set of parentheses. This is going to now indicate the input that came from the system.

The input that comes from the user of my program has to be called upon using methods that allow them to know that they should interact with the program. One of the most common methods that is going to be used is the next method. The string that I write out is going to come after the user has typed in their response to the prompted question.

Example

```
String state_name;
State_name = user_response.next() ;
```

After the customer, has entered in the data that is required of them, a dot is going to be inserted from a list that will appear containing methods that I am going to be allowed to use. All I am going to need to do at this point is click on the method that I am wanting to use and then type a semicolon at the end of the line.

Example

```
String state_name;
System.out.print (“ Type in the state that you currently reside in: “ ) ;
state_name = client_input.next () ;
```

In trying to get data from the users, I am going to use print rather than the printin function. If I want to use print in, I am going to have to locate my mouse to a different line after the data has produced a result. However, when using print, I am going to be able to stay on the same line that I am working on.

Example

```
String city_name ;
System.out.print (“What city are you living in: “);
city_name = client_input.next() ;
```

While the code is going to be exactly the same minus the response from the client, Java moves the data to a new variable so that I do not have to do it myself.

Example

String complete name;
Where you live = state_name + “” + city_name ;
System.out.printIn (“You are living in” + state and city name);

As the program I created is run, Java will then pause it long enough for the user to input their response. After the data has been placed on the memory, the output will be displayed on the screen, “personalizing” the experience that the user is having to them.

The point behind the scanner class is to get information from the user while the input from the user is going to be assigned to the variables it belongs to.

Options

There is an option list that is available to me by using the JOptionPane function as I attempt to get information that I need from the user of my program. The class is going to appear on the screen allowing the user to put in the appropriate response before displaying back what they typed in.

Example

Import javax.swing.jOptionPane ;

I will have to tell Java what class I want to use inside of the swing directory this time.

Because I did not want to change my code that I had already written, I went ahead and created a new project for this method. As I put my code in, I noticed that the javax.swing.jOption was underline as if it was an error. But, it is not an error it is going to direct the data to the variable that it needs to be stored under.

Example

```
String your_age;
Your_age = JOptionPane.showInputDialog (“ Your age “) ;
```

jOptionPane enables me to type a full stop so that my options list appears. On this list I am going to select showInputDialog by double clicking on it. This will place brackets in my code that is going to be where the message I want to be displayed to my users will be put. I am going to need to use double quotes with this as well.

Example

```
String your_age ;
Your_age = JOptionPane.showInputDialog (“ your age”) ;

String your birthday;
Your birthday = “You were born on” + month+”” + day;
JOptionPane.showMessageDialog ( null, birthday );
```

This is where I want my dialog box to appear again with all the options that

are open for me to use. The word null has been put into the brackets so that Java knows that this box has nothing to do with the program that I just created. The punctuation is going to be shown in the text for the final product just as I have it entered into Java.

At the end of my code, there is going to be some new code that I did not put there. This code is meant to tell the program when it should exit the system because it has been executed to its fullest. Even in doing this, the values and variables that are stored into the memory for Java are going to be stored for the next time that the program is used.

Whenever my code is executed properly, there are going to be boxes that pop up asking for different pieces of data from my users such as their name or their age. After the “Okay” button is clicked by that users, the input is going to be displayed with what the user entered into the box. This is also going to cause the program to exit the system because everything that needed to be done, has been done.

Chapter six: Controlling How Java Operates

All of the code writing that has been done with Java has been what is called sequential programming. This is where the code is entered and the program reads it from top to bottom not skipping any line that has been entered by me. But, not all of the programs with Java are going to work like this.

There is some script that is going to only be run once a condition that has been set is met. For example, perhaps I do not want anyone from a specific state to be allowed into my site, so if this condition is found as true, then the program will run, but if the user is from that state, the program is going to be terminated. Doing this is going to control how Java operates by using conditional logic.

The most common conditional statement is going to be the if statement.

If

With an if statement, the code will be executed whenever the condition has been invoked rather than waiting on the user to do something. The if statement is going to be an easy script.

```
If (expression) {
}
```

Using the example from earlier, if the user is from New York, then I am not going to want them to get into my site. Instead of entering all of the code that is

required for this, I am instead going to shorten the script so that it is easier to read by me and other programmers.

Example

```
if( client < New York) {
        // This site is restricted
}
```

So, if my user is not from New York, there are going to be two sets of brackets that will be skipped by Java as the program is executed. But, anything that is placed between them, the code will be carried out, but the condition that I set forth has to be met before this can happen.

Another short hand trick that I learned is going to be that the greater than symbol is going to be carried out by a triangle that is going to point to the right.

Example

```
If( client > New York) {
        //Message for the user
}
```

The code is going to be the same even for those who do not live in New York.

If/else

When an if statement is not going to be sufficient, then I am going to use an if

else statement.

```
If( what I want to be completed before the program runs) {
}
Else {
}
```

The first line of this code is going to be started with an if statement followed by the parameters that I have set into place. The brackets will divide my code up so that the code is executed based on my parameters. The second part is going to come after the else statement. So if this does not happen then this will.

With an if else statement, it is just like an if statement except it has a second expression that is going to fall between a new set of brackets. If the first condition does not get met, then the code will be quartered and be run through the second expression to see if it meets that condition. If I end up missing one of the brackets, then I am going to receive an error code.

```
If(expression_one) {
}
Else if ( expression_ two ) {
}
Else {
}
```

The conditions that can be set inside of my if else statements are as follows:

Greater than (>)

Less than (<)
Greater than or equal to (>=)
Less than or equal to (<=)
AND (&&)
OR (||)
A value of (==)
NOT (!)

With the AND condition, more than one condition is going to be tested at the same time.
Example

Else if (user > 45 && user < 55)

For this example, I want to know if my users are older than 45 but younger than 55. My user has to meet both conditions in order for the expression to work properly which is why I used the AND operation as well.

Ifs that are nested

When I place an expression inside of another, I am nesting the two expressions. This can be done with any if expression.

Example

```
if( client < 55) {
        system.out.printIn ( " 55 or less") ;
}
```

To see if the client is older than 45 I am going to place another condition inside of the first.

Example

```
If( client < 45) {
If( client > 48 && client < 45 ) {
        System . out. printIn (“ You are 46 or 47”) ;
}
Else {
System.out.printIn (“ 45 or younger”) ;
}
}
```

Any brackets that I use are going to determine if the code is executed properly or not.

Boolean

Booleans are only allowed to be true or false values. If they are anything else then they are not true Booleans.

The keywords int, string, or any other keyword does not need to be used when doing Booleans. Instead, I am going to simply type Boolean with a lowercased b to ensure that anything after it is a Boolean. But, I have to ensure that the values are true or false. I will use two equals signs to evaluate the values of

my expression.
Example

```
Boolean client = true;
If ( user == true) {
System. out. printIn (“the expression contains the truth”) ;
}
Else {
System.out.printIn(“this expression is false”) ;
```

If statements are going to check variables entered to see if they make true Boolean statements. In the event that what the user entered is already true, then it is going to be checked to see if it is false. It is not going to need to be stated that the value is not false because it is going to be easy to see that the value is actually true and vice versa.

The NOT operator is also going to be used when Booleans are being created by me.

Example

```
Boolean client = true ;
If( !user) {
System. out. printIn (“there is no truth”) ;
}
Else {
System.out.printIn (“this is the truth”) ;
}
```

The only difference in the NOT Boolean statements is the (!user) function. The operator is going to be placed before the variable with the use of an exclamation mark. As always, the variable is going to have to be tested to see if it is negative.

Being that the user variables are usually set to true, they are then tested to see if there are any false values in them. It is exactly the opposite if the value is set to false.

Chapter seven: Switches and Loops

As I am working on my script, I am going to use switch statements so that I have a wider range of options for the values that I am going to set for my variables that I use. In essence, these are really long and somewhat complicated if else if statements.

This is how a switch statement is going to look:

```
Switch ( variable_to_test_ {
Case value:
Code_here;
Break;
Case value:
Code_here;
Break;
Default:
Values_not_caught_above;
}
```

The keyword used for switch statements is going to be switch which will then have a set of parentheses following it. The variables that I want to check will need to go in the middle of the brackets that I have placed in my switch statement. Not only will the variables be placed between the brackets, but so will other parts of the statements. The case keywords are going to review all the powers that have to be reviewed by the program.

Case values are going to use colons instead of semicolons. The events that are going to occur with the program are going to need to correspond with the value correctly so that the code is run the way that it should be. The break keyword is going to be used whenever there is a new case that has been added to the statement.

Default values are going to be used but they do not have to be used by me. I can add any value that I want to for the statement to be evaluated in my switch statement.

Example

```
Public static void main(String [ ] args) {
Int user = 87;
Switch ( user ) {
Case 45:
System. out. printIn(“ the price is 45 dollars”) ;
Break ;
Case 65:
System. out. printIn (“ the price is 65 dollars”) ;
Break ;
Case 88:
System. out. printIn (“ the price is 88 dollars”) ;
Break ;
Default:
System. out. printIn (“ the price is 45, 65, and 88 dollars”) ;
}
}
```

The switch statement code is going to need to test the integers that are being entered by the user. The codes value has been set to 87 in this example which tells the program to check everything that is inside of it. Each case will be checked individually to see if there is one that matches the value that was set by me. After the case that matches has been found, the code will stop and the program will be terminated.

The values that are out of range cannot be tested even though they have been placed in the case.

Example

Case (user >= 55) ;

This case cannot be used and will result in an error from Java.

But, I can use a case such as this:

Case 5: Case 6: Case 7: etc.

Each value that is used will need to be spelled out and checked.

Loops

The flow in Java is sequential and moves down the page just like a book is written. But, there are functions that can be used so that the flow is changed to something different. In order to interrupt the flow that Java naturally uses, I am

going to create a loop. With a loop, the program is going to continuously run the lines of script until it is told to stop.

Example

Int subtraction 100-5-3-4-8-6-2-9

I am not always going to want to use this method but the loop will keep going until the parameter that I have set up has been met. Once that condition has been met successfully, then the loop will be terminated and the next chunk of code is going to be evaluated.

Loops that use for

The loops that are going to use the keyword for are going to be one of the most commonly used loops when I am working with Java. this loop is going to set how many times the code has to be repeated before it is terminated.

For loops are going to look a little something like this:

```
For ( beginning_value; last_value ; increment_amount) {
//I am going to enter my code into this section here
}
```

The parentheses are going to contain three different values that are going to determine the entire loop. The beginning value is where the loop is going to start, the last value is going to be the value that the loop needs to reach, and the increment amount is going to determine how many times the loop is going to go

around. The increment amount is going to be defaulted to a value of one, but it does not have to be, I can make it whatever integer that I see fit.

The brackets that come next in the expression are going to be the code that is sectioned off so that it can be repeated.

Example

```
Boxed  loops ;
Public class ForLoops {
Public static void main (String [ ] args) {
Int loopVal;
Int last_value = 100;
For (loopVal = 5; loopVal < last_value; loopVal++) {
System. out. printIn ( “Loop Value = “ + loopVal) ;
}
}
}
```

The value that the integer is set to is going to be called a loopVal and this is going to be where the integer for the variable is placed. This variable is the one that will be used up until the loop is terminated.

Java has to be told where the loop is going to start and where it is going to end. loopVal is going to be a smaller value than the one that is used for the last value. When the word for is used, the value will need to be an amount that is smaller than the variable that has been placed at the end of the loop.

loopVal++ will result in letting me know what integer is being used each time the amount I am going to go up each time the loop is repeated.

Loops for while

A loop that uses while statements are going to be easier to understand, but they are not used as often.

While loops look similar to for loops.

```
While ( condition) {
}
```

While will be lower cased and my parameter is going to be in parentheses.

Example

```
Int loopVal = 8;
While (loopVal < 33) {
System.out.printIn (“my text”);
loopVal++ ;
}
```

The conditions that I set forth are going to be less than adequate while the code placed in the brackets will be displayed. Some loops can be on repeat for all eternity since they do not have any value assigned to them.

To eliminate the loopVal, I have to reach the end of my condition. While loops

are going to count values instead of checking them. the code will keep going until a button has been pressed on the users keyboard.

Do while

Do while loops look different than for and while loops.

```
Int loopVal = 6;
Do {
System. out. printIn( text") ;
loopVal++;
}
While ( loopVal < 2 ) ;
```

Loops are going to go up until the condition is satisfied. The distinction between while loops and do while loops is that the script I enter is going to be inside of two brackets. This code will be examined from top to bottom until the condition has been met.

Loop is going to be terminated and the code will not be executed if Java decides to not run it. As I wait for the program to test my script, I am going to need to use a while loop before I can use a do loop. The value will change for the variable that I have set forth in order for the code to run. The text is not going to be displayed as it was with the other loops.

Chapter eight: Arrays

The arrays that Java uses can hold multiple values at the same time. It is easier for me to think of an array like a list that has been placed on a spread sheet. I am going to be able to create as many columns in the array as I want to.

Example

0 – 55
1 – 654
2 – 566
3 – 552
4 – 6
5 – 852

Every row is going to be assigned a position just as if I had created it on a spread sheet. The positions are going to always start at zero and go up by one each time that an integer is added.

As I set up my arrays, I am going to give Java the knowledge of the data that I am placing in the array. I am going to have to tell Java how many positions I am using so that it knows where it should stop.

Syntax for arrays:

int [ ] aryNums;

Normal integers are going to be placed in the brackets after I have officially selected the data type that is going to be used. The brackets let Java know that it is an array that is being used rather than any other data type. Arrays use the aryNums function so that it ensures that the array is going to be set up properly. After that, any variable that I want to put into the array can be placed.

Java will know how many places that I am using for the array once I have set up an new object for the array.

Example

aryNums = new int [55];

This tells Java that 55 spaces are going to be used for integers in this array. They will range from zero all the way to fifty-five.

As I go to assign values to the positions, then I am going to use the normal methods that I have used previously.

aryNum [8] = 32 ;

The amount of 32 is going to be assigned to the space of 8 in the array.

I have to remember that arrays have to start at zero.

Example

Setting up my arrays like this is going to make it easier for me to deal with.

Int [ ] aryNims = { 8, 5, 6, 2 } ;

This method has created my array inside of the brackets where the first value is placed at zero. The square brackets that come after my keyword are going to make sure that the data type is not repeated elsewhere. This method is going to also be used with string values and chars when I want to use new keywords.

Example

String [ ] aryStrings = { " Vampires, Werewolves, Hybrids, Witches" } ;

This will work with Java

Boolean [ ] aryBools = { yes, no, no, yes } ;

This will not work.

If I want to use Boolean values I have to use a new keyword all together.

Example

Boolean [ ] aryBools = new Boolean [ ] { yes, no, no, yes } ;

Arrays for loops

Arrays use special rules for their loops. The values in an array will be assigned using the array syntax. This does not make sense to me, but this is

how values will be assigned,

Example

```
Package prjarrays ;
Public class ArrayTest {
Public static void main (String [ ]  args) {
Int [ ]  ticket _ prices = new int [ 65 ] ;
Int I ;
For ( I = 0; I < ticket_prices . length ; i++) {
Ticket_prices [ I ] = I +1;
System. out. printIn ( ticket_prices [ 6 ] ) ;
}
}
}
```

The array above will contain 65 integers. The loop is going to go until this number has been reached and the I is going to be a value that is smaller than the arrays size.

To appoint values to the positions that I want them to be in, I am going to use a different code.

Example

```
Ticket_prices [ I ] I + 1 ;
```

The values that I use are not going to be hard coded when they are placed

between the brackets, instead every variable is going to increase by one digit each time that the loop is repeated. The array is also going to increase with the loops value. This value is going to be assigned thanks to the last bit of the script that is entered. The loop is going to start at zero always and go up through the number 65 for this example.

The next bit of script will be the value for the positions in the array. In the event that I want to type out all my code, it is going to end up jumbled because of the amounts found in the array.

Sorting

The arrays I make can be sorted out by using the sorting method. I will however have to reference what I am sorting to a directory in Java. The import statement will be used for this.

Import java. util. Arrays ;

Example

```
Package prjarrays;
Import java . util. Arrays ;
Public class ArraysTest {
Public static void main ( String [ ]  args) {
Int [ ]  aryNums;
aryNums = new int [ 9 ] ;
aryNums [ 0 ] = 65 ;
aryNums [ 1 ] = 54;
```

```
aryNums [ 2 ] = 689 ;
aryNums [ 3 ] = 654;
aryNums [ 4 ] = 6543 ;
aryNums [ 5 ] = 55 ;
```

The libraries for an array are going to be imported along with the sort method.

```
Arrays . sort( aryNums ) ;
```

The keyword that falls first in the importing is array after that I am going to insert a dot. That dot is going to open up a box that is going to give me a list of everything that I can do with that array. The sort word is going to say where the name of the array is placed and then the brackets are going to fall after that.

Example

```
Package prjarrays;
Import java . util. Arrays ;
Public class ArraysTest {
Public static void main ( String [ ]  args) {
Int [ ]  aryNums;
aryNums = new int [ 9 ] ;
aryNums [ 0 ] = 202 ;
aryNums [ 1 ] = 1696 ;
aryNums [ 2 ] = 385 ;
aryNums [ 3 ] = 2956 ;
aryNums [ 4 ] = 4554 ;
aryNums [ 5 ] = 126 ;
Arrays . sort (aryNums) ;
```

```
Int I;
For ( I = 0 ; I < aryNums . length ; i++) {
System. out. printIn (“num: “ + aryNums [ I ] ) ;
}
}
}
```

Sorting is going to arrange things in a descending order when I write out my own code for sorting. I can also convert an array into objects that are used for integers that have also been imported from a directory.

Example

```
Package prjarrays;
Import java . util. Arrays ;
Import java.util. collections ;
Public class SortDescending {
Public class ArraysTest {
Public static void main ( String [ ]  args) {
Int [ ]  aryNums;
aryNums = new int [ 99 ] ;
aryNums [ 0 ] = 65 ;  aryNums [ 1 ] = 2415 ; aryNums [ 2 ] =  ;645
aryNums [ 3 ] = 263 ; aryNums [ 4 ] = 4005 ; aryNums [ 5 ] = 12245 ;
Arrays . sort (aryNums) ;
Int I;
For ( I = 0 ; I < aryNums . length ; i++) {
System. out. printIn (“num: “ + aryNums [ I ] ) ;
}
```

}
}

Array strings

Strings for integers can be placed in arrays as well.

Example

```
String [ ]  aryString= new String [ 6 ] ;
aryString [ 0 ] = “i” ;
aryString [ 1 ] = “am” ;
aryString [ 2 ] = “using” ;
aryString [ 3 ] = “java” ;
aryString [ 4 ] = “yay” ;
```

In this example, there are six positions needing filled with data. In this loop however, the positions will be displayed on the screen.

Syntax

```
Int I ;
For ( I = 0 ; I < aryString . length ; i++ ) {
System. out. printIn ( aryString [ I ] ) ;
}
```

A loop is going to continue up till a value of lesser amount is found than what the range calls for.

I can also sort arrays alphabetically. Java uses Unicode for the letters that are entered into the system. Letters that are capitalized are going to come before those that are not.

Example

```
Package strings1 ;
Import java. util. Arrays ;
Public class StringArrays {
Public static void main(String [ ]  args) {
String [ ]  aryString= new String [ 5 ] ;
aryString [ 0 ] = “I ” ;
aryString [ 1 ] = “have” ;
aryString [ 2 ] = “made” ;
aryString [ 3 ] = “an” ;
aryString [ 4 ] = “array” ;
arrays. Sort ( aryString ) ;
int I ;
for ( I = 0; I < aryString . length ; i++) {
system. out. printIn ( aryString [ I ] ) ;
}
}
}
```

Multi dimensional

The arrays that I have created can be put into multiple columns that are going to

be known as multi-dimensional arrays.

Example

60 – 1550 – 1285 -436 – 151 - 252
41 – 2035 – 4325 -5326 – 1231 - 2333
23 – 3240 – 6357 – 3225 – 1425 – 44352
335 – 40352 – 12321 – 8237 – 135214 - 53545
4654 – 50232 – 8699 – 6625 – 103 – 6656
5354- 62350 – 653 – 644 – 1632 – 161

Arrays that are multi dimensional are arranged in a certain fashion that is going to tell Java that it is a multi dimensional array over a regular array.

Syntax

```
Int [ ] [ ] aryAmounts = new int [ 8 ] [ 33 ] ;
```

The array syntax is like normal, but there are now two brackets that allow Java to know how many rows there are and how many columns will be made.

Example

```
aryAmounts[0][0] = 106;
aryAmounts[0][1] = 124;
aryAmounts[0][2] = 433;
aryAmounts[0][3] = 11554;
aryAmounts[0][4] = 262;
```

First the row is going to start with zero just like any other array, while the columns are going to be different for each array because each array is going to require a different number of columns.

Example

The second row for my array will be

```
aryAmounts[1][0] = 2560;
aryAmounts[1][1] = 43255;
aryAmounts[1][2] = 560;
aryAmounts[1][3] = 133;
aryAmounts[1][4] = 3385;
```

The positions in my loop is going to depend on the values I use. The first example uses the I variable while the second uses the same variable. Both are going to start at zero.

Lists

I cannot tell the exact values in an array but I can use the ArrayList so that I can look at the structure of the data. The items on my list can be added or deleted. But, a normal array is going to have data that does not move because I am not going to have the ability to change the arrays size for what I have made.

When I set up an array list, I am going to have to pull data from the directories in Java.

Syntax:

import java.util.ArrayList;

After I have brought in the data that is needed to make a new object, I am going to use a different syntax.

ArrayList listTest = new ArrayList( );

I am not going to use brackets when I create lists for my array.

Example

listTest.add("and");
listTest.add("for");
listTest.add("but ");
listTest.add(8);

Anything that is in the parentheses is going to be placed on my list for my array. I am going to be able to add any object that I want with this method. The first three items that fall on my list are going to be objects that are normally added to a string. The last amount is going to be an integer.

All of the items that are on my list can be found by their index values.

Example

listTest.get(5);

The number five tells me exactly where I am going to need to go to find what I am looking for. Index values are always going to start at zero just like other things that are used in Java. With this in mind, for the above example, I am going to look at position six.

When I want to delete objects, I am going to use a different syntax.
Example
listTest.remove(5);
I can also delete the string values on my array.
listTest.remove(“but”);
In removing items, the list is going to then resize itself. Because of this, I have to be very careful in deleting objects or else my lists index numbers are going to be changed dramatically and I am not going to know where anything is on my list. So, if I remove the second item on my list, then the next three items will be moved up on my list after it has been deleted.

Conclusion

Thank for making it through to the end of *Java*, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever it may be.

The next step is to download NetBeans and start creating your own Java programs with the information that I have provided for you. Some of it may seem confusing, but it is going to get easier the more that you practice it.

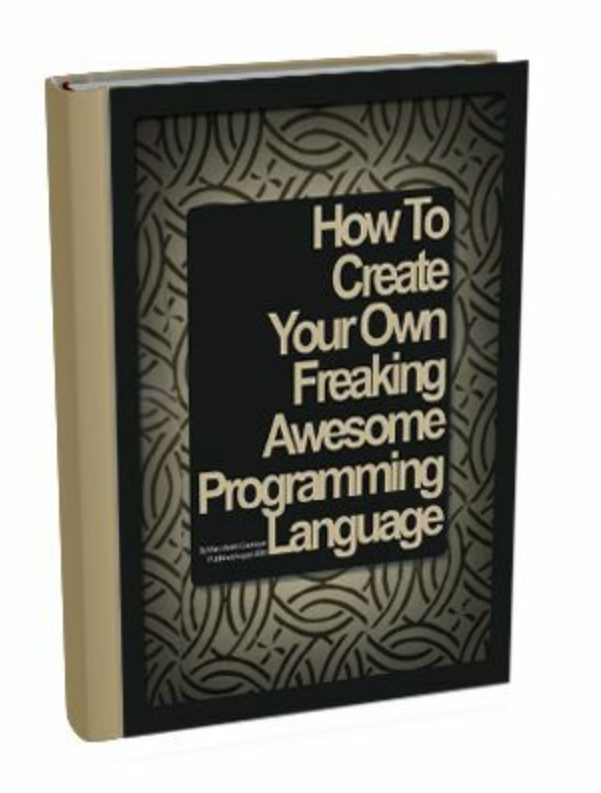
If you found any piece of this book confusing, go back and read it again. As I stated in the introduction, programming is not easy and it is not going to come to you overnight.

Do not stop learning Java just because one of the codes that you use does not work. If you do that, then you are never going to learn where you messed up. Instead, take that mistake and learn from it so that you do not have to repeat those same mistakes.

Finally, if you found this book useful in anyway, a review on Amazon is always appreciated!

Good luck!

Join the *thousands* of coders who created their own language.



“It's been a lot of fun, and surprisingly little work to get a language that covers 95% of what JavaScript can do.”

— Jeremy Ashkenas, created CoffeeScript after reading the book.

“I love this book! It helped me create my own Awesome-to-PHP compiler. It's not a full featured compiler yet but I had a lot of fun coding it and I learned a LOT.”

— Julien Desrosiers, created Phlower after reading the book.

“I never had a chance to study language internals at university, and while very interested, I found their inner workings intimidating. With this course I've found a simple but engaging introduction to the world of lexers, parsers, interpreters and compilers, and suddenly a whole new world in programming has opened up. Highly recommended.”

— David Bolton

A few languages created from this system ...

SavageRubby

It works or it's free.

Not sure this system is for you? Try it and if you're not happy, I'll give you your money back and you keep everything.

Get it here !

The follow eBook is reproduced below with the goal of providing information that is as accurate and reliable as possible. Regardless, purchasing this eBook can be seen as consent to the fact that both the publisher and the author of this book are in no way experts on the topics discussed within and that any recommendations or suggestions that are made herein are for entertainment purposes only. Professionals should be consulted as needed prior to undertaking any of the action endorsed herein.

This declaration is deemed fair and valid by both the American Bar Association and the Committee of Publishers Association and is legally binding throughout the United States.

Furthermore, the transmission, duplication or reproduction of any of the following work including specific information will be considered an illegal act irrespective of if it is done electronically or in print. This extends to

The information in the following pages is broadly considered to be a truthful and accurate account of facts and as such any inattention, use or misuse of the information in question by the reader will render any resulting actions solely under their purview. There are no scenarios in which the publisher or the original author of this work can be in any fashion deemed liable for any hardship or damages that may befall them after undertaking information described herein.

Additionally, the information in the following pages is intended only for informational purposes and should thus be thought of as universal. As befitting its nature, it is presented without assurance regarding its prolonged validity or interim quality. Trademarks that are mentioned are done without written consent and can in no way be considered an endorsement from the trademark holder.