



Java Foundations

Syntax Introduction

Copyright

Introduction to Java Programming by Eric Matthews is licensed under a Creative Commons Attribution 3.0 Unported License. This license allows you to

- Copy, distribute and transmit the work
- Adapt the work
- Make commercial use of the work

Under the following conditions:

You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to one or more of the following web pages.

www.DeveloperGeekResources.com

www.EducationAnytime.com

Table of Contents

<i>Writing Some “Hello World” Programs</i>	5
Java is a case sensitive language	5
Bare bones program (for our use now).....	5
Literals.....	6
Compiling your program	6
Running your program	6
Hello The Java Way	7
Variables.....	7
Concatenation operator.....	8
Put into practice – first java program	9
Basic operators	9
Conditional logic	10
if	10
A quick sidebar on indenting	11
If else	12
Nested if	12
if else if	13
switch (Case logic)	14
Put into practice – conditional logic	16
Time to talk about String.....	16
Declaring and initializing a String	17
Determining the length of a String	17
String case conversion	17
Trim	17
String to a character array	17
Another way to declare and use String	18
StartsWith method (and EndsWith method)	18
Concatenation	19
Arrays in Java	19
Initializing and array	20
Origin of an array	20
Using a variable to determine array size	20
Loops	21
for	21
Increment operator	22
Decrement operator	23
while	23
do	23
nested loops	24
Put into practice – loops and other stuff we have covered up until now	24

Passing Command Line Arguments Into Your Program.....25

Comment codes in Java.....26

Single line comments**26**

Multiple line comments.....**26**

A final word on this subject.....26

WRITING SOME “HELLO WORLD” PROGRAMS

We are going to start learning Java by covering the following subjects:

- Variables/Literals
- Conditional Logic
- Loops
- Arrays
- Compilation
- Running your program

My intent is to give you enough familiarity and practice with the syntax that you are acclimated enough to hit the next lesson, which is “First Steps in Object Oriented Programming”. I have taken this approach in courses I have taught on structured programming. I got my audience initially comfortable with the syntax so I could help them better focus on what I considered to be mission critical subject...namely writing subroutines and functions and achieving reuse in their code (if only on a personal level).

Since this approach has brought me success in the past I want to employ it again, though applied only to those subjects that are mission critical in learning, and more importantly, thinking in Java. So, your mission here is to get comfortable with some basic Java syntax so that I will have your full attention for the next subject.

We are not going to get into all the intricacies of these topics in this chapter. We are going to build a foundation so that you can begin to write a series of programs that I like to refer to as “hello world” programs.

I have also decided for a variety of reasons to expose you to the language though console (command line) programming. One of the reasons I choose this method is that I wanted to show you Java in its purest form. As such you do not need to use and IDE like Netbeans or Eclipse. A text editor and the windows command prompt (the command line shell if you are using linux or some other posix variant) will suffice for the code in these lessons.

JAVA IS A CASE SENSITIVE LANGUAGE

REMEMBER THIS!

BARE BONES PROGRAM (FOR OUR USE NOW)

At this point I am not going to explain any of the code contained here. I will later on (I promise). For now, just understand that this is what you will need to write your programs.

```
class <class_name>
{
    public static void main (String[] arguments)
    {
```

```
        System.out.println();
    }
}
```

LITERALS

Source HelloNonJavaWay.java

```
class HelloNonJavaWay
{
    public static void main (String[] arguments)
    {
        System.out.println("Hello World");
    }
}
```

COMPILING YOUR PROGRAM

>**javac** Hello.java

REMEMBER *The name of the class must be the same name as your file, minus the file extension or your program will not compile!*

If your path is set up correctly you should be able to run the compiler from anywhere on your system. If you are not able to invoke the compiler you will need to set the path. The path statement will look something like...

```
path=d:\jdk1.3.1_02\bin
```

Note: *As you can see by the jdk reference above this was written long, long ago. You will likely be using a more recent version of the jdk. Also, you will not that all this toycode written long ago still works like a champ today.*

RUNNING YOUR PROGRAM

```
>java Hello
Hello World
```

NOTE: *When running your program you refer to the name of the class!*

Since we will be using the 'command prompt' there are a few things you need to know. On Windows NT systems the 'command prompt' defaults are less than optimal. On Windows 9x systems the 'command prompt' itself is less than stellar. Refer to the Powerpoint titled, "CommandPrompt.ppt" for some tips on optimizing the 'command prompt'.

HELLO THE JAVA WAY

Source Hello.java

```
public class Hello
{
    public void helloworld()
    {
        System.out.println("Howdy do");
    }
}
```

Source UseHello.java

```
class UseHello
{
    public static void main (String[] arguments)
    {
        Hello h = new Hello();
        h.helloworld();

        Helloh2 = new Hello();
        h2.helloworld();

        String s = new String();
        s = "foobar";
        System.out.println(s);
        System.out.println(s.length());
    }
}
```

VARIABLES

Java supports the following native data types:

Keyword	Type
char	16 bit unicode
boolean	true/false
byte	8 bit signed integer
short	16 bit signed integer
int	32 bit signed integer
long	64 bit signed integer
float	32 bit signed floating point numbers
double	64 bit signed floating point numbers

In Java most everything is an object. There are very few native data types. For now we are working with some of these native data types with one noted exception below.

Source vars1.java

```
class Vars1
{
```

```
public static void main (String[] arguments)
{
    String strng = "Hello World";
    char ch = 'Z';
    boolean bool = false;
    double dbl = 128.34;

    System.out.println(strng);
    System.out.println(ch);
    System.out.println(bool);
    System.out.println(dbl);
}
}
```

You may have noticed that String is not included in the table above. Technically “String” is not a variable, it is a class. We are not ready to yet cover this differentiation, but I felt that I could not leave the ability to create a string out of the discussion at this point. For now I want you to think of “String” as a variable. While this might offend Java purists that may read this, oh well. I had a choice of compromising education or fudging a little bit on the technical details of the language.

In analyzing the code keep in mind the following information.

- The equal sign is not an equal sign in Java it is an assignment operator.
- You should initialize variables that you declare (because it is good practice). There are times when variable assignment is absolutely required and times when they are not. However, if you always initialize your variables when you declare them then this is just one less detail to remember. I will not be covering these rules in this book!
- The char data type can only hold a single character. When initialized the value must be wrapped in single quotes.
- When initializing “String” the value should be wrapped in double quotes.
- A Boolean value is either true or false and is not wrapped in quotes when initialized.
- Integers are initialized to a value that is not wrapped in quotes.

Again, stick with these data types for now. We will cover variable usage in more detail later. We will also cover variable scope later as well.

Another quick note (again)... Java is a case sensitive language. If you have noticed I have began the names of my programs and my class names with a capital letter. This is a stylistic convention you will often see used in Java. It is not required and frankly, not one I am really fond of. But, who I am to go my own way on this!

CONCATENATION OPERATOR

Source Concat.java

```
class Concat
{
    public static void main (String[] arguments)
    {
        String strng = "Hello World";
        char ch = 'Z';
        boolean bool = false;
    }
}
```

```

double dbl = 128.34;

System.out.println("String: " + strng + "; char: " + ch + "; boolean: " +
bool + "; double: " + dbl);
}
}

```

The + sign serves as the concatenation operator in Java. This symbol also doubles as an arithmetic operator. The Java compiler understands the context of when it is used when it tokenizes your code.

PUT INTO PRACTICE – FIRST JAVA PROGRAM

Using three variables write a Java program that displays “firstname”, “lastname”, and “age”. Your output should look like...

```

C:\Programming\Java>java ans_first_prog
Firstname: Edward Lastname: Ucator
Full name: Edward Ucator
Age: 29.0
C:\Programming\Java>_

```

BASIC OPERATORS

Operator	Meaning
=	Assignment
==	Equals
!=	Not equals
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

I do not believe any further explanation of the table above is required.

CONDITIONAL LOGIC

Java supports if and case logic. “if” statements can be nested, and you can code “if else” logic. I have placed very little narrative in this section as very little should be required.

if

Syntax

```
if (<expression to evaluate>)  
{  
    <statement1>;  
    <optional statement2>;  
    <optional statementN>;  
}
```

The “expression to evaluate” is any expression that evaluates to a Boolean value. If the expression is true the statements get executed, if false they do not. There is nothing new here. What may be new to you is the syntax...

```
if ()  
{  
  
}
```

The braces are what’s referred to as a block. If you have programmed in C, C++, or Perl then you already know what a block is and you can move on. If you are a Cobol or Visual Basic programmer then this will be new syntax to you. A block is a very simple concept. It allows you to group a collection of one or more statements together for execution.

On a technical note, if you only have one statement then a block is not required. For example the following two pieces of syntax are equivalent.

```
if (<expression to evaluate>)  
{  
    <statement1>;  
}  
  
if (<expression to evaluate>)  
    <statement1>;
```

While the last “if” is syntactically correct it is good programming style to always include the block.

Source My_if.java

```
class My_if  
{  
    public static void main (String[] arguments)  
    {  
        String strng = "Hello World";  
        if (strng == "Hello World")  
        {  
            System.out.println(strng);  
            strng = "Goodbye World";  
        }  
    }  
}
```

```
        System.out.println(strng);
    }
}
}
```

If you have run the program above and analyzed it you will see that it is pretty straight forward. I would like you to make a copy of this program, change the expression (`strng == "Hello World"`) to (`strng = "Hello World"`). Try to recompile the program. Granted you will make many errors on your own. I did want you to see that substituting the assignment operator for the equal operator produces a compiler error. Remember the expression in an "if" statement must evaluate true or false. By changing the equal operator to an assignment operator you lose your test condition. Assignment is an explicit directive and therefore cannot be evaluated as true or false. I am pointing all this out for people that come from languages where "=" is the equals operator. Some languages allow "=" to be used for both assignment and equivalency. Java does not!

A quick sidebar on indenting

If you have never worked with a language that uses blocks please look at the following two code examples:

One-

```
class my_if
{public static void main (String[] arguments)
{String strng = "Hello World";
    if (strng == "Hello World")
    {System.out.println(strng);
    strng = "Goodbye World";
    System.out.println(strng);}}
```

Two-

```
class my_if {
    public static void main (String[] arguments){
        String strng = "Hello World";
        if (strng == "Hello World"){
            System.out.println(strng);
            strng = "Goodbye World";
            System.out.println(strng);}
    }
}
```

The first example is not real easy to read. It is easy to forget that others will be reading your code. You should strive to write your code in a manner that takes into account that others will be reading it.

Example two is much better. Example two is also the method that many code generators use to produce code. I find this method acceptable. I prefer to line up my opening and closing braces with one another. I find this method more readable when looking at code where there are a lot of nested if's and nested loops.

These two programs are identical to "My_if.java". They all compile and produce identical object code. There is no compelling reason that I can think of to write your program like example one. There are many good ways to write readable code. I am not laying claim to one way.

If else

Syntax

```
if (<expression to evaluate>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
else
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source My_ifelse.java

```
class My_ifelse
{
    public static void main (String[] arguments)
    {
        String strng = "Goodbye World";
        if (strng == "Hello World")
        {
            System.out.println("We executed the if");
            System.out.println(strng);
            strng = "Goodbye World";
            System.out.println(strng);
        }
        else
        {
            System.out.println("We executed the else");
            System.out.println(strng);
            strng = "Hello World";
            System.out.println(strng);
        }
    }
}
```

Nested if

Syntax

```
if (<expression to evaluate>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
    if (<expression to evaluate>)
    {
        <statement1>;
        <optional statement2>;
        <optional statementN>;
    }
}
```

Source My_nested_if.java

```
class My_nested_if
{
    public static void main (String[] arguments)
    {
        String strng = "Yes";
        String strng2 = "";
        if (strng == "Yes")
        {
            System.out.println("We executed the if");
            strng2 = "True";
            System.out.println(strng2);
            if (strng2 == "True")
            {
                System.out.println("We executed the nested if");
                System.out.println(strng);
            }
        }
    }
}
```

if else if**Syntax**

```
if (<expression to evaluate>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
else if (<expression to evaluate>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source My_if_else_if.java

```
class My_if_else_if
{
    public static void main (String[] arguments)
    {
        String strng = "No";

        if (strng == "Yes")
        {
            System.out.println("first if executes");
        }
        else if (strng == "No")
        {
            System.out.println("else if executes");
        }
    }
}
```

```
    }
    else
    {
        System.out.println("else executes");
    }
}
}
```

switch (Case logic)

Syntax

```
switch(<expression>
{
    case <case_constant1> :
        <statement1>;
        <optional statement2>;
        <optional statementN>;
        break;
    case <case_constant2> :
        <statement1>;
        <optional statement2>;
        <optional statementN>;
        break;

    case <case_constantN> :
        <statement1>;
        <optional statement2>;
        <optional statementN>;
        break;

    default:
        <statement1>;
        <optional statement2>;
        <optional statementN>;
        break;
}
```

Source Case_logic.java

```
class Case_logic
{
    public static void main (String[] arguments)
    {
        String day = "";
        int i = 5;

        switch (i)
        {
            case 1:
                day = "Monday";
                System.out.println(day);
                break;
            case 2:
                day = "Tuesday";
                System.out.println(day);
        }
    }
}
```

```
        break;
        case 3:
            day= "Wednesday";
            System.out.println(day);
            break;
        case 4:
            day = "Thursday";
            System.out.println(day);
            break;
        case 5:
            day = "Friday";
            System.out.println(day);
            break;
        case 6:
            day = "Saturday";
            System.out.println(day);
            break;
        case 7:
            day = "Sunday";
            System.out.println(day);
            break;
    }
}
}
```

There are some limitations to “switch”. It can only test for equality in the expression. Also, it only works with the char, int, byte, and short data types. Given these issues there are some limitations in using switch that you do not find using “if” logic. Finally, the case statements themselves are not blocks. But, you can code loops and conditional logic with a case statement (something not well documented or discussed).

The “break” keyword is used to exit the “switch statement once a “case” is true. Without break you would get linear code fall through until the next “break” was encountered. There certainly may be times when this is a desired outcome. For example, we may want to return the rest of the days of the week from our desired start date. The following code would achieve this affect.

Source Case_fallthrough.java

```
class Case_fallthrough
{
    public static void main (String[] arguments)
    {
        String day = "";
        int i = 5;

        switch (i)
        {
            case 1:
                day = "Monday";
                System.out.println(day);
            case 2:
                day = "Tuesday";
                System.out.println(day);
```

```
        case 3:
            day= "Wednesday";
            System.out.println(day);
        case 4:
            day = "Thursday";
            System.out.println(day);
        case 5:
            day = "Friday";
            System.out.println(day);
        case 6:
            day = "Saturday";
            System.out.println(day);
        case 7:
            day = "Sunday";
            System.out.println(day);
            break;
    }
}
```

Generally speaking you will not want fall through in your code.

PUT INTO PRACTICE – CONDITIONAL LOGIC

Write a simple program of your choosing that demonstrates your understanding of using conditional logic and anything else you have learned thus far. This is a free form exercise.

TIME TO TALK ABOUT STRING

Java has a char data type. This is not the most useful data type as it will only store one character. If you are coming from C or C++ you can take the char data type and create what is referred to as a character class to hold a string of a given size.

The “String” class allows you the equivalent to the character array in C or C++. Though we have not yet talked about object is Java I still want to note that even though you can use the string class like you would a variable it is not a variable. String is an object in Java.

The following is a program that covers some of the methods available in “String”.

Author’s note: In some of the examples below I am introducing you to methods (Java’s name for a subroutine), as well as to some code that includes topics we have yet to cover (like loops). This is a real chicken and the egg dilemma for me. My goal in this module is to present you with basic Java syntax. I also want to keep the integrity of this document so that you can use it as a reference later on. This is why I included the information on Strings contained below. For now I just want you to think of these as built-in subroutines.

Source Str1.java

```
class Str1
{
    public static void main(String[] args)
```

```
{
```

Declaring and initializing a String

```
String lastname = "Williams";
System.out.println(lastname);

// can change name
lastname = "Jones";
System.out.println(lastname);

// get a character from the string
System.out.println(lastname.charAt(3));
```

Determining the length of a String

```
// get length of string
int str_len = lastname.length();
System.out.println(str_len);
```

String case conversion

```
// try to convert to lowercase
lastname.toLowerCase();
System.out.println(lastname + " - notice nothing happens");

// convert lastname to lowercase via assignment
lastname = lastname.toLowerCase();
System.out.println(lastname);

// convert lastname to uppercase via assignment
lastname = lastname.toUpperCase();
System.out.println(lastname);
```

Trim

```
lastname = "  Jones  ";
System.out.println(lastname);
// trim a String
lastname = lastname.trim();
System.out.println(lastname);
```

String to a character array

Please note I have not yet discussed loops in Java. This is presented to keep things you can do with Strings together.

```
// dump lastname to a character array
char arr[] = lastname.toCharArray();
for (int i=0; i<arr.length; i++)
{
```

```
        System.out.println(arr[i]);
    }

}
}
```

Source Str2.java

```
class Str2
{
    public static void main(String[] args)
    {
        String lastname1 = "Williams";
```

Another way to declare and use String

```
String lastname2 = new String();
lastname2 = "Smith";
```

When you declare a String you should use this convention. As I said once before String is not a primitive data type it is a class in Java. A class gets instantiated into an object. String is bizarre in the sense that while it is an object you do not have to explicitly create it using the “new” keyword. In other words

```
String lastname1 = "Williams";
```

Is really interpreted by the Java compiler as

```
String lastname1 = new String();
lastname1 = "Smith";
```

It is my opinion that you should always use the “new” keyword when instantiating a String class. The reason for doing this will not be apparent to you at this point, but it will later on.

StartsWith method (and EndsWith method)

```
if (lastname1.startsWith("Wil"))
{
    System.out.println(lastname1 + " starts with Wil");
}
else
{
    System.out.println(lastname1 + " does not start with Wil");
}

if (lastname2.startsWith("Wil"))
{
    System.out.println(lastname2 + " starts with Wil");
}
else
{
    System.out.println(lastname2 + " does not start with Wil");
}
```

```
// the endsWith() methods works in a similar vane
}
}
```

Concatenation

Source Str3.java

```
class Str3
{
    public static void main(String[] args)
    {
        String firstname = new String();
        firstname = "Bernie";
        String lastname = new String();
        lastname = "Williams";
        String space = new String();
        space = " ";

        // concatenation
        String name = new String();
        name = name.concat(firstname + space + lastname);

        System.out.println(name);

    }
}
```

The methods provided in the String class are very nice and keeps one from reinventing the wheel for common functions.

One final thought. As you are learning this language someone might look over your shoulder and tell you that you should use StringBuffer instead of String. While this is true in many situations you do not need to worry about this at this time. You are too early in your journey learning Java for this to be an issue.

ARRAYS IN JAVA

A Java array once dimensioned cannot be re-dimensioned. If you need an array in Java than can be redimensioned you will use a Java class called Vector. Also, arrays in Java are classes, they are not native data types. Here are some basic examples for working with arrays. The code and comments in the code should tell the story without cause for additional narrative.

Source Arr1.java

```
class Arr1
{

    public static void main(String[] args)
    {
```

Initializing and array

```
// 3 ways to initialize an array

//1
int arr[];
arr=new int[3];
int x=24;
int y=25;
int z=26;
arr[0]=x;
arr[1]=y;
arr[2]=z;
System.out.println(arr[0] + " " + arr[1] + " " + arr[2]);
//determining how many elements an array has
System.out.println("arr.length is " + arr.length);

//2
int arr2[];
arr2=arr;
System.out.println(arr2[0] + " " + arr2[1] + " " + arr2[2]);

//3
int arr3[] = {1, 2, 3};
int i;
//looping through an array

for (i=0; i<arr.length; i++)
{
    System.out.println("arr[" + i + "] = " + arr[i]);
}
```

Origin of an array

```
//an arrays origin
System.out.println(arr2.getClass().getSuperclass());

}

}
```

Source Arr2.java

```
class Arr2
{

    public static void main(String[] args)
    {
```

Using a variable to determine array size

```
// declare array, size is determined by a variable
```

```
int i = 0;
int a_siz = 5;
int arr[];
arr = new int[a_siz];
int a_init = 0;

// initialize the array
for (i=0; i<arr.length; i++)
{
    arr[i] = a_init;
}

// print array
for (i=0; i<arr.length; i++)
{
    System.out.println("arr[" + i + "] = " + arr[i]);
}

// set some array elements and reprint
arr[1] = 5;
arr[3] = 17;

for (i=0; i<arr.length; i++)
{
    System.out.println("arr[" + i + "] = " + arr[i]);
}

/* while we can create an array size based on a variable
   we cannot change the size of the array once it has
   been created.
*/
a_siz = 6;
// arr[5]=7; //this line will result in an index
            //out of bounds error at runtime

    System.out.println(arr.length);
// see still has 5 elements

// Note: there is no way to expand the size of an array

}
}
```

Again, I am not going to offer any narrative as I do not believe any is needed.

LOOPS

Java offers a number of types of loops. I will demonstrate each of them.

for

Syntax

```
for (<loop initializer>; <loop test condition>; <increment loop>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source Forloop.java

```
class Forloop
{
    public static void main (String[] arguments)
    {
        int i;
        for (i=1; i<10; i++)
        {
            System.out.println("number is: " + i);
        }
    }
}
```

If you have programmed in C, C++, or Perl then you are familiar with this loop. If not then the syntax for this loop may look a bit different to you. This loop offers some pretty terse syntax. The first part of the loop is the initializer. In the program above we have declared a variable and set its value to one in the loop. The next statement is the test condition for our loop. The loop will execute until the value of “i” is 10. Finally, we need a mechanism for advancing the loop. We do this by applying the iteration operator (++) to our initialized variable. This is the common way writing the loop, though we could have also written the loop this way.

```
int i=1;
for (; i<10; )
{
    System.out.println("number is: " + i);
    i++;
}
```

In the loop above we initialized our variable when we declared it. We iterate the loop within the loop itself. This works the same as the first loop. We could even have written

Increment operator

i++

as

i = (i + 1)

If you were fuzzy on the iteration operator you should now understand its function based on the last statement. I added these code variations to the loop to help clarify how the for loop works as its syntax can be strange if viewed for the first time. I would strongly recommend that you code for loop’s the way I first presented as this is the accepted standard.

Note: The iteration operator can be used anywhere in your code. PS, as a note of trivia the language C++ got its name because it was an “iteration” of C. That and fifty cents won’t come close to buying you a latte.

Decrement operator

```
int = 9;

i--

//i is now 8
```

while

Syntax

```
while (<expression is true>)
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
}
```

Source Whileloop.java

```
class Whileloop
{
    public static void main (String[] arguments)
    {
        int i=1;
        while (i<10)
        {
            System.out.println("number is: " + i);
            i++;
        }
    }
}
```

This loop executes until the expression is false.

do

Syntax

```
do
{
    <statement1>;
    <optional statement2>;
    <optional statementN>;
} while (<expression is true>);
```

Source Doloop.java

```
class Doloop
```

```
{
  public static void main (String[] arguments)
  {
    int i=1;
    do
    {
      System.out.println("number is: " + i);
      i++;
    }
    while (i < 10);
  }
}
```

The difference between the “do” loop and the “for” and “while” loops is that the “do” loop will always execute at least one time.

nested loops

Source Nestedforloop.java


```
class Nestedforloop
{
  public static void main (String[] arguments)
  {

    int i;
    int a=0;
    for (i=5; i>=1; i--)
    {
      System.out.println("outer loop number is: " + i);
      for (a=i; a<5; a++)
      {
        System.out.println("inner loop number is: " + a);
      }
    }

  }
}
```

PUT INTO PRACTICE – LOOPS AND OTHER STUFF WE HAVE COVERED UP UNTIL NOW

A palindrome is a word or sentence that is spelled the same backwards or forwards. For example “level” is a palindrome. Write a program that can distinguish these and that works like the one depicted below.



```
D:\>java palindrome2 level
arr[0] = l l
arr[1] = e e
arr[2] = v v
arr[3] = e e
arr[4] = l l
We have a palindrome

D:\>java palindrome2 "get help"
arr[0] = g p
arr[1] = e l
arr[2] = t e
arr[3] = h
arr[4] = h
arr[5] = e t
arr[6] = l e
arr[7] = p g
We do not have a palindrome
```

In order to complete this workshop I need to show you how to pass command line arguments into your program.

PASSING COMMAND LINE ARGUMENTS INTO YOUR PROGRAM

Here is a simple program that demonstrates how to pass a string argument from the command prompt to your program.



```
D:\JavaCodeWIP>java arg hello
hello
```

Source Arg.java

```
public class Arg
{
    public static void main (String[] args){
        String str = "";

        if (args.length == 1)
        {
            str = args[0];
            System.out.println(str);
        }
        else
        {
            System.out.println("one argument is needed, only one");
        }
    }
}
```

Methods in Java can accept and return arguments. This will be covered later. For now I want you to understand that the main method accepts an argument that is actually a String array named args.

```
public static void main (String[] args)
```

In this instance (no pun intended) we are testing in our conditional based on the length of the array.

```
if (args.length == 1)
```

“length” is a method available in the string class that will tell us how many elements are in the array. As you can see we are testing that the array contains at least one. One final note, while not required by the compiler, if we do not code the String array as an argument in our main program we will receive an exception error at runtime. Hence, its use is required.

COMMENT CODES IN JAVA

Up to now the programs have been trivial enough that they did not require any comment codes. I do not want to introduce comment codes to you.

Single line comments

Syntax

```
// <comment>

// comment line

if (myvar > 30)    //comments can come after code
```

Multiple line comments

Syntax

```
/*
<comments can go anywhere in between>
*/

/*
this is the
type of comment code that
can span
multiple
lines
*/
```

A FINAL WORD ON THIS SUBJECT

You should now have enough familiarity with basic syntax of Java to move into the next subject. Make sure you are comfortable with this material. The next subject is going to expose you to the world of Object Oriented Programming (OOP). This is going to be a new way of thinking about programming for you so you want to be sure that you are comfortable with the subject matter we just covered.