19/11/18

**DISPLAYING HELLO WORLD**

* Methods in java contain a list of instructions you want it to perform.
* **What is system.periiod…? because bucky used this in his tutorial about system.out.println() method.**
* Println() is an already built in method that allows us to print a line out.

In system.out.println(), it means print it on our computer just because of the system before it.

* You can println() to a network, a database and many other things and also our system.

**VARIABLES**

* There is a huge difference between print() and the println() methods. The print() only prints out the value you want it to print but the println() displays the value first and then creates a new line, i.e. it will move the cursor to a new line or the next line below the printed value.
* The maximum value a byte can take is 127. A byte can have -128 to 127 as its variable value but not more than that. The byte does not mean it should store values in binary forms. Byte only takes integer values but not the value i.e. not 1’s and 0’s.
* The following is the maximum number each integer variable can take.

Byte (1 byte) – 2^8 = 256; 256/2=128; = -128 to 127(maximum value of byte)

Short (2 bytes) 2^16 = -32768 to 32767

Int (4 bytes) 2^32 =-2147483648 to 2147483647

Long (8 bytes) 2^64 = -9223372036854775808 to 9223372036854775807

* **OBSERVATION:** the float and long data types do not work unless you add L to long and add F to the float.
* **OBSERVATION:** the float and double have 7 and 15 precisions. This means that for the float, the numbers before and after the decimal point must not be more than 7. And the double must not have more than fifteen numbers before and after the decimals.
* **OBSERVATION:** even exponents are used when initializing a variable.

Example: double man=2.2e10;

* **OBSERVATION:** wow it’s possible to create negative values and have them printed the same way. Examples is int a = -29; this would also be printed as -29.

**QUESTIONS:**

1. How many precisions has the double and the float data types?

**ANS:** From what I saw in my java tutorial notebook, the double can have 15 significant figures. And significant figures simply mean to start from the first non-zero number from the left. Therefore, if a number has 15 significant figures or is rounded off to it then it means that such number has only 15 numbers, if you begin count from the first non-zero number to the fifteenth which is the last one.

The float on the other hand has 7 precisions of significant figures. That is, it can only have seven numbers when you count from the non-zero number from the left.

**Note: please note that my ide, which is netbeans, is taking 15 precisions to mean 16 numbers all together. And that seven precisions means 8 numbers after and before the decimal point. Also, when you are using a non-zero number note that the total numbers will become 7.**

**GETTING USER INPUTS**

* The scanner class takes in whatever the user types in using the keyboard and then stores the value in a variable.
* System.in means system input and the only input we have is the keyboard.
* Scanner name=new Scanner(System.in);

The first word scanner means that we are using the scanner here. The name is the variable. System.in stands for our computer input, and the only input available is the keyboard. So whatever the scanner takes in is stored inside the name variable.

* Scanner name=new Scanner(System.in);

**System.out.println(name.nextLine())’**

**According to Bucky:**

1. The first statement above finally creates the scanner which will now allow you to input whatever you wish to input.
2. The second code then prints out the input. And you would be wowed that what it prints is any datatype that you have entered.
3. The .nextLine means that you should pause till you get input from your scanner.

* **observation:** anywhere an object of scanner is used in a print and even in a testing or conditions, it creates a scanner just as **Bucky’s scanner**.

**23/11/2018**

**A BASIC CALCULATOR**

* I just discovered that the double can have the total number of digits before and after the decimal point to be sixteen and this is really stunning.
* Also, the double would print out in one decimal place if the answer is going to have just a number that will need to allow one to be carried to the other side. But it would print to sixteen decimal places if the last sum of the decimals would give a number that would not warrant one to be carried to the other side of the decimal number.

**MATH OPERATORS**

* If you wonder how to create a calculator that would accept all kinds of digit data types, then what you need is the double variable declaration for the answer.
* However, the answer you get when you add two integers would end in a decimal place of one as the case may be.
* Another name for modulus operator is remainder operator. According to my own understanding.
* When using modulus and the datatype for your answer is set to integer value, if the first number before the modulus is lesser than the one after the operator, then the answer will be just the first number. Already, we all know that the first number is lesser than the other. And I guess that this kind of situation is rarely seen.

**IF STATEMENT (conditional statement)**

* A typical example of where to use an **if statement** is in a website. There you would like a condition that says “if you are a boy then take to boys website and if a girl take to girls website.”
* Remember that there is nothing like !== but !=, which means not equal to.
* The if statement as we know is used to test conditions.
* The problem below won’t run as I desire. I want the Scanner to give the strangerName same name as myName but the condition will give me the else result after inputting the right strings as those of myName.

**import java.util.Scanner;**

**public class IfStatement {**

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

**Scanner collector=new Scanner(System.in);**

**String myName = "Onyejiaku Theodore Kelechukwu";**

**String strangerName;**

**System.out.println("Please enter your full name in Uppercase, that is the first letter in capital:");**

**strangerName=collector.nextLine();**

**if(strangerName==myName){**

**System.out.println("Yes you are authorised to use this computer");}**

**else{**

**System.out.println("You are not authorised to use this computer, you are not 'Theodore'");**

**}**

**}**

**}**

* However, the program above would correspond with the If itself when I use integer values.
* **Observation:** using the scanner to input the value of strangerName, even if correct would give me a negative result. But initializing the strangerName value to be that of myName would give a surprising positive result.

LOGICAL OPERATORS

* Logical operators are also called multiple condition testing operators to me.
* Double ampersand and the double pipes are used for multiple condition testing
* &&(double ampersands) and ||(double pipes)
* Please note that the if itself and not the else means positive, while the else means negative.
* Using the ampersand means that you are using the AND GATE. This would print positive if all the conditions are true. But would print false if just one of the conditions is zero or negative or if one of the conditions are not met.
* Using the double pipes means that you are using the OR GATE. And this means that your result would be positive if all or just one of the conditions is true. But would print false if all the conditions are not met. Hence, at most, only one condition must be true if there will be any positivity.
* There is also among this logical operators what is called the not operator. It is just an interjection mark (!).

Example:

If(!(a>0&&a<2)

**27/11/2018**

**SWITCH STATEMENT**

* Rather than using the if statement on and on and on, we rather use the switch statement.

Example: int age=4;

If(age=1){ blablablabla}

If(age=2){ blablablabala}

If(age=3){blablablablabla}

… and so on

* You don’t write the if statement ten times to test a certain variable as in the case of age above. What you should use is the switch statement.
* There is what is known as default and also add break after the default statement. The default is used when the variable is not having the proper values it should have, or maybe the value is higher or lower or not what is to be put as a **case** in the switch statement.

**WHILE LOOP**

* The while loop allows you to run a code a multiple of times but only write it once. So it saves you a bunch of codes.
* Recall that placing the print before and after the increment produces a different result. This can be understood better by drawing a flow-chart for it.

**USING MULTIPLE CLASSES**

* When a class is **public,** it means that the class is open to the public and any other thing.
* **Void** means that the method is going to do something but it is not going to return any data.
* When you try to run any java program it first searches for the main method, else other classes will not run. Hence there should be a main class for all other classes to run.
* Whenever you want to use a method of another class inside the main class, the first thing to is to create an object of that class, which you want to use its method, inside the main class.
* An object is created using the following syntax:

**ClassName objectName = new ClassName();**

* The method is called telling the object what function it is to perform.

Example: **classObject.alert();**

* Why we have the separator, a full stop or a period, is because there can be more than one method for a class.
* In the netbeans ide, remember that any class that you are adding to your lists of classes must be contained in the package which houses the main class and the rest of other classes.

USING METHODS WITH PARAMETERS

* An argument can also be called “extra parameter data”.
* An argument is any extra information that your method needs in order for it to work.

**MANY METHODS AND INSTANCES**

* There are two kinds of variables, they are **public and private variables.**
* **Public** variables are variables that can be used by any class. **Private** variables are variables that are used by only the methods of the class in which the variable is declared or initialized. Not even the object of its class which is inside the main class can use it.
* Private variables can be modified or accessed by only the methods of its class.

Example:

private String myName;

* Remember that methods can be made to return some kind of datatypes or return nothing, returning nothing means adding **void** to the left hand side of the method.
* Whenever you have a variable inside a class but outside all the methods of the certain class then this variable is called an **instance variable**.
* Truly, when a variable is declared private, it cannot be accessed by another class, not even the main class or its own object that is inside the main class. But this is not the case with a public variable.
* **NOTE:** Int a=5; int b;

A=b;

If you think that b becomes 5 then you are wrong. **Remember** that the assignment operator is used to assign values, and comes after the variable or operand which you want to assign a value. Therefore, the expression above does not assign the value of **a** to **b**, but it now overrides the original value of **a**. And **b** will still be without a value.

**4/12/2018**

**CONSTRUCTORS**

* A constructor allows you to initialize variables before you create an object.
* You can create many objects you want for a certain class.
* The syntax for any constructor is given below:

Public Classname (Parameter arguments){

Code here

}

* All you need to do is to input the argument into the object you will create inside the main class.

**5/12/18**

**NESTED IF STATEMENTS**

* Nested if statement is when you have an if else statement inside another if else statement.
* This means that the nested if else statement is to be placed inside the if or else of the first or parent if else statement.
* Example of nested if else statement is

Int age = 60;

If (age<50){

System.out.println(“You are young”)}

Else{

System.out.println(“You are old”)

If (age >75){

System.out.println(“You are REALLY old”)

Else{

System.out.println(“Don’t worry, you are not that really old”)

}

}

* Notice, from the example above, that the second if else statement is inside the else statement. That is what is called nested if else statement.
* **Nested if statement** **can also be called nested if else statement**, and it should not be confused with **nested else if statement**.
* It is stunning that you can create infinite nested if else statements. All you need to do is to nest it inside the if or the else or both of the main statement. And you can continue from the rest if you wish to create as many as possible.
* The way to know when and how to nest inside an if or else is to understand the condition specified.

Example: int age=30;

If(age > 30){

//action here

}

From the example above, you can decide to go on and create a nested if statement or if else statement inside the if that might have a condition of **if (age >50  
)** and you can continue to name ages that are greater than the main if. This is because the condition is that age must be greater than 30. The same goes to the else statement, in as much as the condition is less than 30, you can go ahead and create as many conditions such as **25, 20, 15, and so on** because these values are less than 30.

**ELSE IF STATEMENT**

* Else if statement is a way of given Java many options.
* With ordinary if statement, you give Java an option and the alternative.

Example:

Int age = 18;

If(age >18){

System.out.println(“You are an adult”);

}else{

System.out.println(“You are not an adult”);

}

From the example above, we gave a condition and we expect only one option, the positive or the negative result. But with else if we can give multiple options.

* This is done by adding to the middle of if and else something called **else if**.
* Among the multiple options that will be listed, the result would only be one out of all of them. Because the program will look only for the option that satisfies the condition.
* The if is the main test or condition and the rest are only the alternative options if the if is not met.
* Do not write like this > =, but like this >=. This is because it they mean different things.
* **Observation:** this kind of statements, that is if statement in conjunction with else if statement, only one action or result is to be performed even though certain conditions that might be true are satisfied.

**Example:**

Int a = 20;

If(a>20)

Else if (a>20 && a < 40)

From the example above, the one that will be performed, although both statements have a > 20, will have to be the **if.** This is because that is the first condition the program encounters and since the condition is true it will execute it and then end the program. Thus, even if the **else if** also has one of its condition met, the program will only choose one out of all the conditional statements, i.e. it will start from the one on top.

**6/12/18**

**CONDITIONAL OPERATORS (TERNARY OPERATORS)**

* Conditional operators allow us to perform the same action as the **if else statement,** but it is compact and small.
* From my java notes 7. A conditional operator is also called a “**ternary operator”.**
* The ternary operator has **three operands** which are

**(Boolean-expression) ? (expression1) : (expression2)**

* Example of ternary operator is

Int age=2;

System.out.println(age>1 ? “You are more than a year older” : “You are not yet more than one year”);

* From the example above, the conditional or ternary operator was used. Inside the parenthesis was a condition which should be a Boolean value, that is **“true” or “false”.**

The **?** is used to mean that if the condition is true that it should print the first expression, but if it is false that it should print the second expression. The **:** is used to separate the first expression from the second expression.

* Do not forget that the ternary operator or the conditional operator performs the same function as the if statement- positive and negative.
* **Note:** do not forger that using = in a condition would always give you an error. Rather use == which is the equal comparison operator, whereas the = is just an assignment operator.
* **NOTE:** the ternary operator does not necessarily require the parenthesis to surround its expressions. Therefore, you can choose to use the parenthesis or not.

**SIMPLE AVERAGING PROGRAM**

* Do not forget that System.in is our keyboard.
* When you have a while loop condition as

While ( counter < 10)

This means that this while loop is going to loop ten times if the counter is post-incremented, that is from 0 to 9 after initializing the counter to be 0.

**FOR LOOPS**

* The for loop takes in three statements or arguments:

1. The first one is where you want the loop to start
2. The second one is where you want the loop to end.
3. The third one is how much you want to increment it by.

* **Example:**

For(int counter =1; counter < = 10; counter++1){

System.out.println(counter);

}

The result of this program would be 1 to 10.

* Once it got to eleven, it stopped because it has been set to only execute ten times.
* **Observation:** my ide is just displaying answers in integer, even if it is in double or float.
* I used the for loop in an else if statement to create my dream program that can calculate the power of any program.
* Inside the if and some other else if statement are some conditions. But the main condition, which is the main brain behind everything is this for loop statement.

For(int counter = 1;counter <= powerNumber;counter ++){

result = result \* number;

}

With this, if the power or the powerNumber is 2 it will loop only twice, because the counter is one and then increments till two.

* Remember **do while loop.** It will loop continuously as long as the Boolean expression inside its while loop is true. Even the result will continue to increase but there will be a difference if the do while loop is set before the initialization. Therefore, the do while loop will not increase the result as long as it is set before the initialization variables.

Examples:

1. do { int a=0;

//initialization here and other codes

}while();

ii //initialization here

do {

//other codes here

}while();

**COMPOUND INTEREST PROGRAM**

* The formula for compound interest is :

A = P ( 1 + R ) ^ n

* Compound interest is used to determine how much you are going to make in the future considering your interest.
* Also, compound interest can also be used to determine the number of your Youtube subscribers and how much you are going to get in the future.
* You can also use it to calculate investments, mortgage payments, and many more.

**14/12/2018**

**DO WHILE LOOPS**

* Remember what the while loop did, it tested a condition and anytime the condition is true it executed the loop body. However, the **do while loop** executes the loop body before testing the condition.
* The syntax is

do {

//what you want the loop to do

} while(Boolean expression);

* **Did you know** that the body code would execute even if the condition is false? This is because, the do while loop performs its action before testing the condition.

**Example:**

Int counter = 15;

do{

System.out.println(counter);

counter++;

}while(counter<10);

The example above would print only 15. This is because the action is performed before the condition is tested.

* The do while loop has many applications in java.

**MATH CLASS METHODS**

* The math class has many methods. One of them is the one that was used in the compound interest formula. The **pow** method.
* To make use of this methods, all you need to do is to just type “Math.” and many methods would appear.
* **The common methods include:**

1. **abs**
2. **Ceil**
3. **Floor**
4. **Max**
5. **Min**
6. **Power**
7. **sqrt**

* **Absolute value:** One of the methods is Math.abs(variable). This gives how many numbers a certain number or variable is away from 0.

**Example:**

Math.abs(-26); would give 26, because there are 26 numbers away from 0, i.e -1 to -26.

* **Ceiling:** this is use to round up a number.

**Example:**

Math.ceil(7.4); this would give 8.

* **Floor:** this is just the opposite of **ceil.** It rounds down a number.
* **Maximum:** this takes only two arguments and gives out the maximum of the two arguments or variables.
* **Minimum:** this is just the opposite of **max.**
* **Power:** this takes in just two arguments, and then finds the power of the first number raised to the second.
* **Square root:** this takes in number or a variable and finds its square root.
* Notice that you do not have to create an object or any class to use this methods, this is because these methods are **built in methods**.
* The methods listed here are just the 7 seven most common methods used. Nevertheless, there are still many or several of them.

**12/15/18**

**RANDOM NUMBER GENERATOR**

* To use the Random class, one will have to import the Random class.

Example:

Import java.util.Random;

* After this has been done, then an object of class random is created the normal way an object is created.

Example:

Random dice = new Random();

* This is an example of the Random generator:

Example:

Number = dice.nextInt(6);

The six means that it can output one out of six values. And this numbers are from 0 to 6.

* Also, we can make the values become what we want it to be, in case of a dice game.

Example:

Number = 1 + dice.nextInt(6);

With the presence of 1 in this expression, the values that can now be displayed randomly are still six in total, but now from 1 to 6.

* Another example is

Number = 2 + dice.nextInt(10); This means that it will have ten values, but from 2 to 10.

**19/12/2018**

**INTRODUCTION TO ARRAYS**

* An array is a variable that stores multiple values.
* It is useful when you want to create a variable that has a bunch of related information.
* Array can only store one kind of datatype. That is, it cannot store both int and String or any other datatype.
* The syntax for an array is:

1. What type of array do you want
2. Name your array
3. Put square brackets – tells java that you want to create an array.
4. Inside the second bracket, put the number of values you want

Example:

Int myArray[] = new in[10];

* The starting index of your array begins with zero
* An array won’t be necessary if we have to type in the indexes of all the array variable. This method is called the “array initializer”.
* The syntax for array initializer is:

Int myarray[] = {1,2,3,4,5]

* The array initializer method is the most used when dealing with arrays.
* The first kind is used when a user needs to enter the values, example is a school grade result average. While the other is that the results are just there and do not need a user to start inputting them.

**CREATING ARRAY TABLE**

* When setting a counter to be lesser than the total number of values of an array, you can just use the .length property to do this.

Example:

For(int counter=0;counter<myArray.length;counter++)

Here the .lenght property means the total number of values the array contains. It is a built in property in java.

* \t is used to create a single space between texts or strings.

**SUMMING UP ELEMENTS OF AN ARRAY**

* this is all about finding the total or sum of all the elements.
* If you are using this kind this kind of initialization, then it means you will have to use the **for loop** to input your elements.

int array[] = new int[10];

* If it is not this but the one below, then you will have to just do the sum.

Int array[] = {1,2,3,4,5,6,7,8,9,10};

**23/12/18**

**ARRAY ELEMENTS AS COUNTERS**

* This involves making the elements of an array become counters.
* **I do not understand this topic.**

**ENHANCED FOR LOOP**

* This kind of **for loop** is different in the sense that it needs just two parameters.
* A new variable and the array in in question.

Example:

For(int x: myArray)

Please note that the array has already been declared before this.

* Example:

Int myArray[] = new int[7];

Int total =0;

For(int x: myArray){

total=total + x; //or total+=x;

}

From my own understanding, the x indicates the various elements. The myArray indicates the array we are working, because there can be many arrays. The total+=x; means to add the elements of the array to the total.

**28/12/2018**

**ARRAYS IN METHODS**

* Arrays can be placed inside methods.
* The array can be placed inside a method which would accept a non-primitive data type called array.
* The method will then use the enhanced for loop to add the value of the elements with any value of your choice.
* Next would be to print the elements to see the result.
* Note that the method was called inside the main method.
* **Observation:** using the enhanced for loop to manipulate the array does not override the initial values of the elements of the array.
* **Example of the this is:**

Public static void main(String[] args){

Int[] myArray = {1,2,4};

MyArrayChanger(myArray[]);}

Public static void myArrayChanger(int[] c){

For(int x: c) // Making the elements of the array become x and also ready for looping.

{ x = x +5; // as it loops through, it adds five to the value of every elements

System.out.println(x)//Prints out the new values of the elements of the array but does not override the initial values.

}

**29/12/2018**

**MULTIDIMENSIONAL ARRAYS**

* A multidimensional array is any array that has more than one row.
* A multidimensional array can be declared in many different ways, but the most common is the two usual ways.

1. The first is using the array initializer

Example: The array is initialized in the following way

Int[][] myArray = {{6,7},{9,67}}//This is a two dimensional array of 2 \* 2 size.

1. The second is using the other method

Example:

Int[][] myArray = new myArray[6][7]; //This is declaring an array

//With this you will have to start initializing the values

//one after the other

/\*NOTE: from the program above, the 6 and 7 means that it is an array with 6 rows and 7 columns.\*/

PS: DO NOT USE NUMBERS INSIDE THE ARRAY INDICATORS (the first brackets). THEY ONLY INDICATE THE SIZE AND MORE IMPORTANTLY THAT THIS IS AN ARRAY.

* Initialize the array above as

Int[0][4] myArray = 4;// Here, the value for the element in the first row and forth column is 4.

* The array elements are accessed using the following way:

**The first number is the row (which starts from 0). The second number is the index of the element in question**

MyArray[0][0] //The first 0 represents the first row and the other zero the first element

myArray[0][1] //The first 0 represents the first row and the 1 the second element

myArray[1][0] //The second row and its first element

myArray[1][1] //The second row and the second element

* A nested for loop can be used to display the 2by2 array above.
* The first loop would have a counter initialized to 0. And this would loop only for 2 times.
* The second loop would have a different counter which would also be initialized as 0. And this loop would loop for 3 times.

**Observations:**

1. **The first loop would make the whole loop loop for two times.**
2. **The second loop would loop for only three times.**
3. **The counter inside the first loop would also be used up in the second loop.**
4. **This kind of loop is actually interesting and really challenging.**
5. **However the manual method can also be used.**

* **Note:** just know that both the array and the elements start with 0 and this is the index position.

**TABLE FOR MULTI ARRAYS**

**VARIABLE LENGTH ARGUMENTS**

**8/1/2019**

**TIME CLASS**

* **Observation:** the methods of subclasses do not have static access modifier unlike those of the main class.
* there is a new method now known to me. The method is **String.format().** This method allows you to specify the formats of certain variables. The **return value** of this method is a **String.**

**Example:**

String.format(“%d”,number1);

* when a time format is in the 24 hour format you refer to it as the **military format.**
* **Remember** that private instance variables of any class can only be accessed by the methods of its class.
* there is also another time format known as the **regular/American format.**
* **Specifiers** in java from my own observation do not need decimal points unlike the **c language.** They just need two digits, the first to specify the width between the number and any other element, and the other digit to specify the number of decimal points it should have.
* **Observation:** please note that when using the **String.format()** in any method that will return it as a String value, use the System.out.println(); to display this method. This is because the parent method that uses this method will print nothing unless the System.out.println() is used with the method.

**17/10/19**

**PUBLIC, PRIVATE AND THIS**

* Private variables in subclasses means that only the methods of the class can access this type of instance variable.

Example:

Private number1 = 8; //A private instance variable in a subclass

Subclassobject.number1 = 9; //Printing this in the main class would cause an error because the variable is private and can only be used or modified by the methods of its class.

* Public on the other hand means that the methods and its objects inside the main class can access it or modify it.

Example:

Public number1 = 8; //A public instance variable inside a subclass

Subclassobject.number1 = 9; //The number1 variable has now been modified by an object outside its class.

* Constructors or methods cannot modify instance variables that already have been modified.
* **local variables** are variables that are found inside methods or methods of a subclass.
* To modify instance variables that already have been initialized, the keyword known as **this** is used.

Example:

Private int firstNumber = 45; //initiatlized instance variable

Public void changeNumber(int firstNumber){

**this.**firstNumber = 50; //Now the initialized instance variable has been modified using the **this** keyword.

}

* **Observation:** I observed that **global variables** can be modified by methods which surely can use them. However they can not be modified whenever they are treated as arguments. This means that introducing them as arguments means that the ones you are accepting are just counterfeits, when we already know that the real ones are inside our storage room. However, when the **this keyword** is introduced then we can change the real values. Therefore **this** is the password to mean that when we accept an argument that is a global variable, then we can change the value of the global variables.

**MULTIPLE CONSTRUCTORS**

* You can create as many constructors as possible.
* The only difference would be in the number of arguments and the datatypes of the arguments of the constructors.
* Therefore, the constructor that would be used would be determined by the number of arguments and datatypes of the arguments.
* When you have many objects of the same class, it is advisable to use different names for them. If not done, there would be an error. This is applied when there are many constructors that need to be called.
* Actually, it is really surprising to know that objects are variables. And might be the real variables indeed.
* A single object can not perform the same method constructor. Therefore, different objects would be needed for each constructor.
* **Observation:** accessing a variable, whether in a main class or in a sub class, using an object and the variable will cause an error.

**Example:** object.variable;

**However,** this would print the variable value or access the value of it when it is used inside the System.out.println() method.

**Example:** System.out.println(object.variable);

**SET AND GET METHODS**

**BUILDING OBJECTS FOR CONSTRUCTORS**

**TOSTRING**

* One of the uses of **“this”** keyword is that it replaces the value returned your build in method that is called toString().
* This means that **“this”** is used in place of the value of the toString() value inside another method. Or in other words, it is used in place of toString() method that should have been used inside another method.
* Example:

Public class(String name){ //Constructor

//Here **this** is used in place of the method called toString()

System.out.println(“The name is ”+**this**);}

//The built in method toString() which has **this** as another identity

Public String toString(){

return System.out.println(name);

}

* In the program above where this is used inside the constructor, it might interest you to know that the method toString() itself can be used in place of **this.** This means that the toString() has another identity which is **this.**
* The note above only applies when the **this** keyword is used inside a constructor. This means that **this** keyword when want to be used in place of the **toString()** method in another method except the constructor method will not work . So it cannot be used when it is inside any other place outside constructor.

**COMPOSITION**

**ENUMERATION**

* Enumerations are datatypes. They are however not primitive datatypes.
* Enums are not placed inside subroutines or methods.
* The definition for enum datatype is thus:

**enum (enum-type-name) {**

**//List of enum values**

**}**

* Just like when you want to create any variable, you must use their **datatype name**, e.g. **String, int, Boolean, etc.** you must also declare variables of enum types using the **enum-type-name.**
* The list of enum types inside the enum must be sparated by commas.
* Just like **constructors, enums** must be placed under the class, else there would be mistakes.
* **Example of an enum type:**

**enum Colors {RED, BLUE, YELLOW, GREEN, PURPLE, PINK}**

* By convention enum types must be in CAPITAL letters, but this is just to obey the formal style and not the syntax.
* Note that enum values are contsants. Therefore, it is right to call them enum constants.
* To access or use this constants, you must call them using their enum-type-name and a dot with the one you wish to use.

**Example:**

**Colors.RED, Colors.BLUE, Colors.PINK, etc.**

* Since enum values are constants, it is therefore necessary to create variables that can have this constants.

Like the one we have above, some variables could be:

Liverpool, Chelsea, Manchesterunited, etc.

* These variables must be declared with the enum-type-name so as to be able to use this constants.

Example: Color Liverpool;

Color Chelsea;

* These variables can now be initialized or can have the enum values or constants. The way to give them these constants is given below:

Liverpool = Color.RED;

Chelsea = Color.BLUE;

* You can print these variables with just the variable identifier inside the System.out.println() method.

**Example: System.out.println(Chelsea);**

**//The output would be “BLUE”**

* The syntax code for **enumeration** is written below:

Public class Whatever{ //Main class

Public enum nameOfEnumeration{ //Enumeration

NORTH,

WEST,

EAST,

SOUTH} //End of enum constants

Public static void main(String args[]){

//Programmers code

} //End of main method

}//End of main class

* Values that should have the value of the constants are declared and initialized in the following way:

NameOfEnumeration variable = NameOfEnumeration.North; //A variable has been declared as a datatype enum and initialized as North using the valid initialization NameOfEnumeration.whateverConstant

* **Note: constants** are variables that never change.