For loop , while loop , if , else if , switch statements are identical as c++

Scanner class for having input by user form console…to use v need to import this…

“import java.util.\*;”

Conditional operator works same as expected , no syntax change

For creating a object of any class, we need to follow this syntax:-

Example object1 = new Example(); //generally this syntax is followed

Here for Class Example and its object object1

In if statement n every other statement in java it is for Boolean value…so if(1) which should actually always execute since its 1 therefore should always execute, but it will give error since if statement expects an Boolean thing , such as a comparison which results to true or false or rather having a true or false variable , i.e . a Boolean data type variable .

For having random number generator , import java.util.Random class, and make an object of that class, for the random value we need to assign it to a int or something or directly use it or display it , by using nextInt() method of that Random class object.

Enhanced for loop|Or foreach loop:-

Int bucky[]={1,2,3,4};

Int temp=0;

For(int x : bucky){

Temp+=x;

}

Here x will posses value of the array its associate with the array

In java a method called toString is provided to us, what is really does is that it converts the int or float to String so that we can perform operations on it as we can do it on a string…

We can also override the default toString method so that what we can do is whenever we use object or this we can get it by this toString .toString method is usually overridden so that it returns string, i.e. that method returns string…and when this is used , it reference to the string when we used as %s, what we actually return is String.format() method and set things accordingly if we are overriding it.

we can use static member n method can also be called or used using dot operator using object of the class, like c++ off course it can be used or called with the class name but also with object is possible here.

In Java there is no thing called functions, its called methods which follows CamelCase but not like javaScript , as In in java the very first letter is small and then after it follows camelcasing , though its just a naming convention n not a rule but It should be followed to have good n readable code, i.e. good programming practice is followed. For class names first letter should be used as capital, a common convention.

There are some methods provided by java which can be invoked by the class directly without using the object, like Integer.parseInt(); method…etc.

For printing many methods are there in java, like println , print, printf..etc…they are used as:-

System.out.println(“Hello world”);

Here System is the class , which can be observed since it has first letter as capital, then out is its object and println() is the method, here println is for printing and moving the cursor to the next line.

We can also do normal print, and it will just print normally, printf will work as it does in C.

Final keyword is just like const keyword in c/c++, so as once it has assigned the value you cant change it, and you can/have assign it in ctor .

For GUI :-

//We need to

import javax.swing.JOptionPane;

//Here J , O and P should be in caps…

String testObj1 = JOptionPane.showInputDialog(“Enter first number”);

//what this will basically do is take input from a GUI box which more like a promt kinda thing…

//We can make it print or display in a GUI kinda thing by:-

JOptionPane.showMessageDialog(null , “The thing which is to be displayed” , “this will be shown in the title bar, i.e. like in cmd it says Command Promt , likewise” , JOptionPane.PLAIN\_MESSAGE);

//here the first argument says the displacement that how far it will be displayed, giving null to it puts it //at center…second parameter is the actual thing which is to be displayed , third para is as shown above //, and fourth parameter can have multiple different thing, but one at a time…

Using GUI which change in particular with the Operating system…:-

//using JFrame

Import java.awt.FlowLayout;

Import javax.swing.JFrame; //gives the minimizer , maximize, window features

Import javax.swing.JLabel;

P c tuna extends JFrame{

Private JLabel item1;

Public tuna(){

Super(“The Title bar”); //this statement is almost common when working with gui n JFrame

setLayout(new FlowLayout()); //this statement is almost common when working with gui n JFrame

item1= new JLabel(“this is a sentence”);

item1.setToolTipText(“This is gonna show up on hover”);

add(item1);

}

}

-------

StringBuffer is used when we need to take in space in a string, just as cin and gets…

Use is similar to as we do in normal object of any class, just that the ctor parameter contains the string which should be there in the StreamBuffer variable.

Difference between final , finally and finalize():-

Final is a specifier , which can be given to a class , method or object, if given to class , then it cannot be inherited, if given to method then that method cannot be overridden and if given to object , its value can not be changed after initializing it.

Finally is a block associated with the try catch block , i.e. after the catch block for the cleanup activities for the try catch block…

Finalize() is a method which is called by Garbage collector just before destroying an object to close or terminate the connection between say for example connection between the object and the database or say connection between object and network. It is used for cleanup activities for the object.

However, Java has a feature called "named breaks" in which you can name your loops and then specify which one to break out of. For example:

A: while (true) {

B: while (true) {

break A;

}

}

This code will not loop forever, because the break explicitly leaves loop A.

A non static method can use non static and static both members , but a static method **can only** use static members only.

In Interface, if we have some method declared with no specifier, java by default makes it public , so where we define it, we need to give it access specifier public and no less then that.

We can create compound statements in a class outside a function too, and that too can be static as well as non static…like:-

Class ex{

Int a; //not required to show this concept , but taken as usual

Ex(){ //not required to show this concept , but taken as usual

//ctor of ex…}

}

{System.out.println(“this is a non static compound statement”);}

Psvm(){

}

Static {Systme.out.println(“This is a static compound statement”);}

}

/////

Explanation :-

Here when parsing of class happens , at that time even before the main method is executed, the compound blocks are executed, no matter if they are before the psvm or after they are executed before psvm. From that static compound block is executed first and then the non static one is executed.

If more than one static or non static compound statements exist then , the order of execution among them is acc. To FCFS , i.e. acc. To order , as in who is used/defined /declared first would be executed first among them….

Import java.io.File;

IT does not create file or something, it just has functions which just say if it exists or not…

Firstly it needs to create an object of the class File and in parenthesis put the file name including path like in c++

Function like, exits() function returns true if it exists and false if not…

////in event listeners …we have to create a class whose object works as an handler to the things we ///have and the object is given as a parameter….

Exception Handling…

///

In a method I receive a general Object as parameter and I have to retrieve the name of the class

public void myMethod(Object o)

String className = o.getClass().getName();

...

}

//

public static void arraycopy(

Object src, int srcPos,Object dest, int destPos, int length

)

//

class TestArrayCopyDemo {

public static void main(String[] args) {

char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',

'i', 'n', 'a', 't', 'e', 'd' };

char[] copyTo = new char[7];

System.arraycopy(copyFrom, 2, copyTo, 0, 7);

System.out.println(new String(copyTo));

}

}

Output:caffeine

Simple example of Covariant Return Type

1. **class** A{
2. A get(){**return** **this**;}
3. }
5. **class** B1 **extends** A{
6. B1 get(){**return** **this**;}
7. **void** message(){System.out.println("welcome to covariant return type");}
9. **public** **static** **void** main(String args[]){
10. **new** B1().get().message();
11. }
12. }

//i.e, just the return type is different but still it can be overridden .