**1. Why is Java a platform independent language?**

Java language was developed in such a way that it does not depend on any hardware or software due to the fact that the compiler compiles the code and then converts it to platform-independent byte code which can be run on multiple systems.

The only condition to run that byte code is for the machine to have a runtime environment (JRE) installed in it.

----------------------------------------------------------------------------------------------------------------

**2. What do you understand by an instance variable and a local variable?**

Instance variables are those variables that are accessible by all the methods in the class. They are declared outside the methods and inside the class. These variables describe the properties of an object and remain bound to it at any cost.

All the objects of the class will have their copy of the variables for utilization. If any modification is done on these variables, then only that instance will be impacted by it, and all other class instances continue to remain unaffected.

For Example:

class Athlete {

public String athleteName;

public double athleteSpeed;

public int athleteAge;

}

Local variables are those variables present within a block, function, or constructor and can be accessed only inside them. The utilization of the variable is restricted to the block scope. Whenever a local variable is declared inside a method, the other class methods don’t have any knowledge about the local variable.

For Example:

public void athlete() {

String athleteName;

double athleteSpeed;

int athleteAge;

}

----------------------------------------------------------------------------------------------------------------

**3. Can we overload main method in Java?**

Yes, we can have multiple methods with name “main” in a single class. However if we run the class, java runtime environment will look for main method with syntax

as public static void main(String args[]).

----------------------------------------------------------------------------------------------------------------

**4. What are the access modifiers available in java?**

Java provides access control through public, private and protected access modifier keywords. When none of these are used, it’s called default access modifier.

A java class can only have public or default access modifier.

----------------------------------------------------------------------------------------------------------------

**5. How will you call a non static method from main method in java?**

Non static method are called by creating an Object for the class.

For Example,

public class StaticTest {

public static void main(String args[]) {

// Create Object for the class

StaticTest test = new StaticTest();

// call the method

test.printName();

}

public void printName(){

System.out.println("TestLeaf");

}

}

----------------------------------------------------------------------------------------------------------------

**6. Why String is immutable in java?**

Immutable means unmodifiable or unchangeable. Once string object is created its data or state can't be changed but a new string object is created.

For example:

class immutability{

public static void main(String args[]){

String s="Test";

s.concat(" Leaf"); //concat() method appends the string at the end

System.out.println(s);//will print Test because strings are immutable objects

}

}

From the above program, it is understood that two objects are created but "s" reference variable still refers to "Test" not to "Test Leaf".

But if we explicitly assign it to the reference variable, it will refer to "Test Leaf" object.

For example:

class immutability{

public static void main(String args[]){

String s="Test";

s=s.concat(" Leaf");

System.out.println(s);//will print Test Leaf

}

}

Now it prints "Test Leaf". However still "Test" object is not modified.

Suppose there are 5 reference variables, all refers to one object "Test". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

----------------------------------------------------------------------------------------------------------------

**7. What is the difference between concat() and concatenation operator "+" ?**

Both concat() and + are used to concatenate the strings but they have some differences which are described below:

**a) Java.lang.NullPointerException :**

concat() method if called on null String reference variable, will throw NullPointerException while this is not the case with + operator.

For example,

String s=null;

s.concat(“test”); will throw NullPointerException.

While

String s=null;

System.out.println(s+”test”); will print nulltest.

While concatenating strings with + operator, a null reference variable changes to “null” (i.e a String containing null as its contents.). That’s why s becomes “null” and output is nulltest.

**b) Number of arguments:**

concat() takes only one argument while + can take any number of arguments.

For Example:

"why"+"String"+"is"+"immutable"+"in"+"Java"

**c) Type of Argument:**

Signature of concat() method is: public String concat(String str)

concat() only takes String type as argument. If any other type argument is passed to concat() it will give compile time error.

For example:

s.concat(5); // error as 5 is a non string value (integer)

s.concat(“5”); // no error as here 5 is string (enclosed in “”)

While + can take any type of arguments. While doing concatenation non-string type argument is converted to String by using its toString() method

For example:

String s=”test”;

int i=5;

System.out.println(s + i); // no error, will print test5.

Here int value 5 is a primitive value. It will be first converted to a string value as:

int------>Integer.toString(5)-------> a String representation of int value 5.

**d) Creation of new String Object:**

Java Strings are immutable. Whenever you try to change its contents, a new String object with changed contents is created instead of modifying the original String object.

But concat() returns new String object only when the length of argument string is > 0.

String s=”Test”;

String s1=”Leaf”

String s2=s.concat(s1);

Here concat() returns a new String object whose reference is stored in s2.

But if the length of argument string is 0, then concat() returns same original string instead of returning a new String object. Example,

String s=”test”;

String s1=s.concat(“”);

Here “” (an empty String with length 0) is passed as an argument to concat(). So no new String object created but reference to original string s is returned and stored in s1.

Therefore, s==s1 will give true here i.e both s and s1 are pointing to same objects.

While,

+ creates a new String object every time it concatenates something, except when the concatenation is done at compile time.

For example:

String s=”Test”;

String s1=s +”Leaf”;

Here a new String object s1 is created with concatenated result (Test Leaf).

Similarly,

String s=”Test”;

String s1=”Leaf”;

String s2=s+s1;

Will create a new String object s2.

In both above example concatenation happen at run time.

But, String s= ”Test” + ”Leaf”;

Will not create a not create a new String object because here concatenation will happen at compile time.

There will be only one String object with contents “Test Leaf”. Both String literals “Test” and “Leaf” will be discarded.

**e) Performance:**

Performance wise concat() is consider better than + operator. One of the reasons for this is that concat() returns a new String object only when length of its arguments is > 0.

----------------------------------------------------------------------------------------------------------------