**Core java questions**

1. What is interface and why it is used ?
2. If interface can do the more advance work than class why we use class?
3. What is the different between class and interface?
4. What is the different between instance variable and class level variable?

5. what is JRE JVM and how it is related to oop concept?

6. what is heap memory, string pool memory, stack memory and it's difference ?

7. what is the reason of making a data/variable as private if we can excess it anyway by creating a method in another class ?

**Answers:**

1>Interface provides a contract for all the implementation classes, so its good to code in terms of interfaces because implementation classes can’t remove the methods we are using.

1. Interfaces are good for starting point to define Type and create top level hierarchy in our code.
2. Since a java class can implements multiple interfaces, it’s better to use interfaces as super class in most of the cases.

2>For any implementing feature we need class because using class we create object

that can apply actual coding in practical .

3>In interface we can not have object based on properties and its fully abstract in nature but with class we can have object based on properties and its implemented form.

4>A instance variable is always worked on object to object basis that means each object of class will have theire set of properties different from other objects of the class but if we have a class level variable then each object of the class will have to share that property rather than having seperate copy of each object as we have for instance variable.

5> JRE :-Java Runtime Environment (to say JRE) is an installation package which provides environment to only run(not develop) the java program(or application)onto your machine. JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.

Java Virtual machine(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line hence it is also known as interpreter.

6>

### Stack

* The size of the stack will vary as methods and functions create and delete local variables as needed.
* Memory is allocated and then subsequently freed without you needing to manage the memory allocation.
* Stack has size limits, which can vary according to the operating system you use.
* Variables that are stored on the stack exist for as long as the function that created them are running.

### Heap

* Memory is not managed automatically nor is it as tightly managed by the central processing unit the way stack is managed. You would need to free allocated memory yourself when these blocks are no longer needed.
* The heap is prone to memory leaks, where memory is allocated to unused objects and will not be available to processes other than that.
* There is no size limit in the heap.
* Compared to stack, objects in the heap are much slower to access. It is also slower to write to the memory on the heap.

Stack is easier and faster to use, but it comes with a lot of limitations that you can ignore if you use heap.

When do you use stack?  Stack can only be used for local variables that use up small amounts of memory. The good news is that memory allocation and management is not going to be your problem and access to these objects is very fast. It does suffer from size limitations and the fact that you cannot resize variables on the stack.

When do you use heap? You use the heap to allocate memory if there are variables that you need to be accessed globally, as opposed to just being available only to the methods and functions that created it. Heap is also good when you have a need for a lot of memory since it has no limit on memory size. You can also resize the variables on the heap.

We all know that JVM divides the allocated memory to a Java program into two parts. one is Stack and another one is heap. Stack is used for execution purpose and heap is used for storage purpose. In that heap memory, JVM allocates some memory specially meant for string literals. This part of the heap memory is called String Constant Pool.

Whenever you create a string object using string literal, that object is stored in the string constant pool and whenever you create a string object using new keyword, such object is stored in the heap memory.

For example, when you create string objects like below, they will be stored in the String Constant Pool.

|  |  |
| --- | --- |
|  | String s1 = "abc";    String s2 = "xyz";    String s3 = "123";    String s4 = "A"; |

And when you create string objects using new keyword like below, they will be stored in the heap memory.

|  |  |
| --- | --- |
|  | String s5 = new String("abc");    char[] c = {'J', 'A', 'V', 'A'};    String s6 = new String(c); |

When you create a string object using string literal, JVM first checks the content of to be created object. If there exist an object in the pool with the same content, then it returns the reference of that object. It doesn’t create new object. If the content is different from the existing objects then only it creates new object.

7>One can not access directly the data member ,it has to be by some method wheteher it is by same class or other class.This is part of OOP principal so we have to

go by it.