1. What is JVM and explain me the Java memory allocation

1. Java Platform has 2 components JVM and Java API

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. There are three notions of the **JVM**: specification, implementation, and instance. The specification is a document that formally describes what is required of a **JVM** implementation.
JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

1) Classloader
Classloader is a subsystem of JVM that is used to load class files. Class loaders are hierarchical. Classes are introduced into the JVM as they are referenced by name in a class that is already running in the JVM. So, how is the very first class loaded? The very first class is especially loaded with the help of static main( ) method declared in your class. All the subsequently loaded classes are loaded by the classes, which are already loaded and running. A class loader creates a namespace. All JVMs include at least one class loader that is embedded within the JVM called the primordial (or bootstrap) class loader.

Bootstrap (primordial) :

Loads JDK internal classes, java.* packages.

Extensions :

Loads jar files from JDK extensions directory

System :

Loads classes from system classpath

2) Class(Method) Area
Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap
It is the runtime data area in which objects are allocated.

Java Garbage collector is responsible for reclaiming memory from dead object and returning to Java Heap space.

4) Stack
Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.
Each thread has a private JVM stack, created at the same time as thread.
A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.Stack which is used to store call hierarchy and local variables.
5) Program Counter Register

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack
It contains all the native methods used in the application.

7) Execution Engine

- 1) A virtual processor
- 2) Interpreter: Read bytecode stream then execute the instructions.
- 3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU

2. **What is Polymorphism and encapsulation?**

ONE LINE ANSWER : Polymorphism is the ability of an object to take on many forms. It means the ability of a single variable of a given type to be used to reference objects of different types, and automatically call the method that is specific to the type of object the variable references. In Java this is achieved through Overloading and Overriding.
Encapsulation in java is a *process of wrapping code and data together into a single unit* by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of fully encapsulated class.

3. **What is method overloading and Method over riding?**

Overloading : Compile-time Polymorphism Method overloading is performed *within class*. There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

In java, Method Overloading is not possible by changing the return type of the method.
You can have any number of main methods in a class by method overloading.
One type is promoted to another implicitly if no matching datatype is found.

Ex: The char datatype can be promoted to int,long,float or double and so on.
obj.sum(20,20);//now second int literal will be promoted to long
Overriding: **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable. This is called Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

NOTE:

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Ex: When reference variable of Parent class refers to the object of Child class, it is known as upcasting.

Method is overridden , not the datamembers, so runtime polymorphism can't be achieved by data members.

4. **Why string is Immutable?**

ONE LINE ANSWER : BY DESIGN STRING CLASS IS FINAL….
If string is not immutable, changing the string with one reference will lead to the wrong value for the other references. a string object is a **variable**; a string literal is a **constant** (a fixed sequence of characters between quotation marks) and can be *interned* from constants pool. However objects created in heap are immutable due to :

- **Security**: parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. If it were mutable, these parameters could be easily changed.
- **Synchronization** and concurrency: making String immutable automatically makes them thread safe thereby solving the synchronization issues.
- **Caching**: when compiler optimizes your String objects, it sees that if two objects have same value (a="test", and b="test") and thus you need only one string object (for both a and b, these two will point to the same object).
- **Class loading**: String is used as arguments for class loading. If mutable, it could result in wrong class being loaded (because mutable objects change their state).

5. What is the difference between String and String buffer?

String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.

String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Note: StringBuffer is *synchronized* and so less efficient

6. What is the difference between array and array list?

Java ArrayList class uses a dynamic array for storing the elements.It extends AbstractList class and implements List interface.

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- ArrayList is internally implemented using an array
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Differences

- ArrayList is type safe because it supports generics
- length unchangeable
- For index-based access, both ArrayList and array provides **O(1)** performance but add can be **O(logN)** in ArrayList if adding a new element triggers resize, as it involves creating a new array in background and
- ArrayList because *array allows storing both object* and *primitives and through autoboxing*
- array can be multi-dimensional e.g. you can have a two-dimensional array or a three-dimensional array, which makes it a really special data structure to represent matrices and 2D terrains. On the other hand, ArrayList doesn't allow you to specify dimension.
- Default size of ArrayList is 10, array 0
- primitive data types vs Objects

7. What is the difference between hash map and Hash table?

1. Synchronization or Thread Safe : This is the most important difference between two . HashMap is non synchronized and not thread safe.On the other hand, HashTable is thread safe and synchronized.

2. Null keys and null values : Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.

3. Iterating the values: Hashmap object values are iterated by using iterator .HashTable is the only class

other than vector which uses enumerator to iterate the values of HashTable object.

4. Fail-fast iterator : The iterator in Hashmap is fail-fast iterator while the enumerator for Hashtable is not.
If the Hashtable is structurally modified at any time after the iterator is created in any way except the iterator's own remove method , then the iterator will throw ConcurrentModification Exception.
Structural modification means adding or removing elements from the Collection object (here hashmap or hashtable) . Thus the enumerations returned by the Hashtable keys and elements methods are not fail fast.We have already explained the difference between iterator and enumeration.

5. Performance : Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized . Unsynchronized objects are often much better in performance in compare to synchronized object like Hashtable in single threaded environment.

6. Superclass and Legacy : Hashtable is a subclass of Dictionary class which is now obsolete in Jdk 1.7 ,so ,it is not used anymore. It is better off externally synchronizing a HashMap or using a ConcurrentMap implementation (e.g ConcurrentHashMap).HashMap is the subclass of the AbstractMap class. Although Hashtable and HashMap has different superclasses but they both are implementations of the *"Map"* abstract data type.

8. **What is a vector in Java?**

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity.	Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5) ArrayList uses **Iterator** interface to traverse the elements.	Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also.

9. What is set in java?
Set contain only unique elements while List can contain duplicate elements.
Set is unordered while List is ordered . List maintains the order in which the objects are added .

Class implementing Set interface : HashSet , TreeSet
Note :
a. HashSet maintains the inserted elements in random order while TreeSet maintains elements in the sorted order
b. HashSet can store null object while TreeSet can not store null object.
find detailed explanation here TreeSet vs HashSet in Java

10. What is an abstract class?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

11. What is an interface?
An interface in java is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

12. **Why Java is Platform independent?**

With Java, you can compile source code on Windows and the compiled code (bytecode to be precise) can be executed (interpreted) on any platform running a JVM. So yes you need a JVM but the JVM can run any compiled code, the compiled code is platform independent.

13. What are access modifiers? Give me an example?

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

14. What are java exceptions? Give me an example

Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. Example : unchecked exception nullpointer exception is an object which is thrown at runtime.

15. **What is the difference between throws and throwable?**

throw is a keyword in java which is used to throw an exception manually. Using throw keyword, you can throw an exception from any method or block. But, that exception must be of type **java.lang.Throwable** class or it's sub classes.
Throwable is a super class for all types of errors and exceptions in java. This class is a member of **java.lang** package. Only instances of this class or it's sub classes are thrown by the java virtual machine or by the throw statement. The only argument of catch block must be of this type or it's sub classes. If you want to create your own customized exceptions, then your class must extend this class.

16. **What is the difference between Error and exception?**

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory.
Where as exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

17. **What is the difference between Error, throwable and exception?**
Same as above

18. **What are collection APIs, give me an example**
Variety of Interfaces and classes for storing manipulating different group of data.

Ex: in List, the order is maintained and in Set unique data is maintained

19. **What is the difference between final and finally?**

Final is a keyword is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.

Finally is a block used to place important code, it will be executed whether exception is handled or not.

20. **Will java supports multiple inheritance?**

Java does not directly support multiple inheritence but supports indirectly through multi-level inheritence achieved through interfaces

21. **What are the different types of interface?** (Ans List, set, Queue)

Nested Interfaces: An interface can have another interface i.e. known as nested interface.
Marker or tagged interfaces: An interface that has no member is known as marker or tagged interface. For example: Serializable, Cloneable They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Examples of Collection Interfaces are List, Set etc

22. **What are wrapper class? Give me an example**
Wrapper classes are called so because they "wrap" the primitive data type into an **object** of that class.

- To include in a Collection
- To treat generically / polymorphically as an Object along with other Objects

23. **What is boxing and unboxing in Java?** Explain with an example
Autoboxing and **unboxing** feature converts primitive into object and object into primitive automatically.
The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.
Ex:-
```
Integer i=Integer.valueOf(a); // autoboxing
int i=a.intValue(); // auto unboxing
```

24. **Explain for each loop**
For each loop is a looping construct used to iterate a specified number of times .Below is example for looping controlled by value of variable

```
for( int i = 0;i<100;i++){
syso(i);
}
```

25. **What are iterators, explain with an example**
Iterators are objects that enable traverse a collection of elements. Ex : List Iterator, Iterator, Enumerator,

Example below is Iterator object to traverse HashMap

```
Iterator<Entry<String, Integer>> it = hashmap.entrySet().iterator();

     while (it.hasNext()) {

          Map.Entry<String, Integer> keyValuepair = (Map.Entry<String, Integer>)it.next();

                  System.out.print( " '"+keyValuepair.getKey()+"'  ");

          }

     }
```

26. **How do you access Private variables in different class?**
Through getter methods of the class

27. **Prepare for one java program to write on the board**

```
public class StringReverseProgramDemo{

public static void main(String[] args){
 String name = "Gopala";
char[] nameArray = name.toCharArray();
char[] temp = name.toCharArray();
for(int i = 0; i<nameArray.length;i++){
```

```
temp[i] = nameArray[nameArray.length-1-i];
}
for( char x :temp){
   System.out.print(x);
   }
}
}
```

28. **What is Constructor Over loading?**

This is used to initialize different types of members or different number of members of a class with the help of a parameterized constructor. Ex : when we do not create a default constructor for a class

29. **Whit out using sync key word how do you perform synchronization?**

Synchronization is not required and in place for single-threaded execution (making data local to the thread). Alternatively making all data immutable makes using keyword sync redundant.

30**. What is Super keyword ? when and where do you use it ?**

Super keyword is used to call the super class constructor or method from a child class. I t is used to invoke the parent class constructor from the child class or to call parent class method within the sub class method

The constructor in the superclass is responsible for building the object of the superclass and the constructor of the subclass builds the object of subclass. When the subclass constructor is called during object creation, it by default invokes the default constructor of super-class. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the keyword super, but it should be first statement in a constructor. The keyword super always refers to the superclass immediately above of the calling class in the hierarchy.

When calling super class constructor it should be the first statement in the constructor of the child class
However when calling super class method , it can be placed anywhere in the child class method