Core Java Interview Questions

Contents

Generic Questions

1. ***What are the key concepts of Object Oriented Programming?***

a> Abstraction

"Showing the essential and hiding the non-Essential" is known as Abstraction.

E.g:

- For an average car driver, car is a single thing. He is not concerned about the individual sub-systems of the car – braking systems, acceleration systems etc.
- For an advanced car user (who looks under the hood), a car consists of individual sub-systems (braking etc.) but he doesn't need to know how the sub-systems actually work.
- For a mechanic, each sub-system is a complex system in its own right.

Thus, Abstraction helps us to manage complexity by hiding the complexity from the user and just exposing the functionality.

b> Encapsulation

Encapsulation is the mechanism of binding the data and the code that manipulates the data together.

E.g.:

Let us take a sub-system – transmission system.

The transmission system encapsulates thousand of bits of information like speed, surface type, acceleration etc.

However, as a user, I don't need access to all the information. What I need is a method to affect/change the data (say acceleration) and I have been provided with the gear shaft. Gear shaft is the interface provided to the user to affect the transmission system.

The user doesn't need to know the inner details of how it works as long as the interface works. Also each car manufacturer has the freedom to implement the transmission as per their standard – say a Ferrari vs. maruti 800. However, for the user it still is a stick – gear shaft.

Advantages*:*

1> Acts as a protective cover and prevents data and code from being arbitrarily accessed.
2> Provides a well-defined interface for accessing the functionality of the code and data.
3> Gives freedom to implement as we wish (in this case car manufacture).

However, it raises another question. How is abstraction different from encapsuclation – see the next question.

c> Polymorphism

Poly means many. Morphism means various shapes. This means one interface to be used for multiple similar actions. However, the exact action to be performed depends on the situation.

E.g:

A dog smells cat and barks but when it smells food, it salivates. So the same sense of smell creates different reaction depending on what is being smelled.

In Java, it is implemented through functions of same names in a class.

Advantages:

1> Reduce complexity by using same interface for related activities.
2> Compiler takes care of deciding on the actual. User is freed from making this selection manually.

d> Inheritance

The process by which one object acquires the properties of another object. It offers one big advantage i.e. Supports the concept of hierarchical classification.

Without hierarchy, each object would need to define all of its characteristics explicitly. However, by inheritance, each object would need to define only those characteristics that uniquely define itself. Others are assumed from objects higher than itself.

2. What is the difference between abstraction and encapsulation?

Abstraction:

"Showing the essential and hiding the non-Essential" is known as Abstraction. Abstraction can be achieved by interface .

Encapsulation:

The "Wrapping up of data and functions into a single unit" is known as Encapsulation. Encapsulation is the term given to "the process of hiding the implementation details of the object".

Few Points

- Encapsulation is accomplished in java by using Class. - Keeping data and methods that accesses that data into a singe unit
- Abstraction is accomplished in java by using Interface. - Just giving the abstract information about what it can do without specifying the back ground details
- Information/Data hiding is accomplished in java by using Modifiers - By keeping the instance variables private or protected.

3. What is polymorphism? Explain with an example from your project experience.

E.g: Massaging and enrichment of product information.

Many front office systems have different identifier for the product. This product identifier is present in the feed file that they send to us at the EOD.

However, before loading into our database we have to confirm with the identifier that is present in our datamart.

Say frontoffice1, sends ric and frontoffice2 sends gss and we need to convert into primeID before loading in our database.

So we have multiple functions and depending on the product identifier, fires different query in the database to convert.

4. What are aggregation, association and composition in Java?

Association is an 'Has-A' relationship. In Java you can think of any time a class has an member of of a different type, then there is an Association between them. **E.g:**

```
1.  public class Person
2.  {
3.     private final Name name;
4.     private Costume currentClothes;
5.
6.     //...
7.  }
```

In this case, the Person Has-A name and Has-A Costume, so there is an Association between Person and Name, and Person and Costume.

Aggregation and Composition are specializations of the Association relationship - they are both Has-A relationships, but the difference is the length of that relationship. In the above example, a Person probably has a single Name for the live of the Person, while the Person may have different Costumes depending on the situation. If this is the case then the relationship between Person and Name is Composition, and the relationship between Person and Costume is Aggregation.
Composition is an Association where the containing Object is responsible for the lifetime of the contained Object. To show this in Java Person might have a constructor like this:

```
public class Person
{
   private final Name name;
   public Person (String fName, String lName)
   {
      name = new Name(fName, lName);
   }
   //...
}
```

Typically there would be no 'setName(Name)' methods. The Person is responsible for creating the Name, and when the Person is destroyed then the Name should (usually) be as well.

Aggregation is an Association relationship where the Association can be considered the containing class 'Owning' the contained class, and the lifetime of that relationship is not defined. 'Owning' can be determined as a single-direction Association. In the above example, the Person Has-A Costume, but the Costume would not Have-A Person. More importantly the Person to Costume Association is transient. In Java you can usually tell this because there are setter methods, or other means of adding / replacing the contained Object. So for the Person class you might have:

```
public class Person
{
   private Costume currentClothes;

   public void setClothes(Costume clothes)
   {
      currentClothes = clothes;
   }

   //...
}
```

Inheritance

5. ***Why is multiple inheritances not supported in Java?***
   Java allows multiple inheritance by using classes and interfaces. What is not allowed is multiple inheritance of *implementation*.
   The short answer is because the language designers decided not to. Designers reasoned that adding MI **added too much complexity** to the languages while providing **too little benefit**. Java's creators wanted a language that most developers could grasp without extensive training.
   Some of the complexities are :

1> The number of places where MI is truly appropriate is actually quite small. In many cases, you may be able to use encapsulation and delegation.
2> Multiple implementation inheritance injects a lot of complexity into the implementation. This complexity impacts casting, field access, serialization, identity comparisons and probably lots of other places.
3> The problem is that the compiler/runtime cannot figure out what to do if you have a Cowboy and an Artist class, both with implementations for the draw() method, and then you try to create a new CowboyArtist type. What happens when you call the draw() method? Is someone lying dead in the street, or do you have a lovely watercolor?
4> In object-oriented programming languages with multiple inheritance, the diamond problem is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If D calls a method defined in A (and does not override the method), and B and C have overridden that method differently, then from which class does it inherit: B, or C?
5> It is also
   a. hard to understand
   b. hard to debug
   c. hard to implement, especially if you want it done correctly and efficiently

6. ***Then how does C++ does multiple inheritance ?***
   C++ requires the programmer to state which parent class the feature to be used is invoked from i.e. "Worker::Human.Age". C++ does not support explicit repeated inheritance since there would be no way to qualify which superclass to use (see criticisms). C++ also allows a single instance of the multiple class to be created via the virtual inheritance mechanism (i.e. "Worker::Human" and "Musician::Human" will reference the same object).

7. ***Define interface and abstract class. What's the difference between them and give one real time usage of both.***
8. ***Can we instantiate abstract class?***
   No, you can never instantiate an abstract class. Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because Java's approach to run-time polymorphism is implemented through the use of superclass references. Thus, it must be possible to create a reference to an abstract class so that it can be used to point to a subclass object.

9. ***Is it mandatory that any methods or variable needs to be abstract in the abstract class.***
   No, abstract classes are not required to have any abstract methods, they can simply be marked abstract for general design reasons. An abstract class may contain a full set of functional, integrated methods but have no practical use in its basic form, for example. In other words, they may require extension with

additional methods to fulfil a range of different purposes. If the purpose is not specified by abstract method signatures, the range of potential applications for subclasses can be very broad.

10. ***By default what is behavior of the method in an Interface.***
11. ***Can we have Final in interface?***
    No. Trying to declare an interface as final in Java results in a compilation error. This is a language design decision - Java interfaces are meant to be extendable.

Exception

12. ***What is the Exception Hierarchy?***

In Java, exceptions are objects. When you throw an exception, you throw an object. You can't throw just any object as an exception, however -- only those objects whose classes descend from Throwable. Throwable serves as the base class for an entire family of classes, declared in java.lang, that your program can instantiate and throw.
Throwable has two direct subclasses, Exception and Error.
Exceptions (members of the Exception family) are thrown to signal abnormal conditions that can often be handled by some catcher, though it's possible they may not be caught and therefore could result in a dead thread. Errors (members of the Error family) are usually thrown for more serious problems, such as OutOfMemoryError, that may not be so easy to handle. In general, code you write should throw only exceptions, not errors. Errors are usually thrown by the methods of the Java API, or by the Java virtual machine itself.
Exception subclasses represent errors that a program can reasonably recover from. Except for RuntimeException and its subclasses (see below), they generally represent errors that a program will expect to occur in the normal course of duty
Error subclasses represent "serious" errors that a program generally shouldn't expect to catch and recover from. These include conditions such as an expected class file being missing, or an OutOfMemoryError.
RuntimeException is a further subclass of Exception. RuntimeException and *its* subclasses are slightly different: they represent exceptions that a program shouldn't generally expect to occur, but could potentially recover from. They represent what are likely to be programming errors rather than errors due to invalid user input or a badly configured environment.

13. ***What is difference between checked and unchecked exception? What is the intent for checked exception? And why can't I make it everything as unchecked?***

At compile time, the java compiler checks that a program contains handlers for checked exceptions. Java compiler analyzes by which checked exceptions can result from execution of a method or constructor.For each checked exception which is a possible result, the throws clause for the method or constructor must mention the class or its superclasses of that exception.

The class RuntimeException and its subclasses, and the class Error and its subclasses are unchecked exceptions classes. Because the compiler doesn't forces them to be declared in the throws clause. All the other exception classes that are part of Throwable hierarchy are checked exceptions.

Unchecked exceptions :

- Represent defects in the program (bugs) - often invalid arguments passed to a non-private method. To quote from The Java Programming Language, by Gosling, Arnold, and Holmes : "Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time."
- Are subclasses of RuntimeException, and are usually implemented using IllegalArgumentException, NullPointerException, or IllegalStateException
- A method is not obliged to establish a policy for the unchecked exceptions thrown by its implementation (and they almost always do not do so)

Checked exceptions :

- Represent invalid conditions in areas outside the immediate control of the program (invalid user input, database problems, network outages, absent files)
- Are subclasses of Exception
- A method is obliged to establish a policy for all checked exceptions thrown by its implementation (either pass the checked exception further up the stack, or handle it somehow) It is somewhat confusing, but note as well that RuntimeException (unchecked) is itself a subclass of Exception (checked).

Unchecked exceptions can occur anywhere in a program . Therefore, the cost of checking these exceptions can be greater than the benefit of handling them.Thus, Java compilers do not require that you declare or catch unchecked exceptions in your program code.Unchecked exceptions may be handled as explained for checked exceptions.

Compiler enforced catching or propagation of checked exceptions make it harder to forget handling that exception.
When methods do not declare what unchecked exceptions they may throw it becomes more difficult to handle them. Without declaration you cannot know which exceptions the method may throw. Thus you may not know how to handle them properly. Except of course, if you have access to the code and can see there what exceptions may be thrown from the method.

14. What is finally block and what is the statement u give in try catch block to make sure the program control doesn't reach the finally block.

System.exit is the only normal circumstance that causes a finally block to be skipped. Other circumstances are anything that causes the entire JVM to abort - errors in native code, operating system errors, hardware errors, that kind of thing. Also things you have absolutely no control over.

Static modifier in Java

15. What are static blocks in java?

Static blocks are also called Static initialization blocks . A static initialization block is a normal block of code enclosed in braces, { }, and preceded by the static keyword.

A class can have any number of static initialization blocks, and they can appear anywhere in the class body. The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code. This code will be executed when JVM loads the class. JVM combines all these blocks into one single static block and then executes.

Again if you miss to precede the block with "static" keyword, the block is called "constructor block" and will be executed when the class is instantiated. The constructor block will be copied into each constructor of the class.

16. What are the advantages and dis-advanatges of a static block?

Advantages

- If you're loading drivers and other items into the namespace. For ex, Class class has a static block where it registers the natives.
- If you need to do computation in order to initialize your static variables,you can declare a static block which gets executed exactly once,when the class is first loaded.
- Security related issues or logging related tasks

DisAdvanatges

- There is a limitation of JVM that a static initializer block should not exceed 64K.
- You cannot throw Checked Exceptions.
- You cannot use this keyword since there is no instance.
- You shouldn't try to access super since there is no such a thing for static blocks.
- You should not return anything from this block.
- Static blocks make testing a nightmare.

17. How to handle Exceptions in static blocks?

In methods, an exception can be handled by either passing through the Exception or handling it. But in a static block code, you cannot handle exceptions this way.

Generally a clean way to handle it is using a try-catch block but here since we dont have this option lets look at the available three options.

First: After logging the exception throw a RuntimeException which will end the current thread (unless caught by code instantiating / calling a static method on the class for the first time).

Second is calling System.exit(1) but this is not desirable in a managed environment like a servlet. This option is only for java applications and only if the static initializer block performs some critical (without which the program cannot be run successfully) function like loading the database driver.

Third and final option is to set a flag indicating failure. Later the constructors can check the flag and throw exceptions or retry in rare cases.

18. Given a scenario, one class has a Static block along with Public static void main method which one will get executed first?

Static block.

19. ***What is the use of static blocks, in which scenarios u go for static block?***

If you need to do computation in order to initialize your static variables,you can declare a static block which gets executed exactly once,when the class is first loaded.

20. What is the use of static methods in java?

- When you want to be able to access the method without an instance of the class
- If there is some code that can easily be shared by all the instance methods, extract that code into a static method
- If you are sure that the definition of the method will never be changed or overridden. As static methods can not be overridden

21. What are the constraints on static methods?

static methods cannot access instance variables. They can access only static variables.

Singleton

22. What is the concept of singleton in Java?

A singleton is an object that cannot be instantiated. That might seem counterintuitive - after all, we need an instance of an object before we can use it. So, yes a singleton can be created, but it can't be instantiated by developers - meaning that the singleton class has control over how it is created. The restriction on the singleton is that there can be only one instance of a singleton created by the Java Virtual Machine (JVM) - by prevent direct instantiation we can ensure that developers don't create a second copy.

23. So why would singleton be useful?

Often in designing a system, we want to control how an object is used, and prevent others (ourselves included) from making copies of it or creating new instances. For example, a central configuration object that stores setup information should have one and one only instance - a global copy accessible from any part of the application, including any threads that are running. Creating a new configuration object and using it would be fairly useless, as other parts of the application might be looking at the old configuration object, and changes to application settings wouldn't always be acted upon.

So to prevent direct instantiation, we create a private default constructor, so that other classes can't create a new instance.

We need to provide an accessor method, that returns an instance of the SingletonObject class but doesn't allow more than one copy to be accessed. We can manually instantiate an object, but we need to keep a reference to the singleton so that subsequent calls to the accessor method can return the singleton (rather than creating a new one). To do this, provide a public static method called getSingletonObject(), and store a copy of the singleton in a private member variable.

```
public class SingletonObject
{
   private SingletonObject()
   {
      // no code req'd
```

```
        }
    public static SingletonObject getSingletonObject()
  {
    if (ref == null)
       // it's ok, we can call this constructor
       ref = new SingletonObject();
    return ref;
  }

  private static SingletonObject ref;
}
```

24. What is the main challenge for an singleton object?

Multi threading

We need to make sure that threads calling the getSingletonObject() method don't cause problems, so it's advisable to mark the method as synchronized. This prevents two threads from calling the getSingletonObject() method at the same time. If one thread entered the method just after the other, you could end up calling the SingletonObject constructor twice and returning different values. To change the method, just add the synchronized keyword as follows to the method declaration :-

```
public static synchronized
          SingletonObject getSingletonObject()
```

Cloning

There isn't a clone() method defined in SingletonObject, but there is in the java.lang.Object class which it is inherited from. By default, the clone() method is marked as protected, but if your SingletonObject extends another class that does support cloning, it is possible to violate the design principles of the singleton. So, to be absolutely positively 100% certain that a singleton really is a singleton, we must add a clone() method of our own, and throw a CloneNotSupportedException if anyone dares try!

25. What is lazy loading and early loading?

One of the memory conservation techniques Java programmers find useful is *lazy instantiation.* With lazy instantiation, a program refrains from creating certain resources until the resource is first needed -- freeing valuable memory space. In this tip, we examine lazy instantiation techniques in Java class loading and object creation, and the special considerations required for Singleton patterns.

Lazy instantiation as a resource conservation policy. Lazy instantiation in Java falls into two categories:

Lazy class loading

Lazy object creation

Lazy class loading

The Java runtime has built-in lazy instantiation for classes. Classes load into memory only when they're first referenced. (They also may be loaded from a Web server via HTTP first.)

Lazy class loading is an important feature of the Java runtime environment as it can reduce memory usage under certain circumstances. For example, if a part of a program never is executed during a session, classes referenced only in that part of the program never will be loaded.

Lazy object creation

Lazy object creation is tightly coupled to lazy class loading. The first time you use the new keyword on a class type that previously hasn't been loaded, the Java runtime will load it for you. Lazy object creation can reduce memory usage to a much greater extent than lazy class loading.

26. ***In Singletons we use synchronized method to make it thread safe. How do we overcome for performance tuning***

If the singletons are immutable then they won't cause a decrease in performance.

However, if there are concurrent (synchronized) data structure singletons in the code then the performance will be decreased when compared to non-concurrent data structures.

Basically, think objects as data holders with some methods on them and now think that you have a linked list in your code and many threads are running concurrently and modifying that linked list. Analogous to this, the singleton data structure is data holder and many threads are trying to update it and the system is staying consistent with locks.

Hence, having singletons in concurrent systems do not make significant difference in performance unless they hold a data structure updated by multiple threads.

If you synchronize on any object, all threads will have to wait to obtain a lock on that object, before execution of the synchronized block of code. Other threads will queue up, waiting for the lock to be released. So, synchronizing on singleton instances or the singleton class will result in a decrease in performance. In simpler words, the performance decrease is a factor of how granular the lock is; if you synchronize on a singleton having n methods, all threads will have to wait on only one lock, even if they attempted to execute the n different methods.

Given, that we are using classes that were designed for concurrent access, this might not be a problem. This is better than attempting to reduce lock contention on the singleton. In tuning an application's use of synchronization, then, we should try hard to reduce the amount of actual contention, rather than simply try to avoid using synchronization at all.

27. ***Explain Singleton Pattern?***

Java has several design patterns Where Singleton Pattern is the most commonly used.

Java Singleton pattern define the instantiation process. This design pattern say that at any time there can only be one instance of a singleton (object) created by the JVM of class. This can achieved by class's default constructor is made private, which prevents the direct instantiation of the object by others.

Use

Singletons can be used to create a Connection Pool. In this scenario by using a singleton connection class we can maintain a single connection object which can be used throughout the application.

28. ***What are the problems we can expect with Singleton Pattern?***

- We can't really work with instance variables, since you always work with one and the same instance.
- Using singletons can also make code more difficult to test.
- It is not always possible to have a quick fix for it but simple thing like resetting the state to initial state of the singleton instance for each test run might fix it.

29. ***Can you define a singleton class as static? If Not then why?***

Code Samples

30. ***Given below the piece of code in a method as below***
    Sysout("hello");

    Return;

    Sysout("bye");

    What is the compile time error that u get at the second sysout.
    Code not reachable.

31. ***What is the output from the below code –***
    String s1 = "Hello world";

    s1.substring(3);

    sop(s1);

substring( ) has two forms.

The **first** is String substring(int *startIndex*). Here, *startIndex* specifies the index at which the substring will begin. This form returns a copy of the substring that begins at *startIndex* and runs to the end of the invoking string.

The **second** form allows to specify both the beginning and ending index of the substring: *String substring(int startIndex, int endIndex)*

32. ***What will happen in the below example when we return in try block?***
    try{ … return;}
    catch (exception){}
    finally{}

Finally will be executed.

Strings in Java

1. What is the difference between String s1 = "Hello"; and String s1 = new String("Hello"); ?

String objects are immutable.

E.g.:
String s1 = "Hello";
String s2 = s1;
// s1 and s2 now point at the same string - "Hello"

Now, there is nothing1 we could do to s1 that would affect the value of s2. They refer to the same object - the string "Hello" - but that object is immutable and thus cannot be altered.

If we do something like this:
s1 = "Help!";
System.out.println(s2); // still prints "Hello"

Here we see the difference between mutating an object, and changing a reference. You can change what it points to, but not that which it points at.

s2 still points to the same object as we initially set s1 to point to. Setting s1 to "Help!" only changes the reference, while the String object it originally referred to remains unchanged.

2. In the above example, what are the outputs for s1.equals(s2); and s1==s2

equals( ) method and the == operator perform two different operations. As just explained, the equals( ) method compares the characters inside a String object. The == operator compares two object references to see whether they refer to the same instance. The following program shows how two different String objects can contain the same characters, but references to these objects will not compare as equal:

```
// equals() vs ==
class EqualsNotEqualTo {
public static void main(String args[]) {
String s1 = "Hello";
String s2 = new String(s1);
System.out.println(s1 + " equals " + s2 + " -> " +
s1.equals(s2));
System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
}
}
```

The variable s1 refers to the String instance created by "Hello". The object referred to by s2 is created with s1 as an initializer. Thus, the contents of the two String objects are identical, but they are distinct objects. This means that s1 and s2 do not refer to the same objects and are, therefore, not ==, as is shown here by the output of the preceding example:

Hello equals Hello -> true
Hello == Hello -> false

One's program you may get true (depends on your jvm)

String s1 = "Hello";
String s2 = "Hello";
The big difference is that here we are not explicitly creating a new object , instead if I had used String s2 = new String ("Hello") just like in thread 1 then I will get s1 == s2 not true.

One more thing, a particular JVM can choose to maintain String pool while other may not.
String API just guarantees that equals() will work as intended but not == .

When JVM find two string object with de same content, in order to save memory, insted of create two objects, this two objects stay referenced to the same memory location.
I mean, if you have a string "Hello" already created, when you create a new string with the same word "Hello" JVM detect that already exist this string and then did not create a new object, JVM just make a reference the two objects to the same "Hello", so S1 and S2 are "pointing" to the same place in the memory, and S1 == S2 are true

Collections

3. What are collections

A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. Collection implementations in earlier (pre-1.2) versions of the Java platform included Vector, Hashtable, and array. However, those earlier versions did not contain a collections framework.

4. What Is a Collections Framework?

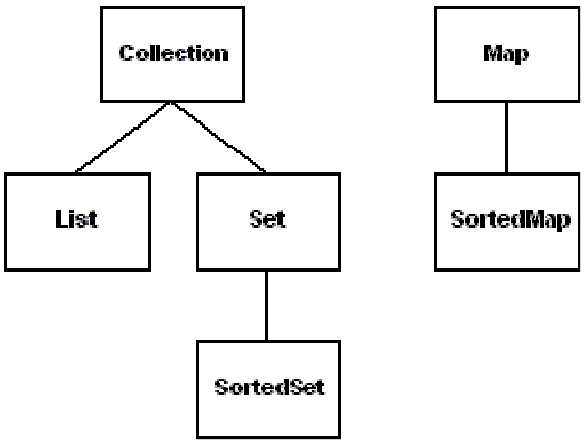
A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation.
- **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface

The Java Collections Framework provides the following **Advantages**:

- Reduces programming effort
- Increases program speed and quality
- Allows interoperability among unrelated APIs
- Reduces effort to learn and to use new APIs
- Reduces effort to design new APIs
- Fosters software reuse

The core collection interfaces are:



- **Collection** — the root of the collection hierarchy. A collection represents a group of objects known as its *elements*. The Collection interface is the least common denominator that all collections implement and is used to pass collections around and to manipulate them when maximum generality is desired. Some types of collections allow duplicate elements, and others do not. Some are ordered and others are unordered. The Java platform doesn't provide any direct implementations of this interface but provides implementations of more specific subinterfaces, such as Set and List.
- **Set** — a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the cards comprising a poker hand, the courses making up a student's schedule, or the processes running on a machine.
- **List** — an ordered collection (sometimes called a *sequence*). Lists can contain duplicate elements. The user of a List generally has precise control over where in the list each element is inserted and can access elements by their integer index (position).
- **Queue** — a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations.
- **Map** — an object that maps keys to values. A Map cannot contain duplicate keys; each key can map to at most one value. If you've used Hashtable, you're already familiar with the basics of Map.

The last two core collection interfaces are merely sorted versions of Set and Map:

- **SortedSet** — a Set that maintains its elements in ascending order. Several additional operations are provided to take advantage of the ordering. Sorted sets are used for naturally ordered sets, such as word lists and membership rolls.
- **SortedMap** — a Map that maintains its mappings in ascending key order. This is the Map analog of SortedSet. Sorted maps are used for naturally ordered collections of key/value pairs, such as dictionaries and telephone directories..

5. ***Difference between Set and Map?***
   A Set is a collection that has no duplicate elements.
   A map is a way of storing key/value pairs. The way of storing a Map is similar to two-column table.
   Set cannot have the duplicate value. But its unorder.
   Map: Group of key and value pair. The key unique one. No duplicate key.

6. ***Explain Hashing technique? What is collision in hashing ?how will u over come collision in hashing technique?***
   Hashing is a technique to make things more efficient by effectively narrowing down the search at the outset. Hashing means using some function or algorithm to map object data to some representative

integer value. This so-called **hash code** (or simply **hash**) can then be used as a way to **narrow down our search** when looking for the item in the map.

collisions: that is, what to do when there is more than one key with the same hash code (in this case, one than more string of a given length). For example, if our keys were random words of English, taking the string length would be fairly useless.

Now, we can solve the problem of **collisions** by having an array of (references to) **linked lists**[2] rather than simply an array of keys/values. Each little list is generally called a **bucket**.

Then, we can solve the problem of having an array that is too large simply by taking the hash code *modulo* a certain array size[3]. So for example, if the array were 32 positions in size, going from 0-31, then rather than storing a key/value pair in the list at position 33, we store it at position (33 mod 32) = 1. (In simple terms, we "wrap round" when we reach the end of the array.) So we end up with a structure something like this:

In practice, we combine these problems and define our task as coming up with a hash function that **distributes hash codes evenly over the entire range of possible integers** (in Java, that means a 4-byte number, or one with about 4 billion possible values)[2]. Then we assume that whatever the size of the array in our hash map (whatever the modulus), the mappings will be distributed more or less evenly.

7. How will you make sure that key returns unique hashcode in hashmap

The default hashCode() method uses the 32-bit internal Java Virtual Machine address of the Object as its hashCode. However, if the Object is moved in memory during garbage collection, the hashCode stays constant. This default hashCode is not very useful, since to look up an Object in a HashMap, you need the exact same key Object by which the key/value pair was originally filed. Normally, when you go to look up, you don't have the original key Object itself, just some data for a key. So, unless your key is a String, nearly always you will need to implement a hashCode and equals method on your key class. Object.hashCode in a native method.

8. What are factors that affect the performance the Hash Map?

An instance of HashMap has two parameters that affect its performance: *initial capacity* and *load factor*. The *capacity* is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. The *load factor* is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the capacity is roughly doubled by calling the rehash method.

As a general rule, the default load factor (.75) offers a good tradeoff between time and space costs. Higher values decrease the space overhead but increase the lookup cost (reflected in most of the operations of the HashMap class, including get and put). The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity, so as to minimize the number of rehash operations. If the initial capacity is greater than the maximum number of entries divided by the load factor, no rehash operations will ever occur.

9. Difference between linkedhashmap and hashmap

Both implement the Map interface and offer mostly the same functionality. Both represent mapping from unique keys to values, and therefore implement the Map interface The most important difference is the order in which iteration through the entries will happen:

- HashMap is a map based on hashing of the keys. It supports O(1) get/put operations. Keys must have consistent implementations of hashCode() and equals() for this to work.
- LinkedHashMap is very similar to HashMap, but it adds awareness to the order at which items are added (or accessed), so the iteration order is the same as insertion order (or access order, depending on construction parameters).
- HashMap makes absolutely not guarantees about the iteration order. It can (and will) even change completely when new elements are added.
- LinkedHashMap will iterate in the order in which the entries were put into the map
- HASH MAP has pair values(keys,values) NO duplication key values unordered unsortered it allows one null key and more than one null values HASH TABLE: same as hash map it does not allows null keys and null values
- LINKED HASH MAP: It is ordered version of map implementation Based on linked list and hashing data structures TREE MAP: Ordered and sortered version based on hashing data structures

10. LinkedHashMap vs. HashMap?

- A LinkedHashMap differs from HashMap in that the order of elements is maintained.
- A HashMap has a better performance than a LinkedHashMap because a LinkedHashMap needs the expense of maintaining the linked list. The LinkedHashMap implements a normal hashtable, but with the added benefit of the keys of the hashtable being stored as a doubly-linked list.
- ***What do generics do in Collections***

The motivation for adding generics to the Java programming language stems from the lack of information about a collection's element type, the need for developers to keep track of what type of elements collections contain, and the need for casts all over the place. Using generics, a collection is no longer treated as a list of Object references, but you would be able to differentiate between a collection of references to Integers and collection of references to Bytes. A collection with a generic type has a type parameter that specifies the element type to be stored in the collection.

E.g., consider the following segment of code that creates a linked list and adds an element to the list:

```
LinkedList list = new LinkedList();
list.add(new Integer(1));
Integer num = (Integer) list.get(0);
```

So, when an element is extracted from the list it must be cast. The casting is safe as it will be checked at runtime, but if you cast to a type that is different from, and not a supertype of, the extracted type then a runtime exception, ClassCastException will be thrown.

Using generic types, the previous segment of code can be written as follows:

```
LinkedList<Integer> list = new LinkedList<Integer>();
list.add(new Integer(1));
Integer num = list.get(0);
```

Here we say that LinkedList is a generic class that takes a type parameter, Integer in this case.

With generics, you achieve polymorphic behavior similar to the example above, but with strong static type-checking; the compiler knows that the two lists are different because they contain different elements, and these lists are guaranteed to contain only a homogeneous set of elements. The sample code below is a translation of the previous example, using generics this time. As you can see from comments in the code, all the errors are caught at compile time. Don't worry about the syntax for now -- we'll cover that shortly.

11. How does HashMap work internally? What needs to be done to make user specified object as key

A hashmap works:
It has a number of "buckets" which it uses to store key-value pairs in. Each bucket has a unique number - that's what identifies the bucket. When you put a key-value pair into the map, the hashmap will look at the hash code of the key, and store the pair in the bucket of which the identifier is the hash code of the key. **For example:** The hash code of the key is 235 -> the pair is stored in bucket number 235. (Note that one bucket can store more then one key-value pair).

When we lookup a value in the hashmap, by giving it a key, it will first look at the hash code of the key that you gave. The hashmap will then look into the corresponding bucket, and then it will compare the key that we gave with the keys of all pairs in the bucket, by comparing them with equals().

We see how this is very efficient for looking up key-value pairs in a map: by the hash code of the key the hashmap immediately knows in which bucket to look, so that it only has to test against what's in that bucket.

Looking at the above mechanism, we can also see what requirements are necessary on the hashCode() and equals() methods of keys:

- If two keys are the same (equals() returns true when you compare them), their hashCode() method must return the same number. If keys violate this, then keys that are equal might be stored in different buckets, and the hashmap would not be able to find key-value pairs (because it's going to look in the same bucket).
- If two keys are different, then it doesn't matter if their hash codes are the same or not. They will be stored in the same bucket, but the hashmap will use equals() to tell them apart.

12. If 2 objects are same, will they return same value of hashcode() or can it be different

13. How does LinkedList work internally

A LinkedList in Java works just as you would expect it to do. If you use the official Collections LinkedList then it will indeed be a *a bunch of object connected to each other by having a 'next' and sometimes 'previous'*.

It has a get(int index) method which is surprising because it would not be very efficient as we would need to start at the beginning and count our way up the list to find the indexth entry and this is not what LinkedLists are good at. The reason it is there is because LinkedList implements the List interface.

However, we should avoid using a LinkedList when most of your access to it is through the get(int index) method as that would clearly be most inefficient. We should use an ArrayList.

14. Can you implement add() and get() methods for LinkedList?

Yes we can.

15. Draw the hierarchical diagram of collection framework.

16. ***What all the collection classes/interfaces have you used in your project?***
17. ***Difference between ArrayList and LinkedList***

    Both are similar, though slightly different in terms of goal and implementation. In a LinkedList, each element is linked to its previous and next element making it easier to delete or insert in the middle of the list. An ArrayList is more as its name subjects used as an array.
    Performance is similar, though LinkedList is optimized when inserting elements before the end of the list, where ArrayList is optimized when adding elements at the end.

    If you frequently add elements to the beginning of the List or iterate over the List to delete elements from its interior, you should consider using LinkedList. These operations require constant-time in a LinkedList and linear-time in an ArrayList. But you pay a big price in performance. Positional access requires linear-time in a LinkedList and constant-time in an ArrayList.

    Advatnages/Disadvantages ArrayList

    The **ArrayList is actually encapsulating an actualy Array**, an Object[]. When we instantiate ArrayList, an array is created, and when we add values into it, the array changes its size accordingly.

- **Fast Random Access**
- **Adding values might be slow** When we don't know the amount of values the array will contain when we create it, a lot of shifting is going to be done in the memory space when the ArrayList manipulates its internal array.
- **Slow manipulation** When you'll want to add a value randomly inside the array, between two already existing values, the array will have to start moving all the values one spot to the right in order to let that happen.

    Advatnages/Disadvantages LinkedList

    The LinkedList is implemented using nodes linked to each other. Each node contains a *previous node* link, *next node* link, and *value*, which contains the actual data. When new data is inserted, a node is inserted and the links of the surrounding nodes are updated accordingly. When one is removed, the same happens – The surrounding nodes are changing their links and the deleted node is garbage collected.

- **Fast manipulation** adding and removing new data *anywhere* in the list is instantanious. Change two links, and you have a new value anywhere you want it.
- **No random access** Even though the get(int) is still there, it now just iterates the list until it reaches the index you specified. It has some optimizations in order to do that, but that's basically it.

18. ***What is hashMap and when will you use HashMap. How HashMap functions?***
19. ***Employee e1 = new Employee(1,"XXX","YYY");***
    Employee e2 = new Employee(1,"XXX","YYY");
    if we add both these objects to hash set, then what value will be printed for hashset.size()

Threads

20. What are threads? Given a scenario where we need to monitor 5 servers for space usage, how will u implement using threads

Simply put, a *thread* is a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time. Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time. The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time. Threads allows us to do this.

21. How do you instantiate thread

There are two ways to create thread in java;

- Implement the Runnable interface (java.lang.Runnable)
- By Extending the Thread class (java.lang.Thread)

Implementing the Runnable Interface

The Runnable Interface **Signature**

```
public interface Runnable {
void run();
}
```

One way to create a thread in java is to implement the Runnable Interface and then instantiate an object of the class. We need to override the run() method into our class which is the only method that needs to be implemented. The run() method contains the logic of the thread.

Order of steps:

1. A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.
2. An object of Thread class is created by passing a Runnable object as argument to the Thread constructor. The Thread object now has a Runnable object that implements the run() method.
3. The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.
4. The thread ends when the run() method ends, either by normal completion or by throwing an uncaught exception.

Extending Thread Class

The procedure for creating threads based on extending the Thread is as follows:

1. A class extending the Thread class overrides the run() method from the Thread class to define the code executed by the thread.
2. This subclass may call a Thread constructor explicitly in its constructors to initialize the thread, using the super() call.
3. The start() method inherited from the Thread class is invoked on the object of the class to make the thread eligible for running.

22. Difference between t.start() and t.run() method

start() enable the Thread to be controlled in states. Explicity invoking run() , violates the flow and state of a Thread execution.
start() is not invoked for the main method thread or "system" or group threads created/set up by the VM. Any new functionality added to this method in the future may have to also be added to the VM.

When you call Start , run gets invoked in a new thread (multi threading) , call run directly and it will get invoked on your calling thread (single threading) you probably never want to do that and it confuses the reader (if you did want that effect but the implementation in a new method and have the run call it, you can then call your new method if you want it run single threaded).

The start() method starts the Thread and executes the run() method's code in the new Thread. Calling the run() method does not start a new Thread, it executes the run() method's code in the current (calling) Thread. So the difference is: Using the start() method you have a new OS level Thread and the Thread's code is executed concurrently to the current Thread's code, but using the run() method, there is no new OS level Thread so it's code gets executed immediately and in sequence with the current Thread.

However, when we call the start method it push the thread in to Ready state & after that thread is on mercy of thread scheduler. when thread get chance to run in running state then JVM call the Run method.

23. What is common compile time exception that you see in threads

24. What is synchronization?

Synchronization is best use with the Multi-Threading in Java, Synchronization is the capability to control the access of multiple threads to share resources. Without synchronization it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value.This often leads to an error.

The Java programming language provides two basic synchronization idioms: synchronized methods and synchronized statements.
To make a method synchronized, simply add the synchronized keyword to its declaration:

In Summary Java Synchronized Keyword provides following functionality essential for concurrent programming :

1) synchronized keyword in java provides locking which ensures mutual exclusive access of shared resource and prevent data race.
2) synchronized keyword also prevent reordering of code statement by compiler which can cause subtle concurrent issue if we don't use synchronized or volatile keyword.
3) synchronized keyword involve locking and unlocking. before entering into synchronized method or block thread needs to acquire the lock at this point it reads data from main memory than cache and when it release the lock it flushes write operation into main memory which eliminates memory inconsistency errors.

25. What does Class.ForName() do in JDBC and Core Java

A call to *Class.forName("X")* causes the class named *X* to be dynamically loaded (at runtime). A call to forName("X") causes the class named X to be initialized (i.e., JVM executes all its static block after class loading). *Class.forName("X")* returns the *Class* object associated with the "X" class. The returned *Class* object is not an instance of the "x" class itself. The JVM keeps track of all the classes that have been previously loaded. This method uses the *classloader* of the class that invokes it. The "X" is the fully qualified name of the desired class.

26. In an ArrayList, employee details like emp id, salary, name have been added. How do you sort by salary? Write code for it. If salary is same for all employees how do you sort other than salary

27. ***What is an Object in java***

A Java object is a set of data combined with methods for manipulating that data. An object is made from a class; a class is the blueprint for the object. Each object made from a class will have its own values for the instance variables of that class. Instance variables are things an object knows about itself.

28. Tell me the methods of java.lang.Object class

- **clone**() Creates a new object of the same class as this object.
- **equals**(Object) Compares two Objects for equality.
- **finalize**() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- **getClass**() Returns the runtime class of an object.
- **hashCode**() Returns a hash code value for the object.
- **notify**() Wakes up a single thread that is waiting on this object's monitor.
- **notifyAll**() Wakes up all threads that are waiting on this object's monitor.
- **toString**() Returns a string representation of the object.
- **wait**() Waits to be notified by another thread of a change in this object.
- **wait**(long) Waits to be notified by another thread of a change in this object.
- **wait**(long, int)

29. Explain about equals() and hashcode() methods

Java.lang.Object has methods called hasCode() and equals(). These methods play a significant role in the real time application. However its use is not always common to all applications. In some case these methods are overridden to perform certain purpose.

hashCode()

As you know this method provides the has code of an object. Basically the default implementation of hashCode() provided by Object is derived by mapping the memory address to an integer value. If look into the source of Object class , you will find the following code for the hashCode.

public native int hashCode();

It indicates that hashCode is the native implementation which provides the memory address to a certain extent. However it is possible to override the hashCode method in your implementation class.

equals()

This particular method is used to make equal comparison between two objects. There are two types of comparisons in Java. One is using “= =” operator and another is “equals()”.More specifically the “.equals()”

refers to equivalence relations. So in broad sense we say that two objects are equivalent they satisfy the "equals()" condition. The source code of Object class you will find the following code for the equals() method.

```
public boolean equals(Object obj)
{
return (this == obj);
}
```

when to override the equals() and hashCode() methods and why it is necessary to override these methods. In this regard there is a rule of thumb that if you are going to override the one of the methods(ie equals() and hashCode()) , we have to override the both otherwise it is a violation of contract made for equals() and hashCode().

30. ***Employee e1= new Employee(1,'xxxx',5000);***
    Employee e2 = new Employee(1,'xxxx',5000);
    If(e1.equals(e2)) -----→ true/false ?
    True.

31. ***What is cloning?***
    protected Object clone() throws CloneNotSupportedException - this method is used to create a copy of an object of a class which implements Cloneable interface. By default it does field-by-field copy as the Object class doesn't have any idea in advance about the members of the particular class whose objects call this method. So, if the class has only primitive data type members then a completely new copy of the object will be created and the reference to the new object copy will be returned. But, if the class contains members of any class type then only the object references to those members are copied and hence the member references in both the original object as well as the cloned object refer to the same object.

    We get CloneNotSupportedException if we try to call the clone() method on an object of a class which doesn't implement the Cloneable interface.

32. ***Why cloning?***
    One reason for creating a local copy of an object is because you plan to modify the object, and you don't want to modify the method caller's object. If we need a local copy, you can perform the operation by using the clone() method of the Object class. The clone() method is defined as protected, but we must redefine it as public in all subclasses that you might want to clone.

33. ***Difference between abstract class and interface, when to go abstract class***
34. ***What are the methods present in object class***
35. ***Questions on equals() and what happens internally when you call equals()***

36. ***In heap memory some of the objects are duplicates, how can you avoid duplicate from heap***
37. ***What is an immutable object in Java?***

Immutable objects are simply objects whose state (the object's data) cannot change after construction. Examples of immutable objects from the JDK include String and Integer.

Immutable objects greatly simplify your program, since they :

- are simple to construct, test, and use
- are automatically thread-safe and have no synchronization issues
- do not need a copy constructor
- do not need an implementation of clone
- allow hashCode to use lazy initialization, and to cache its return value
- do not need to be copied defensively when used as a field
- make good Map keys and Set elements (these objects must not change state while in the collection)