Q1. What are 4 OOPS principal in java?

- **Encapsulation -** encapsulation means data hiding,
- **Abstraction -** Abstraction means hiding the implementation,
- **Inheritance -** Inheritance is a process where child class acquires the properties of super class, and
- **Polymorphism -**
  - ***Compile time polymorphism*** *- can be achieved by using Method overloading &*
  - ***Runtime polymorphism*** *- Runtime polymorphism can be achieved by using Method overriding*

For detail Please read : 4 oops (object oriented programming) concepts in java > Encapsulation, Abstraction, Inheritance, Polymorphism in detail with programs, abstraction interface via Abstract classes, Inheritance via extends and implements,Compile Runtime polymorphism

Q2. Is there any difference between creating string with and without new operator?

When String is created **without new** operator, it will be created in **string pool**.

When String is created using **new** operator, it will force JVM to create new string in heap (not in string pool).

Let's discuss step-by-step what will happen when below 5 statements will be executed >

```
String s1 = "abc";
 String s2 = new String("abc");
 String s3 = "abc";
 String s4 = new String("abc");
 String s5 = new String("abc").intern();
```

`String s1 = "abc";`

No string with "abc" is there in pool, so JVM will create string in **string pool** and s1 will be a reference variable which will refer to it.

`String s2 = new String("abc");`

string is created using **new** operator, it will force JVM to create new string in heap (not in string pool).

```
String s3 = "abc";
```

string with “abc” is there in pool, so s3 will be a reference variable which will refer to “abc” in **string pool.**

```
String s4 = new String("abc");
```

string is created using **new** operator, it will force JVM to create new string in heap (not in string pool).

```
String s5 = new String("abc").intern();
```

string is created using new operator but intern method has been invoked on it, so s5 will be a reference variable which will refer to “abc” in **string pool.**

Q3. What is significance of final in java?

1. **Final memberVariable/instanceVariable** of class must be initialized at time of declaration, once initialized final memberVariable cannot be assigned a new value.
2. **Final method** cannot be overridden, any attempt to do so will cause compilation error.

```
5  class Superclass {
6      final void method(){} //final method
7  }
8
9  class Subclass extends Superclass{
10     void method(){} //Final method cannot be overridden
11 }
```

Runtime polymorphism is not applicable on final methods because they cannot be overridden.

3. **Final class** cannot be extended, any attempt to do so will cause compilation error.

```
5  final class Superclass {  //final method
6  }
7
8  class Subclass extends Superclass{ //final class cannot be extended
9  }
```

For more Please read : [final keyword in java - 20 salient features]

Q4. Can we use custom object as key in HashMap? If yes then how?

For using object as Key in HashMap, we must implements [equals and hashcode method].
Classes must be made [immutable classes].

Q5. How do we override equals and hashcode method, write a code to use Employee(i.e. custom object) as key in HashMap? (Important)

After answering above question, immediately it will be followed up by this questions
We will override equals() and hashCode() like this -

By overriding equals() and hashCode() method we could use custom object as key in HashMap.

1) Check whether obj is null or not.

```
   if(obj==null) //If obj is null, return without comparing obj & Employee class.
```

2) check whether obj is instance of Employee class or not.

```
 if(this.getClass()!=obj.getClass()) //identifies whether obj is instance of Employee class or not.
```

3) Then, type cast obj into employee instance.

```
  Employee emp=(Employee)obj;  //type cast obj into employee instance.
```

```
    @Override
    public boolean equals(Object obj){

           if(obj==null)
                  return false;

           if(this.getClass()!=obj.getClass())
                  return false;

           Employee emp=(Employee)obj;
           return (emp.id==this.id || emp.id.equals(this.id))
                               && (emp.name==this.name || emp.name.equals(this.name));
    }

    @Override
    public int hashCode(){
           int hash=(this.id==null ? 0: this.id.hashCode() ) +
                        (this.name==null ? 0: this.name.hashCode() );
           return hash;
    }
```

Let's say in an organisation there exists a employee with **id=1 and name='sam'** and **some data** is stored corresponding to him, but if modifications have to be made in data, **previous data must be overridden**.

Must read : Overriding equals and hashcode method - Top 18 Interview questions

Q6. What classes should i prefer to use a key in HashMap? (Important)

This question will check your in depth knowledge of Java's Collection Api's. we should prefer **String, Integer, Long, Double, Float, Short and any other wrapper class.** Reason behind using them as a key is that they override equals() and hashCode() method, we need not to write any explicit code for overriding equals() and hashCode() method.

Let's use Integer class as key in HashMap.

```
import java.util.HashMap;
import java.util.Map;

public class StringInMap {
    public static void main(String...a){

          //HashMap's key=Integer class  (Integer's api has already overridden
hashCode() and equals() method for us )
           Map<Integer, String> hm=new HashMap<Integer, String>();
           hm.put(1, "data");
           hm.put(1, "data OVERRIDDEN");

           System.out.println(hm.get(1));

    }
}
/*OUTPUT

data OVERRIDDEN

*/
```

If, we note above program, what we will see is we didn't override equals() and hashCode() method, but still we were able to store data in HashMap, override data and retrieve data using get method.

>Let's check in **Integer's API**, how Integer class has overridden equals() and hashCode() method :

```
      public int hashCode() {
        return value;
}

      public boolean equals(Object obj) {
        if (obj instanceof Integer) {
         return value == ((Integer)obj).intValue();
        }
        return false;
}
```

Q7. Can overriding of hashcode() method cause any performance issues?

Improper implementation of hashCode() can cause performance issues, because in that most of the key-value pairs will be stored on same bucket location and unnecessary time will be consumed while fetching value corresponding to key.

Q8. What are immutable classes in java? How we can create immutable classes in java? And what are advantages of using immutable classes?

Any change made to object of immutable class produces new object.
Example- String is Immutable class in java, any changes made to Sting class.

We must follow following steps for creating immutable classes -

1) **Final class** - Make class final so that it cannot be inherited

2) **private member variable** -> Making member variables private ensures that fields cannot be accessed outside class.

3) **final member variable** -> Make member variables final so that once assigned their values cannot be changed

4) **Constructor** -> Initialize all fields in constructor.
assign all mutable member variable using new keyword.

5) **Don't provide setter methods** in class/ provide only getter methods.

Please read : Program for Creating Immutable class in java

Q9. What are some immutable classes in java?

String is Immutable class in java, any changes made to Sting object produces new String object.

Example of more immutable classes in java (Wrapper class)**>**

- Integer
- Double
- Long
- Short
- Byte
- And all other Wrapper classes.

Important questions on Differences in Collection>

Q10. What are differences between List and Set?

Please read : List and Set - Similarity and Differences

Q11. What are differences between HashMap and Hashtable?

Please read : HashMap and Hashtable - Similarity and Differences

Q12. What are differences between ArrayList and Vector?

Please read : ArrayList and Vector - Similarity and Differences

Q13. What are differences between ArrayList and LinkedList?

Please read : ArrayList and LinkedList - Similarity and Differences

Q14. What are differences between HashSet vs LinkedHashSet vs TreeSet?

Please read : HashSet vs LinkedHashSet vs TreeSet - Similarity and Differences

Q15. What are differences between HashMap and ConcurrentHashMap?

Please read : HashMap and ConcurrentHashMap - Similarity and Differences

Q16. What are differences between HashMap vs Hashtable vs LinkedHashMap vs TreeMap?

Please read : HashMap vs Hashtable vs LinkedHashMap vs TreeMap - Differences

Q17. What are differences between HashMap vs IdentityHashMap?

Please read : HashMap and IdentityHashMap - Similarity and Differences with program

Q18. What are differences between Collection and Collections?

Please read : Collection and Collections - Differences

Q19. What are differences between Comparable and Comparator?

Please read : Comparable and Comparator - differences

Q20. What are differences between Iterator and ListIterator?

Please read : Iterator and ListIterator - Similarity and Differences

Q21. What are differences between Iterator and ListIterator?

Please read : ArrayList and CopyOnWriteArrayList - Similarity and Differences with program

Important questions on Similarity and Differences Collection classes in concurrent and non-concurrent packages >

Q22. What are differences between HashSet and CopyOnWriteArraySet?

Please read : HashSet and CopyOnWriteArraySet - Similarity and Differences with program

Q23. What are differences between TreeSet and ConcurrentSkipListSet?

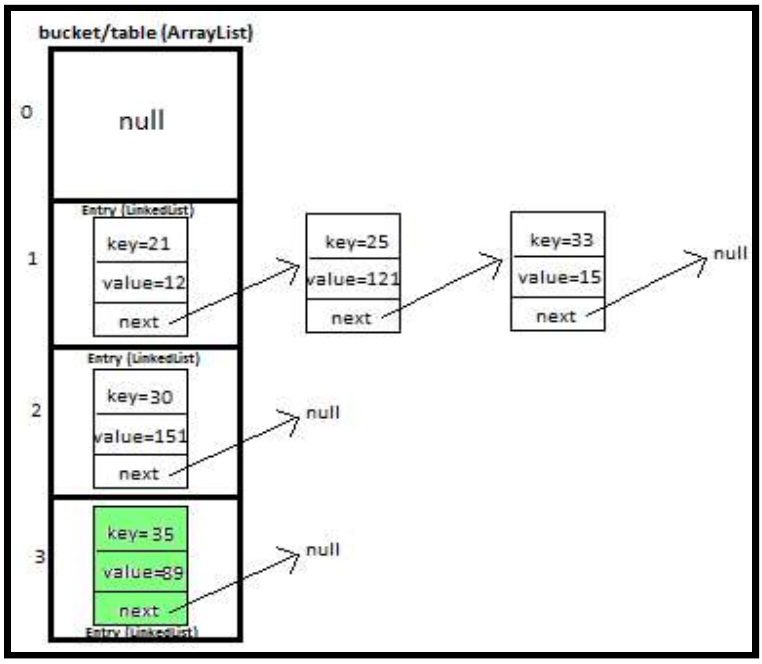
Please read : TreeSet and ConcurrentSkipListSet - Similarity and Differences with program

Q24. What are differences between TreeMap and ConcurrentSkipListMap?

Please read : TreeMap and ConcurrentSkipListMap - Similarity and Differences with program

Q25. How to implement own HashMap in java?

Answer.



Must Read : HashMap Custom implementation

Q26. What do you mean by fail-fast and fast-safe? What is ConcurrentModificationException?

Answer.

Iterator returned by few Collection framework Classes are **fail-fast,** means any structural modification made to these classes during iteration will throw ConcurrentModificationException.

Some important classes whose returned iterator is **fail-fast >**

- ArrayList
- LinkedList
- vector
- HashSet

Iterator returned by few Collection framework Classes are **fail-safe,** means any structural modification made to these classes during iteration won't throw any Exception.

Some important classes whose returned iterator is **fail-safe >**

- CopyOnWriteArrayList
- CopyOnWriteArraySet
- ConcurrentSkipListSet

For more detail read : ConcurrentModificationException, Fail-fast and Fail-safe in detail in java

Few important Exception questions>

Q27. What are differences between Iterator and Enumeration?

Please read : Iterator and Enumeration - Differences and similarities

Q28. Does finally block always execute? Will finally block execute when System.exit is called?

finally is not executed when System.exit is called, finally block is also not executed when **JVM crashes** because of some java.util.**Error**.

Q29. What are differences between checked and unchecked exceptions?

Please read : Checked (compile time exceptions) and UnChecked (RuntimeExceptions) in java - Definition and differences

Q30. What are exception handling keywords in java?

5 keyword in java exception handling

- **try** **- Any exception occurring in try block is catched by catch block.**
- **catch** **-** catch block is always followed by try block.
- **finally** ***finally block*** *can can only exist if try or try-catch block is there, finally block can't be used alone in java.*

  Features *of finally >*

  - finally block is **always executed** irrespective of exception is thrown or not.
  - finally block is optional in java, we may use it or not.

  finally block is ***not executed*** *in following scenarios >*

  - finally is not executed when **System.exit** is called.
  - if in case **JVM crashes** because of some java.util.**Error**.
- **throw** **throw** is a **keyword** in java.
  - **throw** keyword allows us to throw **checked** or **unchecked** exception.
- **throws** **throws** is written in method's definition to indicate that method can throw exception.

Q31. What are differences between Exception *and* Error *in java?*

Please read : *Differences between* ***Exception and Error in java***, **Exception** ***and*** **Error**

Q32. What is difference between throw and throws in java?

Please read : *Differences between* ***throw and throws*** *in java*, **throw** and **throws**

Q33. What is Difference between multiple catch block and multi catch syntax ?

Please read : **Difference between multiple catch block and multi catch syntax**

Q34. What is Difference between Final, Finally and Finalize?

Please read : **Final, Finally and Finalize - difference and similarity in java**

Q35. What is exception propagation in java?

Whenever methods are called stack is formed and an exception is first thrown from the top of the stack and if it is not caught, it starts coming down the stack to previous methods until it is not caught.
If exception remains uncaught even after reaching bottom of the stack it is propagated to JVM and program is terminated.
For more read : *Exception **propagation** in java - **deep understanding** of how checked and unchecked exceptions are **propagated***

Q36. Mention few exception handling best practices ?

Please read : Exception handling best practices and guidelines for using exceptions in java

Q37. What is cloning in java?

Cloning is done for copying the object, cloning can be done using shallow or deep copy, we will discuss it later in post.

Few key points about clone method>

- ***1)*** **Definition** of clone method -

```
protected native Object clone() throws CloneNotSupportedException;
```

  - Clone is a **protected** method - clone method can't be called outside class without inheritance.
  - Clone is **native** method, if not overridden its implementation is provided by JVM.
  - It returns Object - Means explicitly cast is needed to convert it to original object.
- ***2)*** **By default clone method do shallow copy.**
- ***3)*** Class must implement marker interface java.lang.Cloneable. If class doesn't implement Cloneable than calling clone method on its object will throw **CloneNotSupportedException**.
- ***4)*** **shallow copy-** If we implement Cloneable interface, we must override clone method and call super.clone() from it, invoking super.clone() will do shallow copy.
- ***5)*** **Deep copy -** We need to provide custom implementation of clone method for deep copying. When the copied object contains some other object its references are copied recursively in deep copy.

Please read more about cloning on : Cloning in java using clone- Shallow and deep copy - 8 techniques for deep copying- 8 important points about cloning

Q38. Is overriding of static method allows in java?

No, *read :* *Why Static method cannot be overridden in java - Explanation with program*

Q39. Is overriding of private method allows in java?

No, private methods are inherited in subclass and hence cannot be overridden.
Though subclass can have same name of private method in superclass.

```
class A {
    private final void m(){
            System.out.println(1);
    }
}

class B extends A {
    void m(){
            System.out.println(2);
    }
}
```

Q40. What is difference between Interface and abstract class in java?

Please read : 10 Differences between Interface and abstract class in java - in detail with programs

Q41. What is difference between Method overloading and Method overriding in java?

Please read : 10 Difference between Method overloading and Method overriding in java - in detail with programs

Q42. What is difference between equals method and == operator in java? And what will be output of following code snippets?

Code snippet 1 >

```
String s1 = new String("abc");
String s2 = new String("abc");

System.out.println(s1.equals(s2));
```

Code snippet 2 >

```
StringBuffer sb1 = new StringBuffer("abc");
StringBuffer sb2 = new StringBuffer("abc");

System.out.println(sb1.equals(sb2));
```

== operator

The **== operator** checks for referential equality of object, it checks whether two references are referring to same object or not.

equals method

By default equals method checks for referential equality (Class may override the method and provide different functionality)
By default if x == y returns true then equals method also returns true.

Example -

> **>StringBuffer class doesn't overrides equals method** of Object class. **The result is true if both references are referring to same StringBuffer object.**
> **>String class overrides equals method** of Object class and compares one string object to the other string object. **The result is true if characters in both String object appears in same order.**

Output of Code snippet 1 > true
Output of Code snippet 2 > false

Please read : Difference between equals method and == operator in java - testing with String and StringBuffer

Q43. What is constructor chaining in java?

whenever the object of class is created, implicitly default no-arg constructor of class and its super class constructor is called.

Q. But how constructor of superclass is called?
A. Implicitly first statement of constructor is super(), [that means by default first statement of constructor super() is called, super() calls implicit/explicit no-arg constructor of superclass].

Let's we have superclass and subclass like this -

```
class SuperClass{
}

class SubClass extends SuperClass{
}
```

compiler will add default implicit no-arg constructor -

```
class SuperClass{
    SuperClass(){ //no-arg constructor
            super();
 }
}

class SubClass extends SuperClass{
    SubClass(){
            super();
    }
}
```

*For full program please :*Constructor in java - Constructor chaining, access modifiers with constructors, constructor overloading, exception thrown, constructors are not inherited

Q44. What are 4 java platforms ?

- Java SE/ J2SE - Standard Edition,
- Java EE/ J2EE - Enterprise Edition,
- Java ME/ J2ME - Micro Edition,
- JavaFX

Fore more read : Differences and relation between 4 Java Programming Language Platforms > Java SE/ J2SE - Standard Edition, Java EE/ J2EE - Enterprise Edition, Java ME/ J2ME - Micro Edition, JavaFX

Q45. What are Differences between JDK, JRE and JVM, OpenJDK, JIT Compiler?

1) JDK (Java Development Kit)

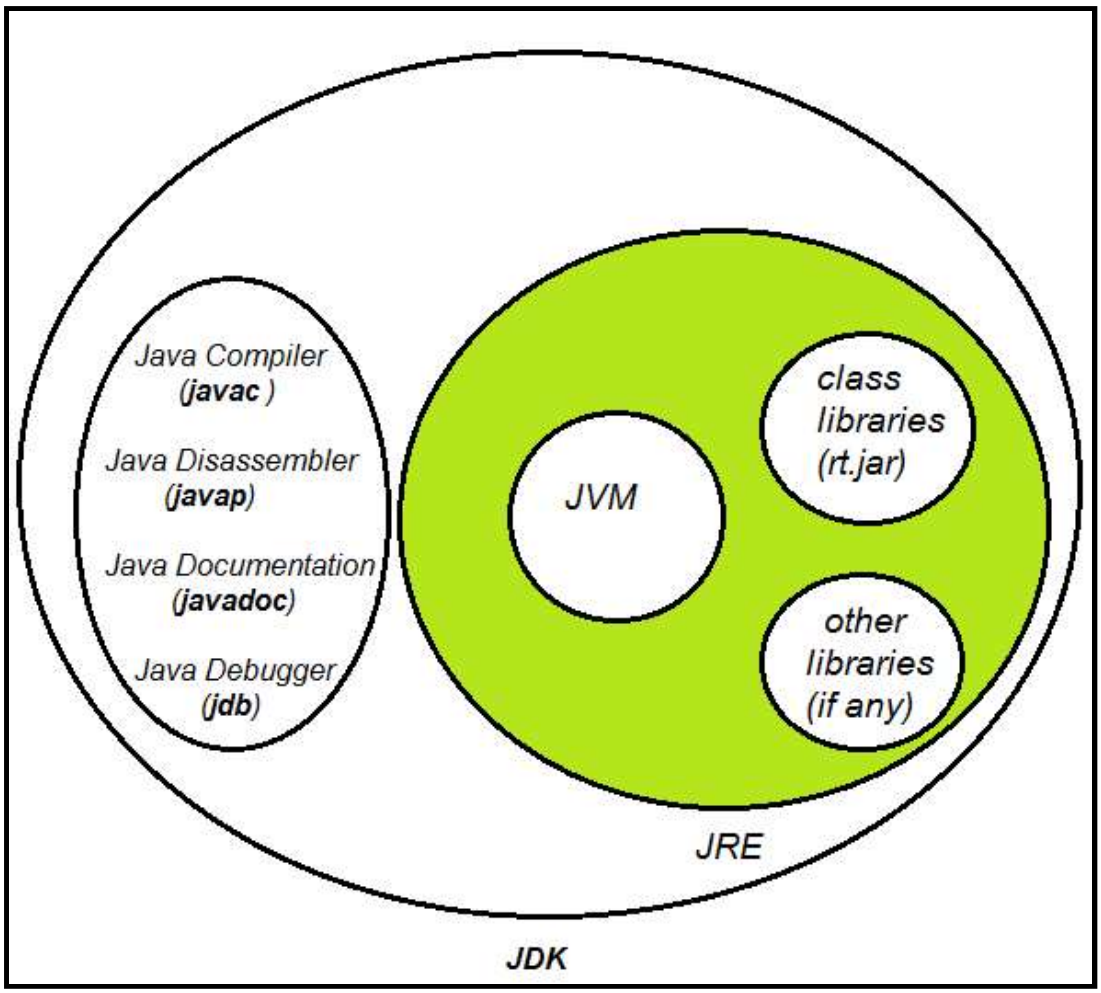
As the name suggests, JDK is required for java development.

You need to have JDK in your system for >

- ***developing, compiling and running Java programs.***

JDK = JRE + JVM.

JDK diagram >



2) JRE (Java Runtime environment)

JRE provides environment for running/executing programs.

You need to have JRE in your system for >

- ***running Java programs.***

JRE = JVM + class libraries (rt.jar) + other libraries (if any).

3) JVM (java virtual machine)

JVM is the virtual machine on which java code executes.

JVM is responsible for converting byte code into machine specific code.

Fore more read : JDK (Java Development Kit), JRE (Java Runtime environment), JVM (java virtual machine), Differences between JDK, JRE and JVM, OpenJDK, JIT Compiler (Just In Time Compiler)

internal working

Q46. How many primitive data types are provided by java ?

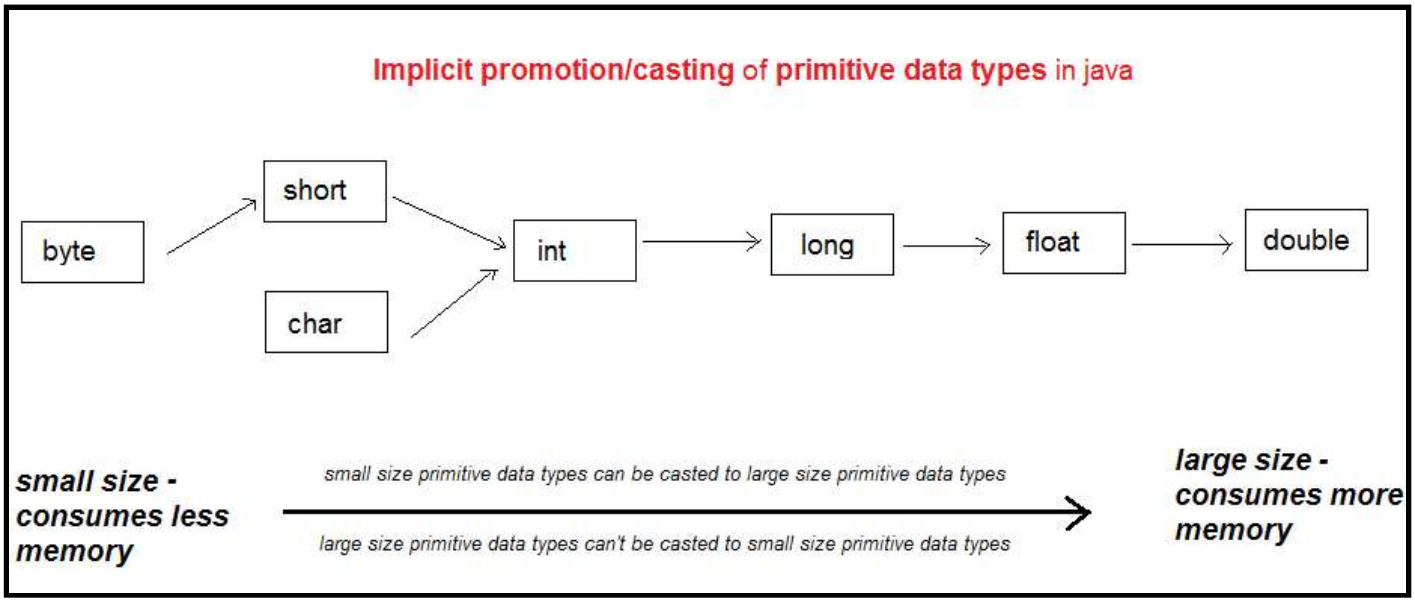
Java provides 8 primitive data types

byte
short
char
int
long
float
boolean

Fore more read : Primitive, Custom/reference Data Types, Integer, Floating-Point, Character and String literal, Escape sequence in java, decimal to hexaDecimal and binary conversion program

Q47. Explain Implicit casting/promotion of primitive Data type in java*?*

Diagram of **Implicit casting/promotion** of **primitive Data type** in java >



boolean cannot be casted implicitly or explicitly to any other datatype.

Q48. What is Thread in java?

- Threads **consumes CPU in best possible manner**, hence enables multi processing. Multi threading **reduces idle time of CPU** which improves performance of application.

For more Please read : What is thread in java

Q49. How to implement Threads in java?

This is very basic threading question. Threads can be created in two ways i.e. by implementing **java.lang.Runnable** interface or extending **java.lang.Thread** class and then extending run method.

1. Thread creation by implementing **java.lang.Runnable** interface.

We will create object of class which implements Runnable interface :

```
MyRunnable runnable=new MyRunnable();
Thread thread=new Thread(runnable);
```

2) And then create Thread object by calling constructor and passing reference of Runnable interface i.e. `runnable` object :

```
Thread thread=new Thread(runnable);
```

Q50. We should implement Runnable interface or extend Thread class. What are differences between implementing Runnable and extending Thread?

Well the answer is you must **extend Thread** only when you are looking to **modify run() and other methods as well**. If you are simply looking to **modify only the run() method implementing Runnable is the best option (**Runnable interface has only one abstract method i.e. run() **).**

1. ***Multiple inheritance in not allowed in java :*** When we implement Runnable interface **we can extend another class as well**, but if we extend Thread class **we cannot extend any other class** because java does not allow multiple inheritance**.** So, same work is done by implementing Runnable and extending Thread but in case of implementing Runnable we are still left with option of extending some other class**. So, it's better to implement Runnable.**

2. ***Thread safety :*** When we implement Runnable interface, **same object is shared amongst multiple threads**, but when we extend Thread class **each and every thread gets associated with new object.**

For more Please read : *Differences between implementing Runnable interface and extending Thread class*

Q51. How can you ensure all threads that started from main must end in order in which they started and also main should end in last? (Important)

Interviewers tend to know interviewees knowledge about Thread methods. So this is time to prove your point by answering correctly. We can use **join() method** to ensure all threads that started from main must end in order in which they started and also main should end in last.
DETAILED DESCRIPTION : Join() method - ensure all threads that started from main must end in order in which they started and also main should end in last. Types of join() method with programs- 10 salient features of join.

Q52. What is difference between starting thread with run() and start() method? (Important)

This is quite interesting question, it might confuse you a bit and at time may make you think is there really any difference between starting thread with run() and start() method.
When you **call start()** method, **main thread internally calls run() method** to start newly created thread. So **run() method is ultimately called by newly created thread.**
When you **call run()** method **main thread** rather than starting run() method with newly thread it start **run() method by itself.**

Q53. What is significance of using Volatile keyword?

Java allows threads to **access shared variables.** As a rule, to ensure that **shared variables are consistently updated**, a thread should ensure that it has **exclusive use of such variables by obtaining a lock** that enforces mutual exclusion for those shared variables.

If a field is declared volatile, in that case the Java memory model ensures that all threads see a consistent value for the variable.

DETAILED DESCRIPTION : Volatile keyword in java- difference between synchronized and volatile with programs, 10 key points about volatile keyword, why volatile variables are not cached in memory

Q54. Differences between synchronized and volatile keyword in Java? (Important)

Its very important question from interview perspective.

1. Volatile does not acquire any lock on variable or object, but Synchronization acquires lock on method or block in which it is used.

2. Volatile variables are not cached, but variables used inside synchronized method or block are cached.

3. When volatile is used will never create deadlock in program, as volatile never obtains any kind of lock . But in case if synchronization is not done properly, we might end up creating dedlock in program.

DETAILED DESCRIPTION : **Differences between synchronized and volatile keyword in detail with programs.**

Q55. Can you again start Thread?

No, we cannot start Thread again, doing so will throw runtimeException java.lang.IllegalThreadStateException. The reason is once run() method is executed by Thread, it goes into **dead state**.

Q56. What is race condition in multithreading and how can we solve it? (Important)

This is very important question, this forms the core of multi threading, you should be able to explain about race condition in detail. When more than one thread try to access same resource without synchronization causes race condition.

So we can solve race condition by using either synchronized block or synchronized method. When no two threads can access same resource at a time phenomenon is also called as **mutual exclusion.**

Q57. How threads communicate between each other?

This is very must know question for all the interviewees, you will most probably face this question in almost every time you go for interview.

Threads can communicate with each other by using **wait(), notify() and notifyAll()** methods.

Q58. Why wait(), notify() and notifyAll() are in Object class and not in Thread class? (Important)

Please read : Why wait(), notify() and notifyAll() are in Object class and not in Thread class

Q59. Is it important to acquire object lock before calling wait(), notify() and notifyAll()?

Yes, it's mandatory to acquire object lock before calling these methods on object. As discussed above **wait(), notify() and notifyAll() methods are always called from Synchronized block only**, and **as soon as thread enters synchronized block it acquires object lock** (by holding object monitor). If we call these methods without acquiring object lock i.e. from outside synchronize block then java.lang. IllegalMonitorStateException is thrown at runtime.

Wait() method needs to enclosed in try-catch block, because it throws compile time exception i.e. InterruptedException.

Q60. How can you solve consumer producer problem by using wait() and notify() method? (Important)

Here come the time to answer **very very important question from interview perspective**. Interviewers tends to check how sound you are in threads inter communication. Because for solving this problem we got to **use synchronization blocks, wait() and notify() method very cautiously**. **If you misplace synchronization**

block or any of the method, that **may cause your program to go horribly wrong.** So, before going into this question first i'll recommend you to understand how to use synchronized blocks, wait() and notify() methods.

DETAILED DESCRIPTION **with program : Solve Consumer Producer problem by using wait() and notify() methods in multithreading.**

Q61. How can you solve consumer producer pattern by using BlockingQueue? (Important)

Now it's time to gear up to face question which is most probably going to be followed up by previous question i.e. after how to solve consumer producer problem using wait() and notify() method. Generally you might wonder why interviewer's are so much interested in asking about solving consumer producer problem using BlockingQueue, answer is they want to know how strong knowledge you have about java concurrent Api's, this Api use consumer producer pattern in very optimized manner, BlockingQueue is designed is such a manner that it offer us the best performance.
BlockingQueue is a interface and we will use its **implementation class LinkedBlockingQueue.**

Key methods for solving consumer producer pattern are >

```
put(i);      //used by producer to put/produce in sharedQueue.
take(); //used by consumer to take/consume from sharedQueue.
```

Q62. What is difference between String, StringBuffer and StringBuilder in java ?

Difference between String, StringBuffer and StringBuilder in java - In depth coverage

Q63. What is reflection in java? Have you ever used reflection directly or directly?

Reflection is used to **load java classes at runtime**. This is very interesting question, though you may not have used reflection directly in java, but definitely you must have used it indirectly, but you must be thinking how. answer is quite simple - Frameworks like struts, spring and hibernate uses reflection for loading classes at runtime.

Q64. What is significance of static in java?

1. The **static** is a **keyword** in java.
2. Static variable, method, class are stored in **perm gen**(permanent generation memory).

   *Static **variable***
3. static variables are also known as **class variables**.

4. We need **not** to create instance of class for accessing static variables.

*Static **method***

5. static methods are also known as **class methods**.
6. We need not to **create instance of class for accessing static methods**.

Static class

7. static class are also known as **static nested class**.
8. Top level class can never be static in java.

*Static **block***

9. static blocks are also known as **static initialization blocks** in java.
10. They are called as soon as class is loaded even before instance of class is created (i.e. before constructor is called).

Please read more : **Static keyword in java - variable, method, class, block - 20 salient features**

*Q65. Can we override **static** method in java?*

Its one of the favourite question of interviewers. Intention is to test very basic core java concept.
Static method cannot be overridden, any attempt to do this will **not** cause compilation error, but the results won't be same when we would have overridden non-static methods.
But why?
Overriding in Java means that the method would be called on the run time based on type of the object and not on the compile time type of it .

But static methods are class methods access to them is always resolved during compile time only using the compile time type information.

For more please read : **Static method cannot be overridden**

Q66. How do you ensure different thread access different resources concurrently without deadlock?

Acquire lock on resources in a particular order and then releasing acquired lock in reverse order won't create any deadlock.

Q67. Which method is called for garbage collection in java? What algorithm does garbage collector follows?

JVM calls finalize method on object for garbage collection

*JVM follows **mark and sweep algorithm** for garbage collection.*

Mark and sweep algorithm *internal working in 3 steps >*

STEP 1 > *Unreferenced objects (garbage) are not reclaimed immediately.*

- *Instead, garbage(unreferenced objects) is gathered until memory is not full.*

STEP 2 >*When memory becomes full >*

- *execution of the program is suspended temporarily,*
- *mark and sweep algorithm collects all the garbage (i.e. all unreferenced objects are reclaimed)*

STEP 3 >*Once garbage is collected >*

- *execution of the program is resumed.*

Q68. What is Singleton class/design pattern in java?

Singleton class means only one instance of class can exist.

For more please read : Singleton in java, Doubly Locked Singleton class/ Lazy initialization, enum singleton, eager initialization in static block

Q69. objects and primitive types are passed by value or reference?

Primitive are passed by value & objects are passed by reference ***in java.***

Q70. What are different type of classes in java?

- ***1) Inner class/ nested class***
- ***2) static nested class***
- ***3) Local inner class***
- ***4) Anonymous inner class***

For detailed explanation must read : Inner class/ nested class, static nested class, local and anonymous inner class in java

Q71. What is Differences between Instance initialization block and Static initialization block?

Please read : Differences between Instance initialization block and Static initialization block in java - Features in detail with programs

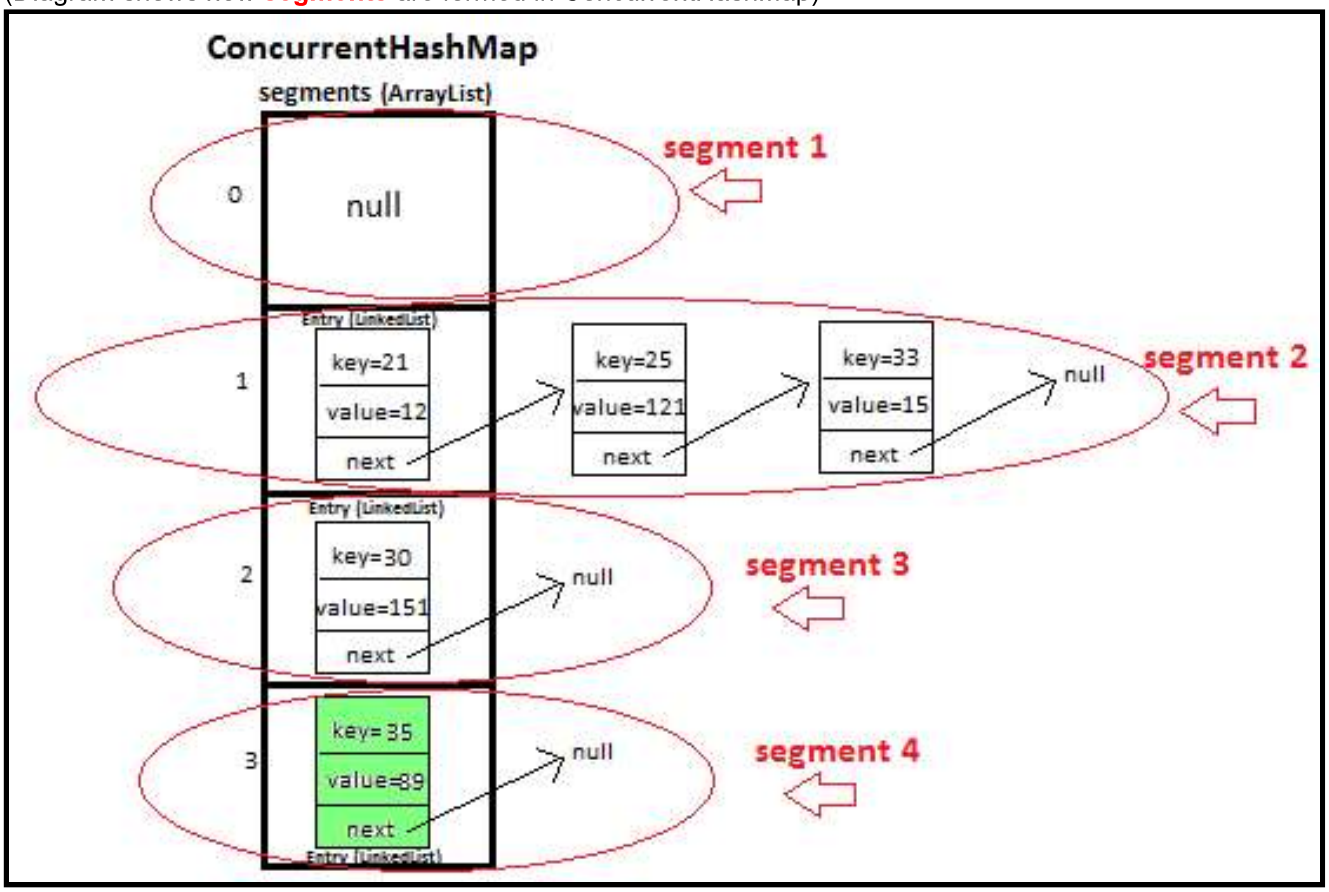
Q72. How ConcurrentHashMap works? Can 2 threads on same ConcurrentHashMap object access it concurrently?

ConcurrentHashMap is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently.

Can threads read the segment locked by some other thread?
Yes. When thread locks one segment for updation it does not block it for retrieval (done by get method) hence some other thread can read the segment (by get method), but it will be able to read the data before locking.

Segments in ConcurrentHashMap with diagram >

we have ConcurrentHashMap with **4 segments -**
(Diagram shows how **segments** are formed in ConcurrentHashMap)



Q73. What is deadlock in multithreading? Write a program to form DeadLock in multi threading and also how to solve DeadLock situation. What measures you should take to avoid deadlock? (Important)

This is very important question from interview perspective. But, what makes this question important is it checks interviewees capability of **creating and detecting deadlock**. If you can write a code to form deadlock, than I am sure you must be well capable in solving that deadlock as well. If not, later on this post we will learn how to solve deadlock as well.
First question comes to mind is, **what is deadlock in multi threading program?**
Deadlock is a situation where two threads are waiting for each other to release lock holded by them on resources.

Q74. What is life cycle of Thread, explain thread states? (Important)

Thread states/ Thread life cycle is very basic question, before going deep into concepts we must understand Thread life cycle.

Thread have following states >

- **New**
- **Runnable**
- **Running**
- **Waiting/blocked/sleeping**
- **Terminated (Dead)**

Thread life cycle in diagram >

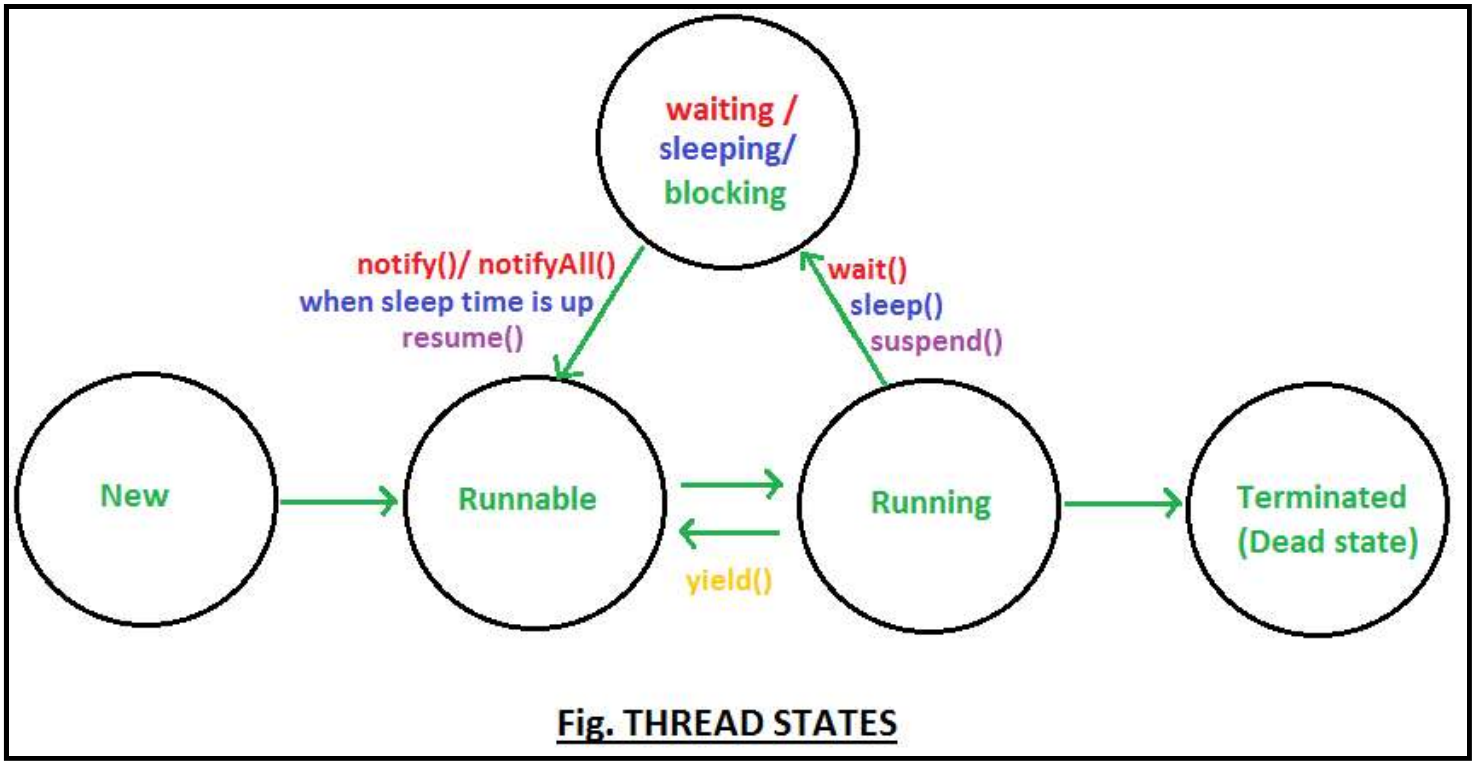


Fig. THREAD STATES

You may like to have in depth knowledge of [Thread states/ Thread life cycle in java & explanation of thread methods which method puts thread from which state to which state.]

Q75. What are daemon threads?

Daemon threads are low priority threads which **runs intermittently in background** for doing **garbage collection.**

Q76. Difference between wait() and sleep() ? (Important)

Please read :

Difference between wait() and sleep() method in threads
wait(),

Sleep() method in threads - 10 key features with programs

Wait() and notify() methods- Definition, 8 key features

Q77. Differences and similarities between yield() and sleep() ?

Please read :

Differences and similarities between yield() and sleep() in threads

Yield() method in threads - 8 key features with programs

Q78. Mention some guidelines to write thread safe code, most important point we must take care of in multithreading programs?

In multithreading environment it's important very important to ***write thread safe code***, thread unsafe code can cause a major threat to your application. I have posted many articles regarding thread safety. So overall this will be revision of what we have learned so far i.e. writing thread safe healthy code and avoiding any kind of deadlocks.

Q79.Difference between notify() and notifyAll() methods, can you write a code to prove your point?

Goodness. Theoretically you must have heard or you must be aware of differences between notify() and notifyAll().But have you created program to achieve it? If not let's do it.

Please read : Difference between notify() and notifyAll() methods, with program

Q80. What will happen if we don't override run method?

This question will test your basic knowledge how start and run methods work internally in Thread Api.

When we call start() method on thread, **it internally calls run() method** with newly created thread. **So, if we don't override run() method newly created thread won't be called and nothing will happen.**

For more Please read : What will happen if we don't override run method of thread?

Q81. What will happen if we override start method?

This question will again test your basic core java knowledge how overriding works at runtime, what what will be called at runtime and how start and run methods work internally in Thread Api.

When we call start() method on thread, **it internally calls run()** method with newly created thread. **So, if we override start() method, run() method will not be called** until we write code for calling run() method.

For more Please read : What will happen if we override start method of thread?

Q82. Can we acquire lock on class? What are ways in which you can acquire lock on class?

Yes, we can acquire lock on class's class object in 2 ways to acquire lock on class.
Thread can acquire lock on class's class object by-

1. Entering **synchronized block**

```
synchronized (MyClass.class) {
   //thread has acquired lock on MyClass's class object.
}
```

Or,

2. by entering **static synchronized methods.**

```
public static synchronized void method1() {
   //thread has acquired lock on MyRunnable's class object.
}
```

Q83. Difference between object lock and class lock?

Please read : difference between object lock and class lock to answer interview, ocjp answers correctly.

Q84. How can you implement your own Thread Pool in java?

ThreadPool is a pool of threads which **reuses a fixed number of threads** to execute tasks.
At any point, **at most nThreads threads will be active processing tasks**. **If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available.**
ThreadPool implementation internally uses LinkedBlockingQueue for adding and removing tasks.
In this post i will be using LinkedBlockingQueue provided by java Api, you can refer this post for implementing ThreadPool using custom LinkedBlockingQueue.

For more please read : Implement Thread pool in java.

Q85. What is significance of using ThreadLocal?

This question will test your command in multi threading, can you really create some perfect multithreading application or not. ThreadLocal is a class which provides thread-local variables.
For more please read : *threadLocal in java.*

Q86. What is busy spin?

When one thread loops continuously waiting for another thread to signal.

Performance point of view - Busy spin is **very bad** from performance point of view, because one thread keeps on looping continuously (and consumes CPU) waiting for another thread to signal.

Solution to busy spin -
We must use sleep() or wait() and notify() method. Using wait() is better option.

Program - Consumer Producer problem with busy spin >
Consumer thread continuously execute (**busy spin**) in while loop till `productionInProcess` is true. Once producer thread has ended it will make boolean variable `productionInProcess` false and **busy spin** will be over.

```
while(productionInProcess){
   System.out.println("BUSY SPIN - Consumer waiting for production to get
over");
}
```

Please read : *See here for Busy spin in detail.*

Q87. What is executor framework?

Executor and ExecutorService are used for following purposes >

- creating thread,
- starting threads,
- managing whole life cycle of Threads.

Executor creates pool of threads and manages life cycle of all threads in it.

In Executor framework, **Executor** interface and **ExecutorService** class are most prominently used.

Executor interface defines very important execute() method which executes command.

ExecutorService interface extends **Executor** interface.

An Executor interface provides following type of methods >

- methods for managing termination and
- methods that can produce a Future for tracking progress of tasks.

For more Please read : **Executor and ExecutorService framework in java**.

Q88. What are differences between execute() and submit() method of executor framework?

For more Please read : **Differences between execute() and submit() method of executor framework**

Q89. What is Semaphore in java 7?

A **semaphore** controls access to a shared resource by using permits.

- **If permits are greater than zero**, then semaphore **allow access to shared resource**.
- **If permits are zero or less than zero**, then semaphore **does not allow access to shared resource**.

These permits are sort of counters, which allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore.

For more Please read : Semaphore in java.

Q90. How can you implement Producer Consumer pattern using Semaphore?

For more Please read : Semaphore used for implementing Producer Consumer pattern.

Q91. What is significance of atomic classes in java 7?

Java provides some classes in **java.util.concurrent.atomic** which offers an alternative to the other synchronization.

*Classes found in **java.util.concurrent.atomic** are >*

- **AtomicInteger**,
- **AtomicLong**, and
- **AtomicBoolean**.

Q92. What are Future and Callable? How are they related?

Future<V> interface provides method for >

- for **returning result** of computation, wait until computation is not completed

Callable<V> interface provides method for computing a result and returning that computed result or throws an exception if unable to do so

Any class implementing Callable interface must override ***call()*** method for computing a result.

How Callable and Future are related?

If you submit a Callable object to an Executor returned object is of Future type.

```
Future<Double> futureDouble=executor.submit(new SquareDoubleCallable(2.2));
```

where, `SquareDoubleCallable` is a class which implements Callable.

For more Please read : Executor and ExecutorService framework in java.

Q93. What is CountDownLatch?

There might be situation where we might like our thread to wait until one or more threads completes certain operation.

A CountDownLatch is initialized with a given ***count*** .

count specifies the number of events that must occur before latch is released.

Every time a event happens ***count*** is reduced by 1. Once count reaches 0 latch is released.

Read more about : CountDownLatch in java.

Q94. Where can you use CountDownLatch in real world?

When you go in amusement park, you must have seen on certain rides there is mandate that at least 3 people (**3 is count**) should be there to take a ride. So, ride keeper (**ride keeper is main thread**) waits for 3 persons (**ride keeper has called await()**).
Every time a person comes count is reduced by 1 (**let's say every person is calling countDown() method**). Ultimately when 3 persons reach count becomes 0 & wait for ride keeper comes to end.

Read more about : CountDownLatch in java.

Q95.What is CyclicBarrier?

There might be situation where we might have to trigger event only when one or more threads completes certain operation.

2 or more threads wait for each other to reach a common barrier point. When all **threads** have **reached** common **barrier point** (i.e. when all threads have called await() method) >

- **All waiting threads are released**, and
- **Event can be triggered** as well.

For more Please read : CyclicBarrier in java.

Q96. Why is CyclicBarrier cyclic?

The barrier is called *cyclic* because CyclicBarrier can be reused after -

- All the waiting threads are released and
- event has been triggered.

Q97. Where could we use CyclicBarrier in real world?

Let's say 10 friends (**friends are threads**) have planned for picnic on place A (Here **place A is common barrier** point). And they all decided to play certain game (**game is event**) only on everyone's arrival at place A. So, all 10 friends must wait for each other to reach place A before launching event.

Now, when all **threads** have **reached** common **barrier point** (i.e. all friends have reached place A) >

- **All waiting threads are released** (All friends can play game), and
- **Event can be triggered** (they will start playing game).

Q98. Similarity and Difference between CyclicBarrier and CountDownLatch in Java?

CyclicBarrier can be **awaited repeatedly**, but **CountDownLatch** can't be awaited repeatedly. i.e. once count has become 0 cyclicBarrier can be used again but CountDownLatch cannot be used again.

For more Please read :Similarity and Difference between CyclicBarrier and CountDownLatch in Java

Q99. What is Phaser in java? Is Phaser similar to CyclicBarrier?

Phaser is somewhat **similar** in functionality of CyclicBarrier and CountDownLatch but it provides more flexibility than both of them.

Phaser provides us flexibility of registering and deRegistering parties at any time.

For registering parties, we may use any of the following -

- constructors, or
- int register(), or
- bulkRegister().

For deRegistering parties, we may use any of the following -

- arriveAndDeregister()

For more Please read : Phaser in java

Q100. Differences and similarity between Phaser and CyclicBarrier?

Like a **CyclicBarrier**, a Phaser can be **awaited repeatedly**.

But, in CyclicBarrier we used to register parties in constructor but Phaser provides us flexibility of registering and deRegistering parties at any time.

Q101. What is Lock in java?

The java.util.concurrent.locks.**Locks** is a interface and its implementations provide more extensive locking operations than can be obtained using synchronized methods and statements.

A lock helps in controlling access to a shared resource by multiple threads. Only one thread at a time can acquire the lock and access the shared resource.

If a second thread attempts to acquire the lock on shared resource when it is acquired by another thread, the second thread will wait until the lock is released. In this way we can achieve synchronization and race conditions can be avoided.

Read lock of a ReadWriteLock may allow concurrent access to a shared resource.

For more Please read : locks and ReEntrantLocks in java

Q102. Difference between synchronized and ReentrantLock?

Please read : Difference between synchronized and ReentrantLock

Question 103. What is Fork/Join Framework ?

*Fork/Join framework enables **parallel programming***. Parallel programming means taking **advantage two or more processors (multicore) in computers**. Parallel programming improves program performance.

Read more about Fork/Join Framework - Parallel programming in java.

Question 104. What is Serialization in java?

Let's start by understanding what is Serialization, it's most basic question which **you will have to answer almost in each and every java interview**. Serialization is process of converting **object into byte stream**.

Serialized object (byte stream) can be:

>Transferred over network.

>Persisted/saved into file.

>Persisted/saved into database.

Once, object have have been transferred over network or persisted in file or in database, we could deserialize the object and retain its state as it is in which it was serialized.

Q105 . Difference between Externalizable and Serialization interface ?

Difference between Externalizable and Serialization interface in java

Q106. How can you avoid certain member variables of class from getting Serialized?

Mark member variables as **static** or **transient**, and those member variables will no more be a part of Serialization.

Q107. What is serialVersionUID? What will be impact of not defining serialVersionUID in class?

This is one my favourite question, The serialization at runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

If we don't define serialVersionUID in the class, and any **modification** is made in class, then we **won't be able to deSerialize our class** because **serialVersionUID generated by java compiler for modified class will be different from old serialized object**. And deserialization process will end up throwing **java.io.**`InvalidClassException` (because of serialVersionUID mismatch)

What is serialVersionUID? Impact of not defining serialVersionUID in class and avoiding `InvalidClassException`

Q108. What are compatible and incompatible changes in Serialization process?

compatible and incompatible changes in Serialization and deSerialization process in java

Q109. What will happen if one the member of class does not implement Serializable interface (Important)?

This is classy question which will check your in depth knowledge of Serialization concepts. If any of the member does not implement Serializable than NotSerializableException is thrown. Now, let's see a program.

Q110. What will happen if we have used List, Set and Map as member of class?

This question which will check your in depth knowledge of Serialization and Java Api's. ArrayList, HashSet and HashMap implements Serializable interface, so if we will use them as member of class they will get Serialized and DeSerialized as well. Now, let's see a program.

Question 111. Is constructor of class called during DeSerialization process?

This question which will check your in depth knowledge of Serialization and constructor chaining concepts. It depends on whether our object has implemented Serializable or Externalizable.
If **Serializable** has been implemented **-** constructor is **not called** during DeSerialization process.
But, if **Externalizable** has been implemented **-** constructor **is called** during DeSerialization process.

DETAILED DESCRIPTION : **Is constructor of class called during DeSerialization process**

Question 112. Is constructor of super class called during DeSerialization process of subclass?

Again your basic java concepts will be tested over here. It is depends on whether our superclass has implemented Serializable or not.
If superclass **has implemented Serializable -** constructor **is not called** during DeSerialization process.
If superclass has **not implemented Serializable -** constructor **is called** during DeSerialization process.
DETAILED DESCRIPTION : **Is constructor of super class called during DeSerialization process of sub class**

Q113. How you can avoid Deserialization process creating another instance of Singleton class?

This is another classy and very important question which will check your in depth knowledge of Serialization and Singleton concepts. I'll prefer you must understand this concept in detail. We can simply use **readResove()** method to return same instance of class, rather than creating a new one.

```
private Object readResolve() throws ObjectStreamException {
     return INSTANCE;
}
```

DETAILED DESCRIPTION : Avoid Deserialization process creating another instance of Singleton class

Q114. How can subclass avoid Serialization if its superClass has implemented Serialization interface ?

If superClass has implemented Serializable that means subclass is also Serializable (**as subclass always inherits all features from its parent class**), for avoiding Serialization in sub-class we can **define writeObject()** method and **throw NotSerializableException()**.

```
private void writeObject(ObjectOutputStream os) throws NotSerializableException {
        throw new NotSerializableException("This class cannot be Serialized");
}
```

DETAILED DESCRIPTION : Can subclass avoid Serialization if its superClass has implemented Serialization interface

Q115. Are primitive types part of serialization process?

Yes, primitive types are part of serialization process. Interviewer tends to check your basic java concepts over here.

Q116. Name few methods of Object class?

wait(), notify() and notifyAll()

equals and hashcode

clone()

toString()