## Core Java - OOPs Concepts: Package Interview Questions

### 124) What is the package?

A package is a group of similar type of classes, interfaces, and sub-packages. It provides access protection and removes naming collision. The packages in Java can be categorized into two forms, inbuilt package, and user-defined package. There are many built-in packages such as Java, lang, awt, javax, swing, net, io, util, sql, etc. Consider the following example to create a package in Java.

//save as Simple.java

**package** mypack;

**public** **class** Simple{

**public** **static** **void** main(String args[]){

    System.out.println("Welcome to package");

   }

}

  
[More details.](https://www.javatpoint.com/package)

### 125) What are the advantages of defining packages in Java?

By defining packages, we can avoid the name conflicts between the same class names defined in different packages. Packages also enable the developer to organize the similar classes more effectively. For example, one can clearly understand that the classes present in java.io package are used to perform io related operations.

### 127) How can we access some class in another class in Java?

There are two ways to access a class in another class.

* **By using the fully qualified name:** To access a class in a different package, either we must use the fully qualified name of that class, or we must import the package containing that class.
* **By using the relative path**, We can use the path of the class that is related to the package that contains our class. It can be the same or subpackage.

### 128) Do I need to import java.lang package any time? Why?

No. It is by default loaded internally by the JVM.

### 129) Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or the same class multiple times. Neither compiler nor JVM complains about it. However, the JVM will internally load the class only once no matter how many times you import the same class.

### 130) What is the static import?

By static import, we can access the static members of a class directly, and there is no to qualify it with the class name.

[More details.](https://www.javatpoint.com/static-import-in-java)

## Core Java - OOPs Concepts: RegEx Interview Questions

### 164) Name some classes present in ****java.util.regex**** package.

There are the following classes and interfaces present in java.util.regex package.

* MatchResult Interface
* Matcher class
* Pattern class
* PatternSyntaxException class



165) How the metacharacters are different from the ordinary characters?

Metacharacters have the special meaning to the regular expression engine. The metacharacters are ^, $, ., \*, +, etc. The regular expression engine does not consider them as the regular characters. To enable the regular expression engine treating the metacharacters as ordinary characters, we need to escape the metacharacters with the backslash.

### 166) Write a regular expression to validate a password. A password must start with an alphabet and followed by alphanumeric characters; Its length must be in between 8 to 20.

The regular expression for the above criteria will be: **^[a-zA-Z][a-zA-Z0-9]{8,19}** where ^ represents the start of the regex, [a-zA-Z] represents that the first character must be an alphabet, [a-zA-Z0-9] represents the alphanumeric character, {8,19} represents that the length of the password must be in between 8 and 20.

### 167) What is the output of the following Java program?

**import** java.util.regex.\*;

**class** RegexExample2{

**public** **static** **void** main(String args[]){

System.out.println(Pattern.matches(".s", "as")); //line 4

System.out.println(Pattern.matches(".s", "mk")); //line 5

System.out.println(Pattern.matches(".s", "mst")); //line 6

System.out.println(Pattern.matches(".s", "amms")); //line 7

System.out.println(Pattern.matches("..s", "mas")); //line 8

}}

**Output**

true

false

false

false

true

**Explanation**

line 4 prints true since the second character of string is s, line 5 prints false since the second character is not s, line 6 prints false since there are more than 3 characters in the string, line 7 prints false since there are more than 2 characters in the string, and it contains more than 2 characters as well, line 8 prints true since the third character of the string is s.

## Core Java: Nested classes and Interfaces Interview Questions

### 168) What are the advantages of Java inner classes?

There are two types of advantages of Java inner classes.

* Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of the outer class including private.
* Nested classes are used to develop a more readable and maintainable code because it logically groups classes and interfaces in one place only.
* **Code Optimization:** It requires less code to write.

### 169) What is a nested class?

The nested class can be defined as the class which is defined inside another class or interface. We use the nested class to logically group classes and interfaces in one place so that it can be more readable and maintainable. A nested class can access all the data members of the outer class including private data members and methods. The syntax of the nested class is defined below.

**class** Java\_Outer\_class{

 //code

**class** Java\_Nested\_class{

  //code

 }

}

There are two types of nested classes, static nested class, and non-static nested class. The non-static nested class can also be called as inner-class

[More details.](https://www.javatpoint.com/difference-between-nested-classes-and-inner-classes)

### 170) What are the disadvantages of using inner classes?

There are the following main disadvantages of using inner classes.

* Inner classes increase the total number of classes used by the developer and therefore increases the workload of JVM since it has to perform some routine operations for those extra classes which result in slower performance.
* IDEs provide less support to the inner classes as compare to the top level classes and therefore it annoys the developers while working with inner classes.

### 171) What are the types of inner classes (non-static nested class) used in Java?

There are mainly three types of inner classes used in Java.

|  |  |
| --- | --- |
| **Type** | **Description** |
| [Member Inner Class](https://www.javatpoint.com/member-inner-class) | A class created within class and outside method. |
| [Anonymous Inner Class](https://www.javatpoint.com/anonymous-inner-class) | A class created for implementing an interface or extending class. Its name is decided by the java compiler. |
| [Local Inner Class](https://www.javatpoint.com/local-inner-class) | A class created within the method. |

### 173) Can we access the non-final local variable, inside the local inner class?

No, the local variable must be constant if you want to access it in the local inner class.

[More details.](https://www.javatpoint.com/local-inner-class)

### 174) How many class files are created on compiling the OuterClass in the following program?

**public** **class** Person {

String name, age, address;

**class** Employee{

**float** salary=10000;

}

**class** BusinessMen{

**final** String gstin="£4433drt3$";

}

**public** **static** **void** main (String args[])

{

  Person p = **new** Person();

}

}

3 class-files will be created named as Person.class, Person$BusinessMen.class, and Person$Employee.class.

### 175) What are anonymous inner classes?

Anonymous inner classes are the classes that are automatically declared and instantiated within an expression. We cannot apply different access modifiers to them. Anonymous class cannot be static, and cannot define any static fields, method, or class. In other words, we can say that it a class without the name and can have only one object that is created by its definition. Consider the following example.

**abstract** **class** Person{

**abstract** **void** eat();

}

**class** TestAnonymousInner{

**public** **static** **void** main(String args[]){

  Person p=**new** Person(){

**void** eat(){System.out.println("nice fruits");}

  };

  p.eat();

 }

}

Output:

nice fruits

Consider the following example for the working of the anonymous class using interface.

**interface** Eatable{

**void** eat();

}

**class** TestAnnonymousInner1{

**public** **static** **void** main(String args[]){

 Eatable e=**new** Eatable(){

**public** **void** eat(){System.out.println("nice fruits");}

 };

 e.eat();

 }

}

Output:

nice fruits

### 176) What is the nested interface?

An Interface that is declared inside the interface or class is known as the nested interface. It is static by default. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The external interface or class must refer to the nested interface. It can't be accessed directly. The nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class. The syntax of the nested interface is given as follows.

**interface** interface\_name{

 ...

**interface** nested\_interface\_name{

  ...

 }

}

[More details.](https://www.javatpoint.com/nested-interface)

### 177) Can a class have an interface?

Yes, an interface can be defined within the class. It is called a nested interface.

[More details.](https://www.javatpoint.com/nested-interface)

### 178) Can an Interface have a class?

Yes, they are static implicitly.

[More details.](https://www.javatpoint.com/nested-interface)

## Garbage Collection Interview Questions

### 179) What is Garbage Collection?

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Due to garbage collection java gives 0 as output to a variable whose value is not set, i.e., the variable has been defined but not initialized. For this purpose, we were using free() function in the C language and delete() in C++. In Java, it is performed automatically. So, java provides better memory management.

[More details.](https://www.javatpoint.com/Garbage-Collection)

### 180) What is gc()?

The gc() method is used to invoke the garbage collector for cleanup processing. This method is found in System and Runtime classes. This function explicitly makes the Java Virtual Machine free up the space occupied by the unused objects so that it can be utilized or reused. Consider the following example for the better understanding of how the gc() method invoke the garbage collector.

1. **public** **class** TestGarbage1{
2. **public** **void** finalize(){System.out.println("object is garbage collected");}
3. **public** **static** **void** main(String args[]){
4. TestGarbage1 s1=**new** TestGarbage1();
5. TestGarbage1 s2=**new** TestGarbage1();
6. s1=**null**;
7. s2=**null**;
8. System.gc();
9. }
10. }

object is garbage collected

object is garbage collected

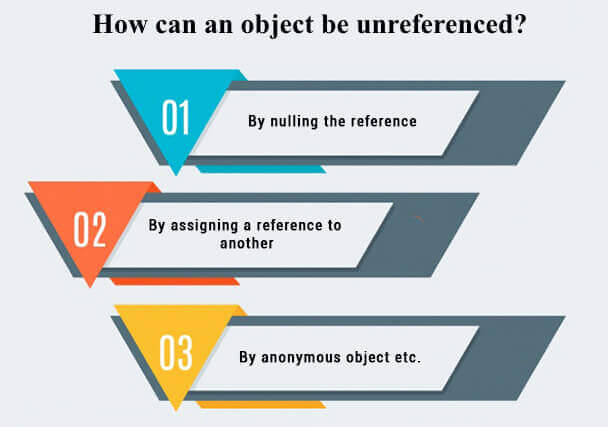
### 181) How is garbage collection controlled?

Garbage collection is managed by JVM. It is performed when there is not enough space in the memory and memory is running low. We can externally call the System.gc() for the garbage collection. However, it depends upon the JVM whether to perform it or not.

### 182) How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.



### 1) By nulling a reference:

1. Employee e=**new** Employee();
2. e=**null**;

### 2) By assigning a reference to another:

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

### 3) By anonymous object:

1. **new** Employee();

### 183) What is the purpose of the finalize() method?

The finalize() method is invoked just before the object is garbage collected. It is used to perform cleanup processing. The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created an object without new, you can use the finalize method to perform cleanup processing (destroying remaining objects). The cleanup processing is the process to free up all the resources, network which was previously used and no longer needed. It is essential to remember that it is not a reserved keyword, finalize method is present in the object class hence it is available in every class as object class is the superclass of every class in java. Here, we must note that neither finalization nor garbage collection is guaranteed. Consider the following example.

**public** **class** FinalizeTest {

**int** j=12;

**void** add()

    {

        j=j+12;

        System.out.println("J="+j);

    }

**public** **void** finalize()

    {

        System.out.println("Object is garbage collected");

    }

**public** **static** **void** main(String[] args) {

**new** FinalizeTest().add();

        System.gc();

**new** FinalizeTest().add();

    }

}



### 184) Can an unreferenced object be referenced again?

Yes,

### 185) What kind of thread is the Garbage collector thread?

Daemon thread.

### 186) What is the difference between final, finally and finalize?

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **final** | **finally** | **finalize** |
| 1) | Final is used to apply restrictions on class, method, and variable. The final class can't be inherited, final method can't be overridden, and final variable value can't be changed. | Finally is used to place important code, it will be executed whether an exception is handled or not. | Finalize is used to perform clean up processing just before an object is garbage collected. |
| 2) | Final is a keyword. | Finally is a block. | Finalize is a method. |

### 187) What is the purpose of the Runtime class?

Java Runtime class is used to interact with a java runtime environment. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory, etc. There is only one instance of java.lang.Runtime class is available for one java application. The Runtime.getRuntime() method returns the singleton instance of Runtime class.

### 188) How will you invoke any external process in Java?

By Runtime.getRuntime().exec(?) method. Consider the following example.

**public** **class** Runtime1{

**public** **static** **void** main(String args[])**throws** Exception{

  Runtime.getRuntime().exec("notepad");//will open a new notepad

 }

}

## I/O Interview Questions

### 189) Give the hierarchy of InputStream and OutputStream classes.

**OutputStream Hierarchy**

Java output stream hierarchy

**InputStream Hierarchy**

Java input stream hierarchy

### 190) What do you understand by an IO stream?

The stream is a sequence of data that flows from source to destination. It is composed of bytes. In Java, three streams are created for us automatically.

* System.out: standard output stream
* System.in: standard input stream
* System.err: standard error stream

### 191) What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented. The ByteStream classes are used to perform input-output of 8-bit bytes whereas the CharacterStream classes are used to perform the input/output for the 16-bit Unicode system. There are many classes in the ByteStream class hierarchy, but the most frequently used classes are FileInputStream and FileOutputStream. The most frequently used classes CharacterStream class hierarchy is FileReader and FileWriter.

### 192) What are the super most classes for all the streams?

All the stream classes can be divided into two types of classes that are ByteStream classes and CharacterStream Classes. The ByteStream classes are further divided into InputStream classes and OutputStream classes. CharacterStream classes are also divided into Reader classes and Writer classes. The SuperMost classes for all the InputStream classes is java.io.InputStream and for all the output stream classes is java.io.OutPutStream. Similarly, for all the reader classes, the super-most class is java.io.Reader, and for all the writer classes, it is java.io.Writer.

### 193) What are the FileInputStream and FileOutputStream?

**Java FileOutputStream** is an output stream used for writing data to a file. If you have some primitive values to write into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through the FileOutputStream class. However, for character-oriented data, it is preferred to use FileWriter than FileOutputStream. Consider the following example of writing a byte into a file.

**import** java.io.FileOutputStream;

**public** **class** FileOutputStreamExample {

**public** **static** **void** main(String args[]){

**try**{

             FileOutputStream fout=**new** FileOutputStream("D:\testout.txt");

             fout.write(65);

             fout.close();

             System.out.println("success...");

            }**catch**(Exception e){System.out.println(e);}

      }

}

**Java FileInputStream class** obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video, etc. You can also read character-stream data. However, for reading streams of characters, it is recommended to use FileReader class. Consider the following example for reading bytes from a file.

**import** java.io.FileInputStream;

**public** **class** DataStreamExample {

**public** **static** **void** main(String args[]){

**try**{

            FileInputStream fin=**new** FileInputStream("D:\testout.txt");

**int** i=fin.read();

            System.out.print((**char**)i);

            fin.close();

          }**catch**(Exception e){System.out.println(e);}

         }

        }

### 194) What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

Java BufferedOutputStream class is used for buffering an output stream. It internally uses a buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. Whereas, Java BufferedInputStream class is used to read information from the stream. It internally uses the buffer mechanism to make the performance fast.

### 195) How to set the Permissions to a file in Java?

In Java, FilePermission class is used to alter the permissions set on a file. Java FilePermission class contains the permission related to a directory or file. All the permissions are related to the path. The path can be of two types:

* D:\IO\-: It indicates that the permission is associated with all subdirectories and files recursively.
* D:\IO\\*: It indicates that the permission is associated with all directory and files within this directory excluding subdirectories.

Let's see the simple example in which permission of a directory path is granted with read permission and a file of this directory is granted for write permission.

**package** com.javatpoint;

**import** java.io.\*;

**import** java.security.PermissionCollection;

**public** **class** FilePermissionExample{

**public** **static** **void** main(String[] args) **throws** IOException {

      String srg = "D:\IO Package\java.txt";

      FilePermission file1 = **new** FilePermission("D:\IO Package\-", "read");

      PermissionCollection permission = file1.newPermissionCollection();

      permission.add(file1);

           FilePermission file2 = **new** FilePermission(srg, "write");

           permission.add(file2);

**if**(permission.implies(**new** FilePermission(srg, "read,write"))) {

           System.out.println("Read, Write permission is granted for the path "+srg );

             }**else** {

            System.out.println("No Read, Write permission is granted for the path "+srg);            }

     }

}

Output

Read, Write permission is granted for the path D:\IO Package\java.txt

### 196) What are FilterStreams?

**FilterStream classes** are used to add additional functionalities to the other stream classes. FilterStream classes act like an interface which read the data from a stream, filters it, and pass the filtered data to the caller. The FilterStream classes provide extra functionalities like adding line numbers to the destination file, etc.

### 197) What is an I/O filter?

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another. Many Filter classes that allow a user to make a chain using multiple input streams. It generates a combined effect on several filters.

### 198) In Java, How many ways you can take input from the console?

In Java, there are three ways by using which, we can take input from the console.

* **Using BufferedReader class:** we can take input from the console by wrapping System.in into an InputStreamReader and passing it into the BufferedReader. It provides an efficient reading as the input gets buffered. Consider the following example.

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**public** **class** Person

{

**public** **static** **void** main(String[] args) **throws** IOException

    {

      System.out.println("Enter the name of the person");

        BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(System.in));

        String name = reader.readLine();

        System.out.println(name);

    }

}

* **Using Scanner class:** The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values. Java Scanner class is widely used to parse text for string and primitive types using a regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces. Consider the following example.

**import** java.util.\*;

**public** **class** ScannerClassExample2 {

**public** **static** **void** main(String args[]){

          String str = "Hello/This is JavaTpoint/My name is Abhishek.";

          //Create scanner with the specified String Object

          Scanner scanner = **new** Scanner(str);

          System.out.println("Boolean Result: "+scanner.hasNextBoolean());

          //Change the delimiter of this scanner

          scanner.useDelimiter("/");

          //Printing the tokenized Strings

          System.out.println("---Tokenizes String---");

**while**(scanner.hasNext()){

            System.out.println(scanner.next());

        }

          //Display the new delimiter

          System.out.println("Delimiter used: " +scanner.delimiter());

          scanner.close();

          }

}



**Using Console class:** The Java Console class is used to get input from the console. It provides methods to read texts and passwords. If you read the password using the Console class, it will not be displayed to the user. The java.io.Console class is attached to the system console internally. The Console class is introduced since 1.5. Consider the following example.

**import** java.io.Console;

**class** ReadStringTest{

**public** **static** **void** main(String args[]){

Console c=System.console();

System.out.println("Enter your name: ");

String n=c.readLine();

System.out.println("Welcome "+n);

}

}

## Serialization Interview Questions

### 199) What is serialization?

Serialization in Java is a mechanism of writing the state of an object into a byte stream. It is used primarily in Hibernate, RMI, JPA, EJB and JMS technologies. It is mainly used to travel object's state on the network (which is known as marshaling). Serializable interface is used to perform serialization. It is helpful when you require to save the state of a program to storage such as the file. At a later point of time, the content of this file can be restored using deserialization. It is also required to implement RMI(Remote Method Invocation). With the help of RMI, it is possible to invoke the method of a Java object on one machine to another machine.



[More details.](https://www.javatpoint.com/serialization)

### 200) How can you make a class serializable in Java?

A class can become serializable by implementing the Serializable interface.

### 201) How can you avoid serialization in child class if the base class is implementing the Serializable interface?

It is very tricky to prevent serialization of child class if the base class is intended to implement the Serializable interface. However, we cannot do it directly, but the serialization can be avoided by implementing the writeObject() or readObject() methods in the subclass and throw NotSerializableException from these methods. Consider the following example.

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.NotSerializableException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

**class** Person **implements** Serializable

{

    String name = " ";

**public** Person(String name)

    {

**this**.name = name;

    }

}

**class** Employee **extends** Person

{

**float** salary;

**public** Employee(String name, **float** salary)

    {

**super**(name);

**this**.salary = salary;

    }

**private** **void** writeObject(ObjectOutputStream out) **throws** IOException

    {

**throw** **new** NotSerializableException();

    }

**private** **void** readObject(ObjectInputStream in) **throws** IOException

    {

**throw** **new** NotSerializableException();

    }

}

**public** **class** Test

{

**public** **static** **void** main(String[] args)

**throws** Exception

    {

        Employee emp = **new** Employee("Sharma", 10000);

        System.out.println("name = " + emp.name);

        System.out.println("salary = " + emp.salary);

        FileOutputStream fos = **new** FileOutputStream("abc.ser");

        ObjectOutputStream oos = **new** ObjectOutputStream(fos);

        oos.writeObject(emp);

        oos.close();

        fos.close();

        System.out.println("Object has been serialized");

        FileInputStream f = **new** FileInputStream("ab.txt");

        ObjectInputStream o = **new** ObjectInputStream(f);

        Employee emp1 = (Employee)o.readObject();

        o.close();

        f.close();

        System.out.println("Object has been deserialized");

        System.out.println("name = " + emp1.name);

        System.out.println("salary = " + emp1.salary);

    }

}

### 202) Can a Serialized object be transferred via network?

Yes, we can transfer a serialized object via network because the serialized object is stored in the memory in the form of bytes and can be transmitted over the network. We can also write the serialized object to the disk or the database.

### 203) What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

**import** java.io.\*;

**class** Depersist{

**public** **static** **void** main(String args[])**throws** Exception{

  ObjectInputStream in=**new** ObjectInputStream(**new** FileInputStream("f.txt"));

  Student s=(Student)in.readObject();

  System.out.println(s.id+" "+s.name);

  in.close();

 }

}

211 ravi

### 204) What is the transient keyword?

If you define any data member as transient, it will not be serialized. By determining transient keyword, the value of variable need not persist when it is restored. [More details.](https://www.javatpoint.com/serialization)

### 205) What is Externalizable?

The Externalizable interface is used to write the state of an object into a byte stream in a compressed format. It is not a marker interface.

### 206) What is the difference between Serializable and Externalizable interface?

|  |  |  |
| --- | --- | --- |
| **No.** | **Serializable** | **Externalizable** |
| 1) | The Serializable interface does not have any method, i.e., it is a marker interface. | The Externalizable interface contains is not a marker interface, It contains two methods, i.e., writeExternal() and readExternal(). |
| 2) | It is used to "mark" Java classes so that objects of these classes may get the certain capability. | The Externalizable interface provides control of the serialization logic to the programmer. |
| 3) | It is easy to implement but has the higher performance cost. | It is used to perform the serialization and often result in better performance. |
| 4) | No class constructor is called in serialization. | We must call a public default constructor while using this interface. |

.

## Networking Interview Questions

### 207) Give a brief description of Java socket programming?

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connectionless. Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket, and DatagramPacket classes are used for connectionless socket programming. The client in socket programming must know two information:

* IP address of the server
* port number

### 208) What is Socket?

A socket is simply an endpoint for communications between the machines. It provides the connection mechanism to connect the two computers using TCP. The Socket class can be used to create a socket.

### 209) What are the steps that are followed when two computers connect through TCP?

There are the following steps that are performed when two computers connect through TCP.

* The ServerSocket object is instantiated by the server which denotes the port number to which, the connection will be made.
* After instantiating the ServerSocket object, the server invokes accept() method of ServerSocket class which makes server wait until the client attempts to connect to the server on the given port.
* Meanwhile, the server is waiting, a socket is created by the client by instantiating Socket class. The socket class constructor accepts the server port number and server name.
* The Socket class constructor attempts to connect with the server on the specified name. If the connection is established, the client will have a socket object that can communicate with the server.
* The accept() method invoked by the server returns a reference to the new socket on the server that is connected with the server.

### 210) Write a program in Java to establish a connection between client and server?

Consider the following program where the connection between the client and server is established.

*File: MyServer.java*

**import** java.io.\*;

**import** java.net.\*;

**public** **class** MyServer {

**public** **static** **void** main(String[] args){

**try**{

ServerSocket ss=**new** ServerSocket(6666);

Socket s=ss.accept();//establishes connection

DataInputStream dis=**new** DataInputStream(s.getInputStream());

String  str=(String)dis.readUTF();

System.out.println("message= "+str);

ss.close();

}**catch**(Exception e){System.out.println(e);}

}

}

*File: MyClient.java*

**import** java.io.\*;

**import** java.net.\*;

**public** **class** MyClient {

**public** **static** **void** main(String[] args) {

**try**{

Socket s=**new** Socket("localhost",6666);

DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());

dout.writeUTF("Hello Server");

dout.flush();

dout.close();

s.close();

}**catch**(Exception e){System.out.println(e);}

}

}

### 211) How do I convert a numeric IP address like 192.18.97.39 into a hostname like java.sun.com?

By InetAddress.getByName("192.18.97.39").getHostName() where 192.18.97.39 is the IP address. Consider the following example.

**import** java.io.\*;

**import** java.net.\*;

**public** **class** InetDemo{

**public** **static** **void** main(String[] args){

**try**{

InetAddress ip=InetAddress.getByName("195.201.10.8");

System.out.println("Host Name: "+ip.getHostName());

}**catch**(Exception e){System.out.println(e);}

}

}

## Reflection Interview Questions

### 212) What is the reflection?

Reflection is the process of examining or modifying the runtime behavior of a class at runtime. The java.lang.Class class provides various methods that can be used to get metadata, examine and change the runtime behavior of a class. The java.lang and java.lang.reflect packages provide classes for java reflection. It is used in:

* IDE (Integrated Development Environment), e.g., Eclipse, MyEclipse, NetBeans.
* Debugger
* Test Tools, etc.

### 213) What is the purpose of using java.lang.Class class?

The java.lang.Class class performs mainly two tasks:

* Provides methods to get the metadata of a class at runtime.
* Provides methods to examine and change the runtime behavior of a class.

### 214) What are the ways to instantiate the Class class?

There are three ways to instantiate the Class class.

* **forName() method of Class class:** The forName() method is used to load the class dynamically. It returns the instance of Class class. It should be used if you know the fully qualified name of the class. This cannot be used for primitive types.
* **getClass() method of Object class:** It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.
* **the .class syntax:** If a type is available, but there is no instance then it is possible to obtain a Class by appending ".class" to the name of the type. It can be used for primitive data type also.

### 215) What is the output of the following Java program?

**class** Simple{

**public** Simple()

 {

   System.out.println("Constructor of Simple class is invoked");

 }

**void** message(){System.out.println("Hello Java");}

}

**class** Test1{

**public** **static** **void** main(String args[]){

**try**{

  Class c=Class.forName("Simple");

  Simple s=(Simple)c.newInstance();

  s.message();

  }**catch**(Exception e){System.out.println(e);}

 }

}

**Output**

Constructor of Simple class is invoked

Hello Java

**Explanation**

The newInstance() method of the Class class is used to invoke the constructor at runtime. In this program, the instance of the Simple class is created.

### 216) What is the purpose of using javap?

The javap command disassembles a class file. The javap command displays information about the fields, constructors and methods present in a class file.

**Syntax**

javap fully\_class\_name

### 217) Can you access the private method from outside the class?

Yes, by changing the runtime behavior of a class if the class is not secured.

[More details.](https://www.javatpoint.com/reflection6)

## Miscellaneous Interview Questions

### 218)What are wrapper classes?

Wrapper classes are classes that allow primitive types to be accessed as objects. In other words, we can say that wrapper classes are built-in java classes which allow the conversion of objects to primitives and primitives to objects. The process of converting primitives to objects is called autoboxing, and the process of converting objects to primitives is called unboxing. There are eight wrapper classes present in **java.lang** package is given below.

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| boolean | Boolean |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

### 219)What are autoboxing and unboxing? When does it occur?

The autoboxing is the process of converting primitive data type to the corresponding wrapper class object, eg., int to Integer. The unboxing is the process of converting wrapper class object to primitive data type. For eg., integer to int. Unboxing and autoboxing occur automatically in Java. However, we can externally convert one into another by using the methods like valueOf() or xxxValue().

It can occur whenever a wrapper class object is expected, and primitive data type is provided or vice versa.

* Adding primitive types into Collection like ArrayList in Java.
* Creating an instance of parameterized classes ,e.g., ThreadLocal which expect Type.
* Java automatically converts primitive to object whenever one is required and another is provided in the method calling.
* When a primitive type is assigned to an object type.

### 220) What is the output of the below Java program?

**public** **class** Test1

{

**public** **static** **void** main(String[] args) {

  Integer i = **new** Integer(201);

  Integer j = **new** Integer(201);

**if**(i == j)

  {

    System.out.println("hello");

  }

**else**

  {

    System.out.println("bye");

  }

  }

}

**Output**

bye

**Explanation**

The Integer class caches integer values from -127 to 127. Therefore, the Integer objects can only be created in the range -128 to 127. The operator **==** will not work for the value greater than 127; thus **bye** is printed.

### 221) What is object cloning?

The object cloning is a way to create an exact copy of an object. The clone() method of the Object class is used to clone an object. The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException. The clone() method is defined in the Object class. The syntax of the clone() method is as follows:

**protected Object clone() throws CloneNotSupportedException**

### 222) What are the advantages and disadvantages of object cloning?

**Advantage of Object Cloning**

* You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
* It is the easiest and most efficient way of copying objects, especially if we are applying it to an already developed or an old project. Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.
* Clone() is the fastest way to copy the array.

**Disadvantage of Object Cloning**

* To use the Object.clone() method, we have to change many syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone(), etc.
* We have to implement the Cloneable interface while it does not have any methods in it. We have to use it to tell the JVM that we can perform a clone() on our object.
* Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
* Object.clone() does not invoke any constructor, so we do not have any control over object construction.
* If you want to write a clone method in a child class, then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
* Object.clone() supports only shallow copying, but we will need to override it if we need deep cloning.

### 223) What is a native method?

A native method is a method that is implemented in a language other than Java. Natives methods are sometimes also referred to as foreign methods.

### 224) What is the purpose of the strictfp keyword?

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language has provided the strictfp keyword so that you get the same result on every platform. So, now you have better control over the floating-point arithmetic.

### 225) What is the purpose of the System class?

The purpose of the System class is to provide access to system resources such as standard input and output. It cannot be instantiated. Facilities provided by System class are given below.

* Standard input
* Error output streams
* Standard output
* utility method to copy the portion of an array
* utilities to load files and libraries

There are the three fields of Java System class, i.e., static printstream err, static inputstream in, and standard output stream.

### 226) What comes to mind when someone mentions a shallow copy in Java?

Object cloning.

### 227) What is a singleton class?

Singleton class is the class which can not be instantiated more than once. To make a class singleton, we either make its constructor private or use the static getInstance method. Consider the following example.

**class** Singleton{

**private** **static** Singleton single\_instance = **null**;

**int** i;

**private** Singleton ()

     {

         i=90;

     }

**public** **static** Singleton getInstance()

     {

**if**(single\_instance == **null**)

         {

             single\_instance = **new** Singleton();

         }

**return** single\_instance;

     }

}

**public** **class** Main

{

**public** **static** **void** main (String args[])

    {

        Singleton first = Singleton.getInstance();

        System.out.println("First instance integer value:"+first.i);

        first.i=first.i+90;

        Singleton second = Singleton.getInstance();

        System.out.println("Second instance integer value:"+second.i);

    }

}



### 228) Write a Java program that prints all the values given at command-line.

**Program**

**class** A{

**public** **static** **void** main(String args[]){

**for**(**int** i=0;i<args.length;i++)

System.out.println(args[i]);

}

}

1. compile by > javac A.java
2. run by > java A sonoo jaiswal 1 3 abc

**Output**

sonoo

jaiswal

1

3

abc

### 229) Which containers use a border layout as their default layout?

The Window, Frame and Dialog classes use a border layout as their default layout.

### 230) Which containers use a FlowLayout as their default layout?

The Panel and Applet classes use the FlowLayout as their default layout.

### 231) What are peerless components?

The lightweight component of Swing is called peerless components. Spring has its libraries, so it does not use resources from the Operating System, and hence it has lightweight components.

### 232) is there is any difference between a Scrollbar and a ScrollPane?

The Scrollbar is a Component whereas the ScrollPane is a Container. A ScrollPane handles its events and performs its scrolling.

### 233) What is a lightweight component?

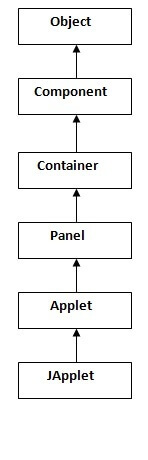
Lightweight components are the one which does not go with the native call to obtain the graphical units. They share their parent component graphical units to render them. For example, Swing components, and JavaFX Components.

### 234) What is a heavyweight component?

The portable elements provided by the operating system are called heavyweight components. AWT is limited to the graphical classes provided by the operating system and therefore, It implements only the minimal subset of screen elements supported by all platforms. The Operating system dependent UI discovery tools are called heavyweight components.

### 235) What is an applet?

An applet is a small java program that runs inside the browser and generates dynamic content. It is embedded in the webpage and runs on the client side. It is secured and takes less response time. It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os, etc. However, the plugins are required at the client browser to execute the applet. The following image shows the architecture of Applet.



When an applet is created, the following methods are invoked in order.

* init()
* start()
* paint()

When an applet is destroyed, the following functions are invoked in order.

* stop()
* destroy()

### 236) Can you write a Java class that could be used both as an applet as well as an application?

Yes. Add a main() method to the applet.

## Internationalization Interview Questions

### 237) What is Locale?

A Locale object represents a specific geographical, political, or cultural region. This object can be used to get the locale-specific information such as country name, language, variant, etc.

**import** java.util.\*;

**public** **class** LocaleExample {

**public** **static** **void** main(String[] args) {

Locale locale=Locale.getDefault();

//Locale locale=new Locale("fr","fr");//for the specific locale

System.out.println(locale.getDisplayCountry());

System.out.println(locale.getDisplayLanguage());

System.out.println(locale.getDisplayName());

System.out.println(locale.getISO3Country());

System.out.println(locale.getISO3Language());

System.out.println(locale.getLanguage());

System.out.println(locale.getCountry());

}

}

**Output:**

United States

English

English (United States)

USA

eng

en

US

### 238)How will you load a specific locale?

By ResourceBundle.getBundle(?) method.

## Java Bean Interview Questions

### 239) What is a JavaBean?

JavaBean is a reusable software component written in the Java programming language, designed to be manipulated visually by a software development environment, like JBuilder or VisualAge for Java. t. A JavaBean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides the easy maintenance. Consider the following example to create a JavaBean class.

//Employee.java

**package** mypack;

**public** **class** Employee **implements** java.io.Serializable{

**private** **int** id;

**private** String name;

**public** Employee(){}

**public** **void** setId(**int** id){**this**.id=id;}

**public** **int** getId(){**return** id;}

**public** **void** setName(String name){**this**.name=name;}

**public** String getName(){**return** name;}

}

### 240) What is the purpose of using the Java bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides the easy maintenance.

### 241) What do you understand by the bean persistent property?

The persistence property of Java bean comes into the act when the properties, fields, and state information are saved to or retrieve from the storage.

## RMI Interview Questions

### 242) What is RMI?

The RMI (Remote Method Invocation) is an API that provides a mechanism to create the distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.

### 243) What is the purpose of stub and skeleton?

**Stub**

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes the method on the stub object, it does the following tasks:

* It initiates a connection with remote Virtual Machine (JVM).
* It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM).
* It waits for the result.
* It reads (unmarshals) the return value or exception.
* It finally, returns the value to the caller.

**Skeleton**

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

* It reads the parameter for the remote method.
* It invokes the method on the actual remote object.
* It writes and transmits (marshals) the result to the caller.

### 244) What are the steps involved to write RMI based programs?

There are 6 steps which are performed to write RMI based programs.

* Create the remote interface.
* Provide the implementation of the remote interface.
* Compile the implementation class and create the stub and skeleton objects using the rmic tool.
* Start the registry service by the rmiregistry tool.
* Create and start the remote application.
* Create and start the client application.

### 245) What is the use of HTTP-tunneling in RMI?

HTTP tunneling can be defined as the method which doesn't need any setup to work within the firewall environment. It handles the HTTP connections through the proxy servers. However, it does not allow outbound TCP connections.

### 246) What is JRMP?

JRMP (Java Remote Method Protocol) can be defined as the Java-specific, stream-based protocol which looks up and refers to the remote objects. It requires both client and server to use Java objects. It is wire level protocol which runs under RMI and over TCP/IP.

### 247) Can RMI and CORBA based applications interact?

Yes, they can. RMI is available with IIOP as the transport protocol instead of JRMP.

## Concurrency Interview Questions

### 34) What are the main components of concurrency API?

Concurrency API can be developed using the class and interfaces of java.util.Concurrent package. There are the following classes and interfaces in java.util.Concurrent package.

* Executor
* FarkJoinPool
* ExecutorService
* ScheduledExecutorService
* Future
* TimeUnit(Enum)
* CountDownLatch
* CyclicBarrier
* Semaphore
* ThreadFactory
* BlockingQueue
* DelayQueue
* Locks
* Phaser

### 35) What is the Executor interface in Concurrency API in Java?

The Executor Interface provided by the package java.util.concurrent is the simple interface used to execute the new task. The execute() method of Executor interface is used to execute some given command. The syntax of the execute() method is given below.

**void execute(Runnable command)**

Consider the following example:

**import** java.util.concurrent.Executor;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.ThreadPoolExecutor;

**import** java.util.concurrent.TimeUnit;

**public** **class** TestThread {

**public** **static** **void** main(**final** String[] arguments) **throws** InterruptedException {

      Executor e = Executors.newCachedThreadPool();

      e.execute(**new** Thread());

      ThreadPoolExecutor pool = (ThreadPoolExecutor)e;

      pool.shutdown();

   }

**static** **class** Thread **implements** Runnable {

**public** **void** run() {

**try** {

            Long duration = (**long**) (Math.random() \* 5);

            System.out.println("Running Thread!");

            TimeUnit.SECONDS.sleep(duration);

            System.out.println("Thread Completed");

         } **catch** (InterruptedException ex) {

            ex.printStackTrace();

         }

      }

   }

}

**Output**

Running Thread!

Thread Completed

### 36) What is BlockingQueue?

The java.util.concurrent.BlockingQueue is the subinterface of Queue that supports the operations such as waiting for the space availability before inserting a new value or waiting for the queue to become non-empty before retrieving an element from it. Consider the following example.

**import** java.util.Random;

**import** java.util.concurrent.ArrayBlockingQueue;

**import** java.util.concurrent.BlockingQueue;

**public** **class** TestThread {

**public** **static** **void** main(**final** String[] arguments) **throws** InterruptedException {

      BlockingQueue<Integer> queue = **new** ArrayBlockingQueue<Integer>(10);

      Insert i = **new** Insert(queue);

      Retrieve r = **new** Retrieve(queue);

**new** Thread(i).start();

**new** Thread(r).start();

      Thread.sleep(2000);

   }

**static** **class** Insert **implements** Runnable {

**private** BlockingQueue<Integer> queue;

**public** Insert(BlockingQueue queue) {

**this**.queue = queue;

      }

      @Override

**public** **void** run() {

         Random random = **new** Random();

**try** {

**int** result = random.nextInt(200);

            Thread.sleep(1000);

            queue.put(result);

            System.out.println("Added: " + result);

            result = random.nextInt(10);

            Thread.sleep(1000);

            queue.put(result);

            System.out.println("Added: " + result);

            result = random.nextInt(50);

            Thread.sleep(1000);

            queue.put(result);

            System.out.println("Added: " + result);

         } **catch** (InterruptedException e) {

            e.printStackTrace();

         }

      }

   }

**static** **class** Retrieve **implements** Runnable {

**private** BlockingQueue<Integer> queue;

**public** Retrieve(BlockingQueue queue) {

**this**.queue = queue;

      }

      @Override

**public** **void** run() {

**try** {

            System.out.println("Removed: " + queue.take());

            System.out.println("Removed: " + queue.take());

            System.out.println("Removed: " + queue.take());

         } **catch** (InterruptedException e) {

            e.printStackTrace();

         }

      }

   }

}

**Output**

Added: 96

Removed: 96

Added: 8

Removed: 8

Added: 5

Removed: 5

### 37) How to implement producer-consumer problem by using BlockingQueue?

The producer-consumer problem can be solved by using BlockingQueue in the following way

**import** java.util.concurrent.BlockingQueue;

**import** java.util.concurrent.LinkedBlockingQueue;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**public** **class** ProducerConsumerProblem {

**public** **static** **void** main(String args[]){

     //Creating shared object

     BlockingQueue sharedQueue = **new** LinkedBlockingQueue();

     //Creating Producer and Consumer Thread

     Thread prod = **new** Thread(**new** Producer(sharedQueue));

     Thread cons = **new** Thread(**new** Consumer(sharedQueue));

     //Starting producer and Consumer thread

     prod.start();

     cons.start();

    }

}

//Producer Class in java

**class** Producer **implements** Runnable {

**private** **final** BlockingQueue sharedQueue;

**public** Producer(BlockingQueue sharedQueue) {

**this**.sharedQueue = sharedQueue;

    }

    @Override

**public** **void** run() {

**for**(**int** i=0; i<10; i++){

**try** {

                System.out.println("Produced: " + i);

                sharedQueue.put(i);

            } **catch** (InterruptedException ex) {

                Logger.getLogger(Producer.**class**.getName()).log(Level.SEVERE, **null**, ex);

            }

        }

    }

}

//Consumer Class in Java

**class** Consumer **implements** Runnable{

**private** **final** BlockingQueue sharedQueue;

**public** Consumer (BlockingQueue sharedQueue) {

**this**.sharedQueue = sharedQueue;

    }

    @Override

**public** **void** run() {

**while**(**true**){

**try** {

                System.out.println("Consumed: "+ sharedQueue.take());

            } **catch** (InterruptedException ex) {

                Logger.getLogger(Consumer.**class**.getName()).log(Level.SEVERE, **null**, ex);

            }

        }

    }

}

**Output**

Produced: 0

Produced: 1

Produced: 2

Produced: 3

Produced: 4

Produced: 5

Produced: 6

Produced: 7

Produced: 8

Produced: 9

Consumed: 0

Consumed: 1

Consumed: 2

Consumed: 3

Consumed: 4

Consumed: 5

Consumed: 6

Consumed: 7

Consumed: 8

Consumed: 9

### 38) What is the difference between Java Callable interface and Runnable interface?

The Callable interface and Runnable interface both are used by the classes which wanted to execute with multiple threads. However, there are two main differences between the both :

* A Callable <V> interface can return a result, whereas the Runnable interface cannot return any result.
* A Callable <V> interface can throw a checked exception, whereas the Runnable interface cannot throw checked exception.
* A Callable <V> interface cannot be used before the Java 5 whereas the Runnable interface can be used.

### 39) What is the Atomic action in Concurrency in Java?

* The Atomic action is the operation which can be performed in a single unit of a task without any interference of the other operations.
* The Atomic action cannot be stopped in between the task. Once started it fill stop after the completion of the task only.
* An increment operation such as a++ does not allow an atomic action.
* All reads and writes operation for the primitive variable (except long and double) are the atomic operation.
* All reads and writes operation for the volatile variable (including long and double) are the atomic operation.
* The Atomic methods are available in java.util.Concurrent package.

### 40) What is lock interface in Concurrency API in Java?

The java.util.concurrent.locks.Lock interface is used as the synchronization mechanism. It works similar to the synchronized block. There are a few differences between the lock and synchronized block that are given below.

* Lock interface provides the guarantee of sequence in which the waiting thread will be given the access, whereas the synchronized block doesn't guarantee it.
* Lock interface provides the option of timeout if the lock is not granted whereas the synchronized block doesn't provide that.
* The methods of Lock interface, i.e., Lock() and Unlock() can be called in different methods whereas single synchronized block must be fully contained in a single method.

### 41) Explain the ExecutorService Interface.

The ExecutorService Interface is the subinterface of Executor interface and adds the features to manage the lifecycle. Consider the following example.

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.TimeUnit;

**public** **class** TestThread {

**public** **static** **void** main(**final** String[] arguments) **throws** InterruptedException {

      ExecutorService e = Executors.newSingleThreadExecutor();

**try** {

         e.submit(**new** Thread());

         System.out.println("Shutdown executor");

         e.shutdown();

         e.awaitTermination(5, TimeUnit.SECONDS);

      } **catch** (InterruptedException ex) {

         System.err.println("tasks interrupted");

      } **finally** {

**if** (!e.isTerminated()) {

            System.err.println("cancel non-finished tasks");

         }

         e.shutdownNow();

         System.out.println("shutdown finished");

      }

   }

**static** **class** Task **implements** Runnable {

**public** **void** run() {

**try** {

            Long duration = (**long**) (Math.random() \* 20);

            System.out.println("Running Task!");

            TimeUnit.SECONDS.sleep(duration);

         } **catch** (InterruptedException ex) {

            ex.printStackTrace();

         }

      }

   }

}

**Output**

Shutdown executor

shutdown finished

### 42) What is the difference between Synchronous programming and Asynchronous programming regarding a thread?

**Synchronous programming:**In Synchronous programming model, a thread is assigned to complete a task and hence thread started working on it, and it is only available for other tasks once it will end the assigned task.

**Asynchronous Programming:**In Asynchronous programming, one job can be completed by multiple threads and hence it provides maximum usability of the various threads.

### 43) What do you understand by Callable and Future in Java?

**Java Callable interface:**In Java5 callable interface was provided by the package java.util.concurrent. It is similar to the Runnable interface but it can return a result, and it can throw an Exception. It also provides a run() method for execution of a thread. Java Callable can return any object as it uses Generic.

**Syntax:**

public interface Callable<V>

**Java Future interface:** Java Future interface gives the result of a concurrent process. The Callable interface returns the object of java.util.concurrent.Future.

Java Future provides following methods for implementation.

* **cancel(boolean mayInterruptIfRunning):** It is used to cancel the execution of the assigned task.
* **get():** It waits for the time if execution not completed and then retrieved the result.
* **isCancelled():** It returns the Boolean value as it returns true if the task was canceled before the completion.
* **isDone():** It returns true if the job is completed successfully else returns false.

### 44. What is the difference between ScheduledExecutorService and ExecutorService interface?

ExecutorServcie and ScheduledExecutorService both are the interfaces of java.util.Concurrent package but scheduledExecutorService provides some additional methods to execute the Runnable and Callable tasks with the delay or every fixed time period.

### 45) Define FutureTask class in Java?

Java FutureTask class provides a base implementation of the Future interface. The result can only be obtained if the execution of one task is completed, and if the computation is not achieved then get method will be blocked. If the execution is completed, then it cannot be re-started and can't be canceled.

**Syntax**

public class FutureTask<V> extends Object implements RunnableFuture<V>