300 Java interview questions | Set 2

[**1**](https://www.javatpoint.com/corejava-interview-questions) [**2**](https://www.javatpoint.com/corejava-interview-questions-2) [**3**](https://www.javatpoint.com/java-multithreading-interview-questions) [**4**](https://www.javatpoint.com/java-collections-interview-questions) [**5**](https://www.javatpoint.com/jdbc-interview-questions)

Core Java - OOPs: Polymorphism Interview Questions

101) What is the difference between compile-time polymorphism and runtime polymorphism?

There are the following differences between compile-time polymorphism and runtime polymorphism.

|  |  |  |
| --- | --- | --- |
| **SN** | **compile-time polymorphism** | **Runtime polymorphism** |
| 1 | In compile-time polymorphism, call to a method is resolved at compile-time. | In runtime polymorphism, call to an overridden method is resolved at runtime. |
| 2 | It is also known as static binding, early binding, or overloading. | It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch. |
| 3 | Overloading is a way to achieve compile-time polymorphism in which, we can define multiple methods or constructors with different signatures. | Overriding is a way to achieve runtime polymorphism in which, we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class. |
| 4 | It provides fast execution because the type of an object is determined at compile-time. | It provides slower execution as compare to compile-time because the type of an object is determined at run-time. |
| 5 | Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time. | Run-time polymorphism provides more flexibility because all the things are resolved at runtime. |

102) What is Runtime Polymorphism?

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

1. **class** Bike{
2. **void** run(){System.out.println("running");}
3. }
4. **class** Splendor **extends** Bike{
5. **void** run(){System.out.println("running safely with 60km");}
6. **public** **static** **void** main(String args[]){
7. Bike b = **new** Splendor();//upcasting
8. b.run();
9. }
10. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Splender)

Output:

running safely with 60km.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

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

103) Can you achieve Runtime Polymorphism by data members?

No, because method overriding is used to achieve runtime polymorphism and data members cannot be overridden. We can override the member functions but not the data members. Consider the example given below.

1. **class** Bike{
2. **int** speedlimit=90;
3. }
4. **class** Honda3 **extends** Bike{
5. **int** speedlimit=150;
6. **public** **static** **void** main(String args[]){
7. Bike obj=**new** Honda3();
8. System.out.println(obj.speedlimit);//90
9. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Honda3)

Output:

90

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

104) What is the difference between static binding and dynamic binding?

In case of the static binding, the type of the object is determined at compile-time whereas, in the dynamic binding, the type of the object is determined at runtime.

**Static Binding**

1. **class** Dog{
2. **private** **void** eat(){System.out.println("dog is eating...");}
4. **public** **static** **void** main(String args[]){
5. Dog d1=**new** Dog();
6. d1.eat();
7. }
8. }

**Dynamic Binding**

1. **class** Animal{
2. **void** eat(){System.out.println("animal is eating...");}
3. }
5. **class** Dog **extends** Animal{
6. **void** eat(){System.out.println("dog is eating...");}
8. **public** **static** **void** main(String args[]){
9. Animal a=**new** Dog();
10. a.eat();
11. }
12. }

[More details.](https://www.javatpoint.com/static-binding-and-dynamic-binding)

105) What is the output of the following Java program?

1. **class** BaseTest
2. {
3. **void** print()
4. {
5. System.out.println("BaseTest:print() called");
6. }
7. }
8. **public** **class** Test **extends** BaseTest
9. {
10. **void** print()
11. {
12. System.out.println("Test:print() called");
13. }
14. **public** **static** **void** main (String args[])
15. {
16. BaseTest b = **new** Test();
17. b.print();
18. }
19. }

**Output**

Test:print() called

**Explanation**

It is an example of Dynamic method dispatch. The type of reference variable b is determined at runtime. At compile-time, it is checked whether that method is present in the Base class. In this case, it is overridden in the child class, therefore, at runtime the derived class method is called.

106) What is Java instanceOf operator?

The instanceof in Java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has a null value, it returns false. Consider the following example.

1. **class** Simple1{
2. **public** **static** **void** main(String args[]){
3. Simple1 s=**new** Simple1();
4. System.out.println(s **instanceof** Simple1);//true
5. }
6. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Simple1)

**Output**

true

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

Core Java - OOPs Concepts: Abstraction Interview Questions

107) What is the abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It displays just the essential things to the user and hides the internal information, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction enables you to focus on what the object does instead of how it does it. Abstraction lets you focus on what the object does instead of how it does it.

In Java, there are two ways to achieve the abstraction.

* Abstract Class
* Interface

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

108) What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

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

109) What is the abstract class?

A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method. Consider the following example.

1. **abstract** **class** Bike{
2. **abstract** **void** run();
3. }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println("running safely");}
6. **public** **static** **void** main(String args[]){
7. Bike obj = **new** Honda4();
8. obj.run();
9. }
10. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Honda4)

**Output**

running safely

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

110) Can there be an abstract method without an abstract class?

No, if there is an abstract method in a class, that class must be abstract.

111) Is the following program written correctly? If yes then what will be the output of the program?

1. **abstract** **class** Calculate
2. {
3. **abstract** **int** multiply(**int** a, **int** b);
4. }
6. **public** **class** Main
7. {
8. **public** **static** **void** main(String[] args)
9. {
10. **int** result = **new** Calculate()
11. {
12. @Override
13. **int** multiply(**int** a, **int** b)
14. {
15. **return** a\*b;
16. }
17. }.multiply(12,32);
18. System.out.println("result = "+result);
19. }
20. }

Yes, the program is written correctly. The Main class provides the definition of abstract method multiply declared in abstract class Calculation. The output of the program will be:

**Output**

384

112) Can you use abstract and final both with a method?

No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

113) Is it possible to instantiate the abstract class?

No, the abstract class can never be instantiated even if it contains a constructor and all of its methods are implemented.

114) What is the interface?

The interface is a blueprint for a class that has static constants and abstract methods. It can be used to achieve full abstraction and multiple inheritance. It is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. However, we need to implement it to define its methods. Since Java 8, we can have the default, static, and private methods in an interface.

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

115) Can you declare an interface method static?

No, because methods of an interface are abstract by default, and we can not use static and abstract together.

116) Can the Interface be final?

No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

117) What is a marker interface?

A Marker interface can be defined as the interface which has no data member and member functions. For example, Serializable, Cloneable are marker interfaces. The marker interface can be declared as follows.

1. **public** **interface** Serializable{
2. }

118) What are the differences between abstract class and interface?

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| An abstract class can have a method body (non-abstract methods). | The interface has only abstract methods. |
| An abstract class can have instance variables. | An interface cannot have instance variables. |
| An abstract class can have the constructor. | The interface cannot have the constructor. |
| An abstract class can have static methods. | The interface cannot have static methods. |
| You can extend one abstract class. | You can implement multiple interfaces. |
| The abstract class **can provide the implementation of the interface**. | The Interface **can't provide the implementation of the abstract class**. |
| The **abstract keyword** is used to declare an abstract class. | The **interface keyword** is used to declare an interface. |
| An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| An **abstract class** can be extended using keyword **extends** | An **interface class** can be implemented using keyword **implements** |
| A Java**abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| **Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

119) Can we define private and protected modifiers for the members in interfaces?

No, they are implicitly public.

120) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

121) How to make a read-only class in Java?

A class can be made read-only by making all of the fields private. The read-only class will have only getter methods which return the private property of the class to the main method. We cannot modify this property because there is no setter method available in the class. Consider the following example.

1. //A Java class which has only getter methods.
2. **public** **class** Student{
3. //private data member
4. **private** String college="AKG";
5. //getter method for college
6. **public** String getCollege(){
7. **return** college;
8. }
9. }

122) How to make a write-only class in Java?

A class can be made write-only by making all of the fields private. The write-only class will have only setter methods which set the value passed from the main method to the private fields. We cannot read the properties of the class because there is no getter method in this class. Consider the following example.

1. //A Java class which has only setter methods.
2. **public** **class** Student{
3. //private data member
4. **private** String college;
5. //getter method for college
6. **public** **void** setCollege(String college){
7. **this**.college=college;
8. }
9. }

123) What are the advantages of Encapsulation in Java?

There are the following advantages of Encapsulation in Java?

* By providing only the setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
* It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
* It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
* The encapsulate class is easy to test. So, it is better for unit testing.
* The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

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.

1. //save as Simple.java
2. **package** mypack;
3. **public** **class** Simple{
4. **public** **static** **void** main(String args[]){
5. System.out.println("Welcome to package");
6. }
7. }

  
[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.

126) How to create packages in Java?

If you are using the programming IDEs like Eclipse, NetBeans, MyEclipse, etc. click on **file->new->project** and eclipse will ask you to enter the name of the package. It will create the project package containing various directories such as src, etc. If you are using an editor like notepad for java programming, use the following steps to create the package.

* Define a package **package\_name**. Create the class with the name **class\_name** and save this file with **your\_class\_name.java**.
* Now compile the file by running the following command on the terminal.
  1. javac -d . your\_class\_name.java

The above command creates the package with the name **package\_name** in the present working directory.

* Now, run the class file by using the absolute class file name, like following.
  1. java package\_name.class\_name

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)

Java: Exception Handling Interview Questions

There is given a list of exception handling interview questions with answers. If you know any exception handling interview question, kindly post it in the comment section.

131) How many types of exception can occur in a Java program?

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

* **Checked Exception:** Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.
* **Unchecked Exception:** Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
* **Error:** Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.

132) What is Exception Handling?

Exception Handling is a mechanism that is used to handle runtime errors. It is used primarily to handle checked exceptions. Exception handling maintains the normal flow of the program. There are mainly two types of exceptions: checked and unchecked. Here, the error is considered as the unchecked exception.

[More details.](https://www.javatpoint.com/exception-handling-and-checked-and-unchecked-exception)

133) Explain the hierarchy of Java Exception classes?

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



134) What is the difference between Checked Exception and Unchecked Exception?

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions, e.g., IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions, e.g., ArithmeticException, NullPointerException, etc. Unchecked exceptions are not checked at compile-time.

[More details.](https://www.javatpoint.com/exception-handling-and-checked-and-unchecked-exception)

135) What is the base class for Error and Exception?

The Throwable class is the base class for Error and Exception.

136) Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. So whatever exceptions are likely to be thrown should be declared in the throws clause of the method. Consider the following example.

1. **public** **class** Main{
2. **public** **static** **void** main(String []args){
3. **try**{
4. **int** a = 1;
5. System.out.println(a/0);
6. }
7. **finally**
8. {
9. System.out.println("rest of the code...");
10. }
11. }
12. }

**Output:**

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

137) What is the output of the following Java program?

1. **public** **class** ExceptionHandlingExample {
2. **public** **static** **void** main(String args[])
3. {
4. **try**
5. {
6. **int** a = 1/0;
7. System.out.println("a = "+a);
8. }
9. **catch**(Exception e){System.out.println(e);}
10. **catch**(ArithmeticException ex){System.out.println(ex);}
11. }
12. }

**Output**

ExceptionHandlingExample.java:10: error: exception ArithmeticException has already been caught

catch(ArithmeticException ex){System.out.println(ex);}

^

1 error

**Explanation**

ArithmaticException is the subclass of Exception. Therefore, it can not be used after Exception. Since Exception is the base class for all the exceptions, therefore, it must be used at last to handle the exception. No class can be used after this.

138) What is finally block?

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. In other words, we can say that finally block is the block which is always executed. Finally block follows try or catch block. If you don't handle the exception, before terminating the program, JVM runs finally block, (if any). The finally block is mainly used to place the cleanup code such as closing a file or closing a connection. Here, we must know that for each try block there can be zero or more catch blocks, but only one finally block. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

  
[More details.](https://www.javatpoint.com/finally-block-in-exception-handling)

139) Can finally block be used without a catch?

Yes, According to the definition of finally block, it must be followed by a try or catch block, therefore, we can use try block instead of catch.[More details.](https://www.javatpoint.com/finally-block-in-exception-handling)

140) Is there any case when finally will not be executed?

Finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).[More details.](https://www.javatpoint.com/finally-block-in-exception-handling)

141) What is the difference between throw and throws?

|  |  |
| --- | --- |
| **throw keyword** | **throws keyword** |
| 1) The **throw** keyword is used to throw an exception explicitly. | The **throws** keyword is used to declare an exception. |
| 2) The checked exceptions cannot be propagated with throw only. | The checked exception can be propagated with throws |
| 3) The **throw** keyword is followed by an instance. | The **throws** keyword is followed by class. |
| 4) The **throw** keyword is used within the method. | The **throws** keyword is used with the method signature. |
| 5) You cannot throw multiple exceptions. | You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException. |

[More details.](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)

142) What is the output of the following Java program?

1. **public** **class** Main{
2. **public** **static** **void** main(String []args){
3. **try**
4. {
5. **throw** 90;
6. }
7. **catch**(**int** e){
8. System.out.println("Caught the exception "+e);
9. }
11. }
12. }

**Output**

Main.java:6: error: incompatible types: int cannot be converted to Throwable

throw 90;

^

Main.java:8: error: unexpected type

catch(int e){

^

required: class

found: int

2 errors

**Explanation**

In Java, the throwable objects can only be thrown. If we try to throw an integer object, The compiler will show an error since we can not throw basic data type from a block of code.

143) What is the output of the following Java program?

1. **class** Calculation **extends** Exception
2. {
3. **public** Calculation()
4. {
5. System.out.println("Calculation class is instantiated");
6. }
7. **public** **void** add(**int** a, **int** b)
8. {
9. System.out.println("The sum is "+(a+b));
10. }
11. }
12. **public** **class** Main{
13. **public** **static** **void** main(String []args){
14. **try**
15. {
16. **throw** **new** Calculation();
17. }
18. **catch**(Calculation c){
19. c.add(10,20);
20. }
21. }
22. }

**Output**

Calculation class is instantiated

The sum is 30

**Explanation**

The object of Calculation is thrown from the try block which is caught in the catch block. The add() of Calculation class is called with the integer values 10 and 20 by using the object of this class. Therefore there sum 30 is printed. The object of the Main class can only be thrown in the case when the type of the object is throwable. To do so, we need to extend the throwable class.

144) Can an exception be rethrown?

Yes.

145) Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

Yes but only unchecked exception not checked.

[More details.](https://www.javatpoint.com/exception-handling-with-method-overriding)

146) What is exception propagation?

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This procedure is called exception propagation. By default, checked exceptions are not propagated.

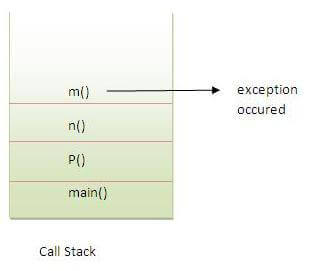
1. **class** TestExceptionPropagation1{
2. **void** m(){
3. **int** data=50/0;
4. }
5. **void** n(){
6. m();
7. }
8. **void** p(){
9. **try**{
10. n();
11. }**catch**(Exception e){System.out.println("exception handled");}
12. }
13. **public** **static** **void** main(String args[]){
14. TestExceptionPropagation1 obj=**new** TestExceptionPropagation1();
15. obj.p();
16. System.out.println("normal flow...");
17. }
18. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestExceptionPropagation1)

**Output:**

exception handled

normal flow...

  
[More details.](https://www.javatpoint.com/exception-propagation)

147) What is the output of the following Java program?

1. **public** **class** Main
2. {
3. **void** a()
4. {
5. **try**{
6. System.out.println("a(): Main called");
7. b();
8. }**catch**(Exception e)
9. {
10. System.out.println("Exception is caught");
11. }
12. }
13. **void** b() **throws** Exception
14. {
15. **try**{
16. System.out.println("b(): Main called");
17. c();
18. }**catch**(Exception e){
19. **throw** **new** Exception();
20. }
21. **finally**
22. {
23. System.out.println("finally block is called");
24. }
25. }
26. **void** c() **throws** Exception
27. {
28. **throw** **new** Exception();
29. }
31. **public** **static** **void** main (String args[])
32. {
33. Main m = **new** Main();
34. m.a();
35. }
36. }

**Output**

a(): Main called

b(): Main called

finally block is called

Exception is caught

**Explanation**

In the main method, a() of Main is called which prints a message and call b(). The method b() prints some message and then call c(). The method c() throws an exception which is handled by the catch block of method b. However, It propagates this exception by using **throw Exception()** to be handled by the method a(). As we know, finally block is always executed therefore the finally block in the method b() is executed first and prints a message. At last, the exception is handled by the catch block of the method a().

148) What is the output of the following Java program?

1. **public** **class** Calculation
2. {
3. **int** a;
4. **public** Calculation(**int** a)
5. {
6. **this**.a = a;
7. }
8. **public** **int** add()
9. {
10. a = a+10;
11. **try**
12. {
13. a = a+10;
14. **try**
15. {
16. a = a\*10;
17. **throw** **new** Exception();
18. }**catch**(Exception e){
19. a = a - 10;
20. }
21. }**catch**(Exception e)
22. {
23. a = a - 10;
24. }
25. **return** a;
26. }
28. **public** **static** **void** main (String args[])
29. {
30. Calculation c = **new** Calculation(10);
31. **int** result = c.add();
32. System.out.println("result = "+result);
33. }
34. }

**Output**

result = 290

**Explanation**

The instance variable a of class Calculation is initialized to 10 using the class constructor which is called while instantiating the class. The add method is called which returns an integer value result. In add() method, a is incremented by 10 to be 20. Then, in the first try block, 10 is again incremented by 10 to be 30. In the second try block, a is multiplied by 10 to be 300. The second try block throws the exception which is caught by the catch block associated with this try block. The catch block again alters the value of a by decrementing it by 10 to make it 290. Thus the add() method returns 290 which is assigned to result. However, the catch block associated with the outermost try block will never be executed since there is no exception which can be handled by this catch block.

Java: String Handling Interview Questions

There is given a list of string handling interview questions with short and pointed answers. If you know any string handling interview question, kindly post it in the comment section.

149) What is String Pool?

String pool is the space reserved in the heap memory that can be used to store the strings. The main advantage of using the String pool is whenever we create a string literal; the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. Therefore, it saves the memory by avoiding the duplicacy.



150) What is the meaning of immutable regarding String?

The simple meaning of immutable is unmodifiable or unchangeable. In Java, String is immutable, i.e., once string object has been created, its value can't be changed. Consider the following example for better understanding.

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring)

**Output:**

Sachin

[More details.](https://www.javatpoint.com/immutable-string)

151) Why are the objects immutable in java?

Because Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object "sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.

  
[More details.](https://www.javatpoint.com/immutable-string)

152) How many ways can we create the string object?

1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. String objects are stored in a special memory area known as the **string constant pool** For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance

2) By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the constant string pool. The variable s will refer to the object in a heap (non-pool).

153) How many objects will be created in the following code?

1. String s1="Welcome";
2. String s2="Welcome";
3. String s3="Welcome";

Only one object will be created using the above code because strings in Java are immutable.

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

154) Why java uses the concept of the string literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

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

155) How many objects will be created in the following code?

1. String s = **new** String("Welcome");

Two objects, one in string constant pool and other in non-pool(heap).

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

156) What is the output of the following Java program?

1. **public** **class** Test
3. **public** **static** **void** main (String args[])
4. {
5. String a = **new** String("Sharma is a good player");
6. String b = "Sharma is a good player";
7. **if**(a == b)
8. {
9. System.out.println("a == b");
10. }
11. **if**(a.equals(b))
12. {
13. System.out.println("a equals b");
14. }
15. }

**Output**

a equals b

**Explanation**

The operator **==** also check whether the references of the two string objects are equal or not. Although both of the strings contain the same content, their references are not equal because both are created by different ways(Constructor and String literal) therefore, **a == b** is unequal. On the other hand, the equal() method always check for the content. Since their content is equal hence, **a equals b** is printed.

157) What is the output of the following Java program?

1. **public** **class** Test
2. {
3. **public** **static** **void** main (String args[])
4. {
5. String s1 = "Sharma is a good player";
6. String s2 = **new** String("Sharma is a good player");
7. s2 = s2.intern();
8. System.out.println(s1 ==s2);
9. }
10. }

**Output**

true

**Explanation**

The intern method returns the String object reference from the string pool. In this case, s1 is created by using string literal whereas, s2 is created by using the String pool. However, s2 is changed to the reference of s1, and the operator **==** returns true.

158) What are the differences between String and StringBuffer?

The differences between the String and StringBuffer is given in the table below.

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | The String class is immutable. | The StringBuffer class is mutable. |
| 2) | The String is slow and consumes more memory when you concat too many strings because every time it creates a new instance. | The StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | The String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | The StringBuffer class doesn't override the equals() method of Object class. |

159) What are the differences between StringBuffer and StringBuilder?

The differences between the StringBuffer and StringBuilder is given below.

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized*, i.e., thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized*,i.e., not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

160) How can we create an immutable class in Java?

We can create an immutable class by defining a final class having all of its members as final. Consider the following example.

1. **public** **final** **class** Employee{
2. **final** String pancardNumber;
4. **public** Employee(String pancardNumber){
5. **this**.pancardNumber=pancardNumber;
6. }
8. **public** String getPancardNumber(){
9. **return** pancardNumber;
10. }
12. }

[More details.](https://www.javatpoint.com/how-to-create-immutable-class)

161) What is the purpose of toString() method in Java?

The toString() method returns the string representation of an object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object, etc. depending upon your implementation. By overriding the toString() method of the Object class, we can return the values of the object, so we don't need to write much code. Consider the following example.

1. **class** Student{
2. **int** rollno;
3. String name;
4. String city;
6. Student(**int** rollno, String name, String city){
7. **this**.rollno=rollno;
8. **this**.name=name;
9. **this**.city=city;
10. }
12. **public** String toString(){//overriding the toString() method
13. **return** rollno+" "+name+" "+city;
14. }
15. **public** **static** **void** main(String args[]){
16. Student s1=**new** Student(101,"Raj","lucknow");
17. Student s2=**new** Student(102,"Vijay","ghaziabad");
19. System.out.println(s1);//compiler writes here s1.toString()
20. System.out.println(s2);//compiler writes here s2.toString()
21. }
22. }

**Output:**

101 Raj lucknow

102 Vijay ghaziabad

[More details.](https://www.javatpoint.com/understanding-toString()-method)

162) Why CharArray() is preferred over String to store the password?

String stays in the string pool until the garbage is collected. If we store the password into a string, it stays in the memory for a longer period, and anyone having the memory-dump can extract the password as clear text. On the other hand, Using CharArray allows us to set it to blank whenever we are done with the password. It avoids the security threat with the string by enabling us to control the memory.

163) Write a Java program to count the number of words present in a string?

**Program:**

1. **public** **class** Test
2. {
3. **public** **static** **void** main (String args[])
4. {
5. String s = "Sharma is a good player and he is so punctual";
6. String words[] = s.split(" ");
7. System.out.println("The Number of words present in the string are : "+words.length);
8. }
9. }

**Output**

The Number of words present in the string are : 10

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?

1. **import** java.util.regex.\*;
2. **class** RegexExample2{
3. **public** **static** **void** main(String args[]){
4. System.out.println(Pattern.matches(".s", "as")); //line 4
5. System.out.println(Pattern.matches(".s", "mk")); //line 5
6. System.out.println(Pattern.matches(".s", "mst")); //line 6
7. System.out.println(Pattern.matches(".s", "amms")); //line 7
8. System.out.println(Pattern.matches("..s", "mas")); //line 8
9. }}

**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.

1. **class** Java\_Outer\_class{
2. //code
3. **class** Java\_Nested\_class{
4. //code
5. }
6. }

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. |

172) Is there any difference between nested classes and inner classes?

Yes, inner classes are non-static nested classes. In other words, we can say that inner classes are the part of nested classes.

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

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?

1. **public** **class** Person {
2. String name, age, address;
3. **class** Employee{
4. **float** salary=10000;
5. }
6. **class** BusinessMen{
7. **final** String gstin="£4433drt3$";
8. }
9. **public** **static** **void** main (String args[])
10. {
11. Person p = **new** Person();
12. }
13. }

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.

1. **abstract** **class** Person{
2. **abstract** **void** eat();
3. }
4. **class** TestAnonymousInner{
5. **public** **static** **void** main(String args[]){
6. Person p=**new** Person(){
7. **void** eat(){System.out.println("nice fruits");}
8. };
9. p.eat();
10. }
11. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAnnonymousInner)

Output:

nice fruits

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

1. **interface** Eatable{
2. **void** eat();
3. }
4. **class** TestAnnonymousInner1{
5. **public** **static** **void** main(String args[]){
6. Eatable e=**new** Eatable(){
7. **public** **void** eat(){System.out.println("nice fruits");}
8. };
9. e.eat();
10. }
11. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAnnonymousInner1)

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.

1. **interface** interface\_name{
2. ...
3. **interface** nested\_interface\_name{
4. ...
5. }
6. }

[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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestGarbage1)

object is garbage collected

object is garbage collected

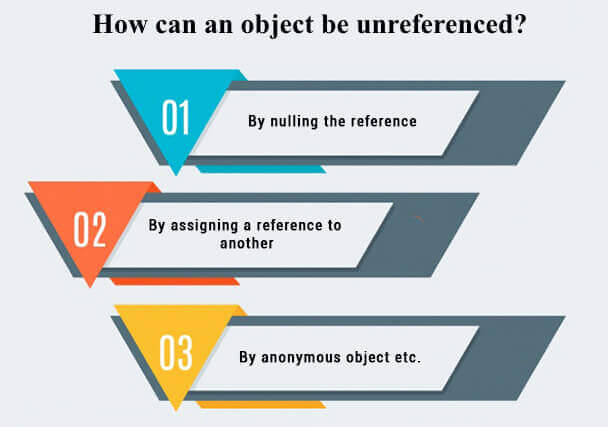
181) How is garbage collection controlled?

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.

1. **public** **class** FinalizeTest {
2. **int** j=12;
3. **void** add()
4. {
5. j=j+12;
6. System.out.println("J="+j);
7. }
8. **public** **void** finalize()
9. {
10. System.out.println("Object is garbage collected");
11. }
12. **public** **static** **void** main(String[] args) {
13. **new** FinalizeTest().add();
14. System.gc();
15. **new** FinalizeTest().add();
16. }
17. }

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.

1. **public** **class** Runtime1{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("notepad");//will open a new notepad
4. }
5. }

I/O Interview Questions

189) Give the hierarchy of InputStream and OutputStream classes.

**OutputStream Hierarchy**

**InputStream 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.

1. **import** java.io.FileOutputStream;
2. **public** **class** FileOutputStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
6. fout.write(65);
7. fout.close();
8. System.out.println("success...");
9. }**catch**(Exception e){System.out.println(e);}
10. }
11. }

**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.

1. **import** java.io.FileInputStream;
2. **public** **class** DataStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileInputStream fin=**new** FileInputStream("D:\\testout.txt");
6. **int** i=fin.read();
7. System.out.print((**char**)i);
9. fin.close();
10. }**catch**(Exception e){System.out.println(e);}
11. }
12. }

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.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **import** java.security.PermissionCollection;
4. **public** **class** FilePermissionExample{
5. **public** **static** **void** main(String[] args) **throws** IOException {
6. String srg = "D:\\IO Package\\java.txt";
7. FilePermission file1 = **new** FilePermission("D:\\IO Package\\-", "read");
8. PermissionCollection permission = file1.newPermissionCollection();
9. permission.add(file1);
10. FilePermission file2 = **new** FilePermission(srg, "write");
11. permission.add(file2);
12. **if**(permission.implies(**new** FilePermission(srg, "read,write"))) {
13. System.out.println("Read, Write permission is granted for the path "+srg );
14. }**else** {
15. System.out.println("No Read, Write permission is granted for the path "+srg);            }
16. }
17. }

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.
  1. **import** java.io.BufferedReader;
  2. **import** java.io.IOException;
  3. **import** java.io.InputStreamReader;
  4. **public** **class** Person
  5. {
  6. **public** **static** **void** main(String[] args) **throws** IOException
  7. {
  8. System.out.println("Enter the name of the person");
  9. BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(System.in));
  10. String name = reader.readLine();
  11. System.out.println(name);
  12. }
  13. }
* **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.
  1. **import** java.util.\*;
  2. **public** **class** ScannerClassExample2 {
  3. **public** **static** **void** main(String args[]){
  4. String str = "Hello/This is JavaTpoint/My name is Abhishek.";
  5. //Create scanner with the specified String Object
  6. Scanner scanner = **new** Scanner(str);
  7. System.out.println("Boolean Result: "+scanner.hasNextBoolean());
  8. //Change the delimiter of this scanner
  9. scanner.useDelimiter("/");
  10. //Printing the tokenized Strings
  11. System.out.println("---Tokenizes String---");
  12. **while**(scanner.hasNext()){
  13. System.out.println(scanner.next());
  14. }
  15. //Display the new delimiter
  16. System.out.println("Delimiter used: " +scanner.delimiter());
  17. scanner.close();
  18. }
  19. }
* **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.
  1. **import** java.io.Console;
  2. **class** ReadStringTest{
  3. **public** **static** **void** main(String args[]){
  4. Console c=System.console();
  5. System.out.println("Enter your name: ");
  6. String n=c.readLine();
  7. System.out.println("Welcome "+n);
  8. }
  9. }

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.

1. **import** java.io.FileInputStream;
2. **import** java.io.FileOutputStream;
3. **import** java.io.IOException;
4. **import** java.io.NotSerializableException;
5. **import** java.io.ObjectInputStream;
6. **import** java.io.ObjectOutputStream;
7. **import** java.io.Serializable;
8. **class** Person **implements** Serializable
9. {
10. String name = " ";
11. **public** Person(String name)
12. {
13. **this**.name = name;
14. }
15. }
16. **class** Employee **extends** Person
17. {
18. **float** salary;
19. **public** Employee(String name, **float** salary)
20. {
21. **super**(name);
22. **this**.salary = salary;
23. }
24. **private** **void** writeObject(ObjectOutputStream out) **throws** IOException
25. {
26. **throw** **new** NotSerializableException();
27. }
28. **private** **void** readObject(ObjectInputStream in) **throws** IOException
29. {
30. **throw** **new** NotSerializableException();
31. }
33. }
34. **public** **class** Test
35. {
36. **public** **static** **void** main(String[] args)
37. **throws** Exception
38. {
39. Employee emp = **new** Employee("Sharma", 10000);
41. System.out.println("name = " + emp.name);
42. System.out.println("salary = " + emp.salary);
44. FileOutputStream fos = **new** FileOutputStream("abc.ser");
45. ObjectOutputStream oos = **new** ObjectOutputStream(fos);
47. oos.writeObject(emp);
49. oos.close();
50. fos.close();
52. System.out.println("Object has been serialized");
54. FileInputStream f = **new** FileInputStream("ab.txt");
55. ObjectInputStream o = **new** ObjectInputStream(f);
57. Employee emp1 = (Employee)o.readObject();
59. o.close();
60. f.close();
62. System.out.println("Object has been deserialized");
64. System.out.println("name = " + emp1.name);
65. System.out.println("salary = " + emp1.salary);
66. }
67. }

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.

1. **import** java.io.\*;
2. **class** Depersist{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. ObjectInputStream in=**new** ObjectInputStream(**new** FileInputStream("f.txt"));
6. Student s=(Student)in.readObject();
7. System.out.println(s.id+" "+s.name);
9. in.close();
10. }
11. }

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*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyServer {
4. **public** **static** **void** main(String[] args){
5. **try**{
6. ServerSocket ss=**new** ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=**new** DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

*File: MyClient.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyClient {
4. **public** **static** **void** main(String[] args) {
5. **try**{
6. Socket s=**new** Socket("localhost",6666);
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

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.

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** InetDemo{
4. **public** **static** **void** main(String[] args){
5. **try**{
6. InetAddress ip=InetAddress.getByName("195.201.10.8");
8. System.out.println("Host Name: "+ip.getHostName());
9. }**catch**(Exception e){System.out.println(e);}
10. }
11. }

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?

1. **class** Simple{
2. **public** Simple()
3. {
4. System.out.println("Constructor of Simple class is invoked");
5. }
6. **void** message(){System.out.println("Hello Java");}
7. }
9. **class** Test1{
10. **public** **static** **void** main(String args[]){
11. **try**{
12. Class c=Class.forName("Simple");
13. Simple s=(Simple)c.newInstance();
14. s.message();
15. }**catch**(Exception e){System.out.println(e);}
16. }
17. }

**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?

1. **public** **class** Test1
2. {
3. **public** **static** **void** main(String[] args) {
4. Integer i = **new** Integer(201);
5. Integer j = **new** Integer(201);
6. **if**(i == j)
7. {
8. System.out.println("hello");
9. }
10. **else**
11. {
12. System.out.println("bye");
13. }
14. }
15. }

**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.

1. **class** Singleton{
2. **private** **static** Singleton single\_instance = **null**;
3. **int** i;
4. **private** Singleton ()
5. {
6. i=90;
7. }
8. **public** **static** Singleton getInstance()
9. {
10. **if**(single\_instance == **null**)
11. {
12. single\_instance = **new** Singleton();
13. }
14. **return** single\_instance;
15. }
16. }
17. **public** **class** Main
18. {
19. **public** **static** **void** main (String args[])
20. {
21. Singleton first = Singleton.getInstance();
22. System.out.println("First instance integer value:"+first.i);
23. first.i=first.i+90;
24. Singleton second = Singleton.getInstance();
25. System.out.println("Second instance integer value:"+second.i);
26. }
27. }

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

**Program**

1. **class** A{
2. **public** **static** **void** main(String args[]){
4. **for**(**int** i=0;i<args.length;i++)
5. System.out.println(args[i]);
7. }
8. }
9. compile by > javac A.java
10. 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?

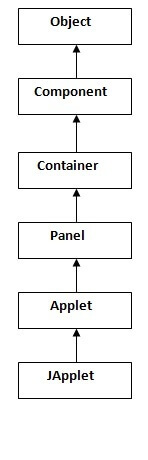
Lightweight components are the one which does not go with the native call to obtain the graphical units. They share their parent component graphical units to render them. For example, Swing components, and JavaFX Components.

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.

1. **import** java.util.\*;
2. **public** **class** LocaleExample {
3. **public** **static** **void** main(String[] args) {
4. Locale locale=Locale.getDefault();
5. //Locale locale=new Locale("fr","fr");//for the specific locale
7. System.out.println(locale.getDisplayCountry());
8. System.out.println(locale.getDisplayLanguage());
9. System.out.println(locale.getDisplayName());
10. System.out.println(locale.getISO3Country());
11. System.out.println(locale.getISO3Language());
12. System.out.println(locale.getLanguage());
13. System.out.println(locale.getCountry());
15. }
16. }

**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.

1. //Employee.java
2. **package** mypack;
3. **public** **class** Employee **implements** java.io.Serializable{
4. **private** **int** id;
5. **private** String name;
6. **public** Employee(){}
7. **public** **void** setId(**int** id){**this**.id=id;}
8. **public** **int** getId(){**return** id;}
9. **public** **void** setName(String name){**this**.name=name;}
10. **public** String getName(){**return** name;}
11. }

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.

Core Java: Data Structure interview questions

248) How to perform Bubble Sort in Java?

Consider the following program to perform Bubble sort in Java.

1. **public** **class** BubbleSort {
2. **public** **static** **void** main(String[] args) {
3. **int**[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
4. **for**(**int** i=0;i<10;i++)
5. {
6. **for** (**int** j=0;j<10;j++)
7. {
8. **if**(a[i]<a[j])
9. {
10. **int** temp = a[i];
11. a[i]=a[j];
12. a[j] = temp;
13. }
14. }
15. }
16. System.out.println("Printing Sorted List ...");
17. **for**(**int** i=0;i<10;i++)
18. {
19. System.out.println(a[i]);
20. }
21. }
22. }

**Output:**

Printing Sorted List . . .

7

9

10

12

23

34

34

44

78

101

249) How to perform Binary Search in Java?

Consider the following program to perform the binary search in Java.

1. **import** java.util.\*;
2. **public** **class** BinarySearch {
3. **public** **static** **void** main(String[] args) {
4. **int**[] arr = {16, 19, 20, 23, 45, 56, 78, 90, 96, 100};
5. **int** item, location = -1;
6. System.out.println("Enter the item which you want to search");
7. Scanner sc = **new** Scanner(System.in);
8. item = sc.nextInt();
9. location = binarySearch(arr,0,9,item);
10. **if**(location != -1)
11. System.out.println("the location of the item is "+location);
12. **else**
13. System.out.println("Item not found");
14. }
15. **public** **static** **int** binarySearch(**int**[] a, **int** beg, **int** end, **int** item)
16. {
17. **int** mid;
18. **if**(end >= beg)
19. {
20. mid = (beg + end)/2;
21. **if**(a[mid] == item)
22. {
23. **return** mid+1;
24. }
25. **else** **if**(a[mid] < item)
26. {
27. **return** binarySearch(a,mid+1,end,item);
28. }
29. **else**
30. {
31. **return** binarySearch(a,beg,mid-1,item);
32. }
33. }
34. **return** -1;
35. }
36. }

**Output:**

Enter the item which you want to search

45

the location of the item is 5

250) How to perform Selection Sort in Java?

Consider the following program to perform selection sort in Java.

1. **public** **class** SelectionSort {
2. **public** **static** **void** main(String[] args) {
3. **int**[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
4. **int** i,j,k,pos,temp;
5. **for**(i=0;i<10;i++)
6. {
7. pos = smallest(a,10,i);
8. temp = a[i];
9. a[i]=a[pos];
10. a[pos] = temp;
11. }
12. System.out.println("\nprinting sorted elements...\n");
13. **for**(i=0;i<10;i++)
14. {
15. System.out.println(a[i]);
16. }
17. }
18. **public** **static** **int** smallest(**int** a[], **int** n, **int** i)
19. {
20. **int** small,pos,j;
21. small = a[i];
22. pos = i;
23. **for**(j=i+1;j<10;j++)
24. {
25. **if**(a[j]<small)
26. {
27. small = a[j];
28. pos=j;
29. }
30. }
31. **return** pos;
32. }
33. }

**Output:**

printing sorted elements...

7

9

10

12

23

23

34

44

78

101

251) How to perform Linear Search in Java?

Consider the following program to perform Linear search in Java.

1. **import** java.util.Scanner;
3. **public** **class** Leniear\_Search {
4. **public** **static** **void** main(String[] args) {
5. **int**[] arr = {10, 23, 15, 8, 4, 3, 25, 30, 34, 2, 19};
6. **int** item,flag=0;
7. Scanner sc = **new** Scanner(System.in);
8. System.out.println("Enter Item ?");
9. item = sc.nextInt();
10. **for**(**int** i = 0; i<10; i++)
11. {
12. **if**(arr[i]==item)
13. {
14. flag = i+1;
15. **break**;
16. }
17. **else**
18. flag = 0;
19. }
20. **if**(flag != 0)
21. {
22. System.out.println("Item found at location" + flag);
23. }
24. **else**
25. System.out.println("Item not found");
27. }
28. }

**Output:**

Enter Item ?

23

Item found at location 2

Enter Item ?

22

Item not found

252) How to perform merge sort in Java?

Consider the following program to perform merge sort in Java.

1. **public** **class** MyMergeSort
2. {
3. **void** merge(**int** arr[], **int** beg, **int** mid, **int** end)
4. {
6. **int** l = mid - beg + 1;
7. **int** r = end - mid;
9. intLeftArray[] = **new** **int** [l];
10. intRightArray[] = **new** **int** [r];
12. **for** (**int** i=0; i<l; ++i)
13. LeftArray[i] = arr[beg + i];
15. **for** (**int** j=0; j<r; ++j)
16. RightArray[j] = arr[mid + 1+ j];

19. **int** i = 0, j = 0;
20. **int** k = beg;
21. **while** (i<l&&j<r)
22. {
23. **if** (LeftArray[i] <= RightArray[j])
24. {
25. arr[k] = LeftArray[i];
26. i++;
27. }
28. **else**
29. {
30. arr[k] = RightArray[j];
31. j++;
32. }
33. k++;
34. }
35. **while** (i<l)
36. {
37. arr[k] = LeftArray[i];
38. i++;
39. k++;
40. }
42. **while** (j<r)
43. {
44. arr[k] = RightArray[j];
45. j++;
46. k++;
47. }
48. }
50. **void** sort(**int** arr[], **int** beg, **int** end)
51. {
52. **if** (beg<end)
53. {
54. **int** mid = (beg+end)/2;
55. sort(arr, beg, mid);
56. sort(arr , mid+1, end);
57. merge(arr, beg, mid, end);
58. }
59. }
60. **public** **static** **void** main(String args[])
61. {
62. intarr[] = {90,23,101,45,65,23,67,89,34,23};
63. MyMergeSort ob = **new** MyMergeSort();
64. ob.sort(arr, 0, arr.length-1);
66. System.out.println("\nSorted array");
67. **for**(**int** i =0; i<arr.length;i++)
68. {
69. System.out.println(arr[i]+"");
70. }
71. }
72. }

**Output:**

Sorted array

23

23

23

34

45

65

67

89

90

101

253) How to perform quicksort in Java?

Consider the following program to perform quicksort in Java.

1. **public** **class** QuickSort {
2. **public** **static** **void** main(String[] args) {
3. **int** i;
4. **int**[] arr={90,23,101,45,65,23,67,89,34,23};
5. quickSort(arr, 0, 9);
6. System.out.println("\n The sorted array is: \n");
7. **for**(i=0;i<10;i++)
8. System.out.println(arr[i]);
9. }
10. **public** **static** **int** partition(**int** a[], **int** beg, **int** end)
11. {
13. **int** left, right, temp, loc, flag;
14. loc = left = beg;
15. right = end;
16. flag = 0;
17. **while**(flag != 1)
18. {
19. **while**((a[loc] <= a[right]) && (loc!=right))
20. right--;
21. **if**(loc==right)
22. flag =1;
23. elseif(a[loc]>a[right])
24. {
25. temp = a[loc];
26. a[loc] = a[right];
27. a[right] = temp;
28. loc = right;
29. }
30. **if**(flag!=1)
31. {
32. **while**((a[loc] >= a[left]) && (loc!=left))
33. left++;
34. **if**(loc==left)
35. flag =1;
36. elseif(a[loc] <a[left])
37. {
38. temp = a[loc];
39. a[loc] = a[left];
40. a[left] = temp;
41. loc = left;
42. }
43. }
44. }
45. returnloc;
46. }
47. **static** **void** quickSort(**int** a[], **int** beg, **int** end)
48. {
50. **int** loc;
51. **if**(beg<end)
52. {
53. loc = partition(a, beg, end);
54. quickSort(a, beg, loc-1);
55. quickSort(a, loc+1, end);
56. }
57. }
58. }

**Output:**

The sorted array is:

23

23

23

34

45

65

67

89

90

101

254) Write a program in Java to create a doubly linked list containing n nodes.

Consider the following program to create a doubly linked list containing n nodes.

1. **public** **class** CountList {
3. //Represent a node of the doubly linked list
5. **class** Node{
6. **int** data;
7. Node previous;
8. Node next;
10. **public** Node(**int** data) {
11. **this**.data = data;
12. }
13. }
15. //Represent the head and tail of the doubly linked list
16. Node head, tail = **null**;
18. //addNode() will add a node to the list
19. **public** **void** addNode(**int** data) {
20. //Create a new node
21. Node newNode = **new** Node(data);
23. //If list is empty
24. **if**(head == **null**) {
25. //Both head and tail will point to newNode
26. head = tail = newNode;
27. //head's previous will point to null
28. head.previous = **null**;
29. //tail's next will point to null, as it is the last node of the list
30. tail.next = **null**;
31. }
32. **else** {
33. //newNode will be added after tail such that tail's next will point to newNode
34. tail.next = newNode;
35. //newNode's previous will point to tail
36. newNode.previous = tail;
37. //newNode will become new tail
38. tail = newNode;
39. //As it is last node, tail's next will point to null
40. tail.next = **null**;
41. }
42. }
44. //countNodes() will count the nodes present in the list
45. **public** **int** countNodes() {
46. **int** counter = 0;
47. //Node current will point to head
48. Node current = head;
50. **while**(current != **null**) {
51. //Increment the counter by 1 for each node
52. counter++;
53. current = current.next;
54. }
55. **return** counter;
56. }
58. //display() will print out the elements of the list
59. **public** **void** display() {
60. //Node current will point to head
61. Node current = head;
62. **if**(head == **null**) {
63. System.out.println("List is empty");
64. **return**;
65. }
66. System.out.println("Nodes of doubly linked list: ");
67. **while**(current != **null**) {
68. //Prints each node by incrementing the pointer.
70. System.out.print(current.data + " ");
71. current = current.next;
72. }
73. }
75. **public** **static** **void** main(String[] args) {
77. CountList dList = **new** CountList();
78. //Add nodes to the list
79. dList.addNode(1);
80. dList.addNode(2);
81. dList.addNode(3);
82. dList.addNode(4);
83. dList.addNode(5);
85. //Displays the nodes present in the list
86. dList.display();
88. //Counts the nodes present in the given list
89. System.out.println("\nCount of nodes present in the list: " + dList.countNodes());
90. }
91. }

**Output:**

Nodes of doubly linked list:

1 2 3 4 5

Count of nodes present in the list: 5

255) Write a program in Java to find the maximum and minimum value node from a circular linked list.

Consider the following program.

1. **public** **class** MinMax {
2. //Represents the node of list.
3. **public** **class** Node{
4. **int** data;
5. Node next;
6. **public** Node(**int** data) {
7. **this**.data = data;
8. }
9. }
11. //Declaring head and tail pointer as null.
12. **public** Node head = **null**;
13. **public** Node tail = **null**;
15. //This function will add the new node at the end of the list.
16. **public** **void** add(**int** data){
17. //Create new node
18. Node newNode = **new** Node(data);
19. //Checks if the list is empty.
20. **if**(head == **null**) {
21. //If list is empty, both head and tail would point to new node.
22. head = newNode;
23. tail = newNode;
24. newNode.next = head;
25. }
26. **else** {
27. //tail will point to new node.
28. tail.next = newNode;
29. //New node will become new tail.
30. tail = newNode;
31. //Since, it is circular linked list tail will points to head.
32. tail.next = head;
33. }
34. }
36. //Finds out the minimum value node in the list
37. **public** **void** minNode() {
38. Node current = head;
39. //Initializing min to initial node data
40. **int** min = head.data;
41. **if**(head == **null**) {
42. System.out.println("List is empty");
43. }
44. **else** {
45. **do**{
46. //If current node's data is smaller than min
47. //Then replace value of min with current node's data
48. **if**(min > current.data) {
49. min = current.data;
50. }
51. current= current.next;
52. }**while**(current != head);
54. System.out.println("Minimum value node in the list: "+ min);
55. }
56. }
58. //Finds out the maximum value node in the list
59. **public** **void** maxNode() {
60. Node current = head;
61. //Initializing max to initial node data
62. **int** max = head.data;
63. **if**(head == **null**) {
64. System.out.println("List is empty");
65. }
66. **else** {
67. **do**{
68. //If current node's data is greater than max
69. //Then replace value of max with current node's data
70. **if**(max < current.data) {
71. max = current.data;
72. }
73. current= current.next;
74. }**while**(current != head);
76. System.out.println("Maximum value node in the list: "+ max);
77. }
78. }
80. **public** **static** **void** main(String[] args) {
81. MinMax cl = **new** MinMax();
82. //Adds data to the list
83. cl.add(5);
84. cl.add(20);
85. cl.add(10);
86. cl.add(1);
87. //Prints the minimum value node in the list
88. cl.minNode();
89. //Prints the maximum value node in the list
90. cl.maxNode();
91. }
92. }

**Output:**

Minimum value node in the list: 1

Maximum value node in the list: 20

256) Write a program in Java to calculate the difference between the sum of the odd level and even level nodes of a Binary Tree.

Consider the following program.

1. **import** java.util.LinkedList;
2. **import** java.util.Queue;
4. **public** **class** DiffOddEven {
6. //Represent a node of binary tree
7. **public** **static** **class** Node{
8. **int** data;
9. Node left;
10. Node right;
12. **public** Node(**int** data){
13. //Assign data to the new node, set left and right children to null
14. **this**.data = data;
15. **this**.left = **null**;
16. **this**.right = **null**;
17. }
18. }
20. //Represent the root of binary tree
21. **public** Node root;
23. **public** DiffOddEven(){
24. root = **null**;
25. }
27. //difference() will calculate the difference between sum of odd and even levels of binary tree
28. **public** **int** difference() {
29. **int** oddLevel = 0, evenLevel = 0, diffOddEven = 0;
31. //Variable nodesInLevel keep tracks of number of nodes in each level
32. **int** nodesInLevel = 0;
34. //Variable currentLevel keep track of level in binary tree
35. **int** currentLevel = 0;
37. //Queue will be used to keep track of nodes of tree level-wise
38. Queue<Node> queue = **new** LinkedList<Node>();
40. //Check if root is null
41. **if**(root == **null**) {
42. System.out.println("Tree is empty");
43. **return** 0;
44. }
45. **else** {
46. //Add root node to queue as it represents the first level
47. queue.add(root);
48. currentLevel++;
50. **while**(queue.size() != 0) {
52. //Variable nodesInLevel will hold the size of queue i.e. number of elements in queue
53. nodesInLevel = queue.size();
55. **while**(nodesInLevel > 0) {
56. Node current = queue.remove();
58. //Checks if currentLevel is even or not.
59. **if**(currentLevel % 2 == 0)
60. //If level is even, add nodes's to variable evenLevel
61. evenLevel += current.data;
62. **else**
63. //If level is odd, add nodes's to variable oddLevel
64. oddLevel += current.data;
66. //Adds left child to queue
67. **if**(current.left != **null**)
68. queue.add(current.left);
69. //Adds right child to queue
70. **if**(current.right != **null**)
71. queue.add(current.right);
72. nodesInLevel--;
73. }
74. currentLevel++;
75. }
76. //Calculates difference between oddLevel and evenLevel
77. diffOddEven = Math.abs(oddLevel - evenLevel);
78. }
79. **return** diffOddEven;
80. }
82. **public** **static** **void** main (String[] args) {
84. DiffOddEven bt = **new** DiffOddEven();
85. //Add nodes to the binary tree
86. bt.root = **new** Node(1);
87. bt.root.left = **new** Node(2);
88. bt.root.right = **new** Node(3);
89. bt.root.left.left = **new** Node(4);
90. bt.root.right.left = **new** Node(5);
91. bt.root.right.right = **new** Node(6);
93. //Display the difference between sum of odd level and even level nodes
94. System.out.println("Difference between sum of odd level and even level nodes: " + bt.difference());
95. }
96. }

**Output:**

Difference between sum of odd level and even level nodes: 11