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

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.

---

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

Output:

```
90
```

More details.

---

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...");}
```

```
3.
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.  }
4.
5.  class Dog extends Animal{
6.   void eat(){System.out.println("dog is eating...");}
7.
8.   public static void main(String args[]){
9.    Animal a=new Dog();
10.   a.eat();
11. }
12. }
```

More details.

---

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

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.

---

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.

---

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

Output

```
running safely
```

More details.

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.  }
5.
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.

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?

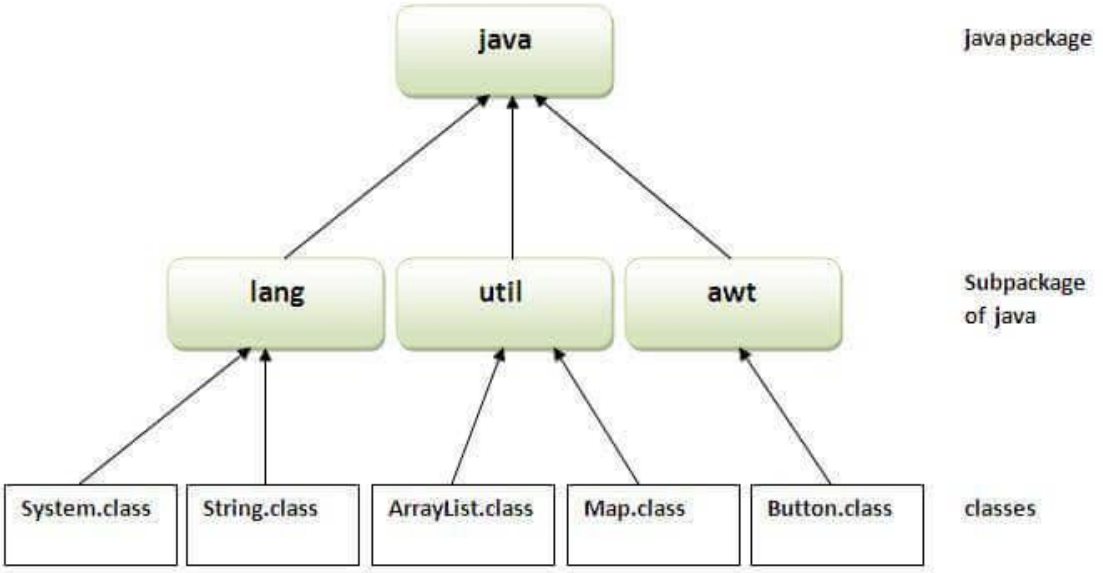
- By providing only the setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
- It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic 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.

---

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.

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.

---

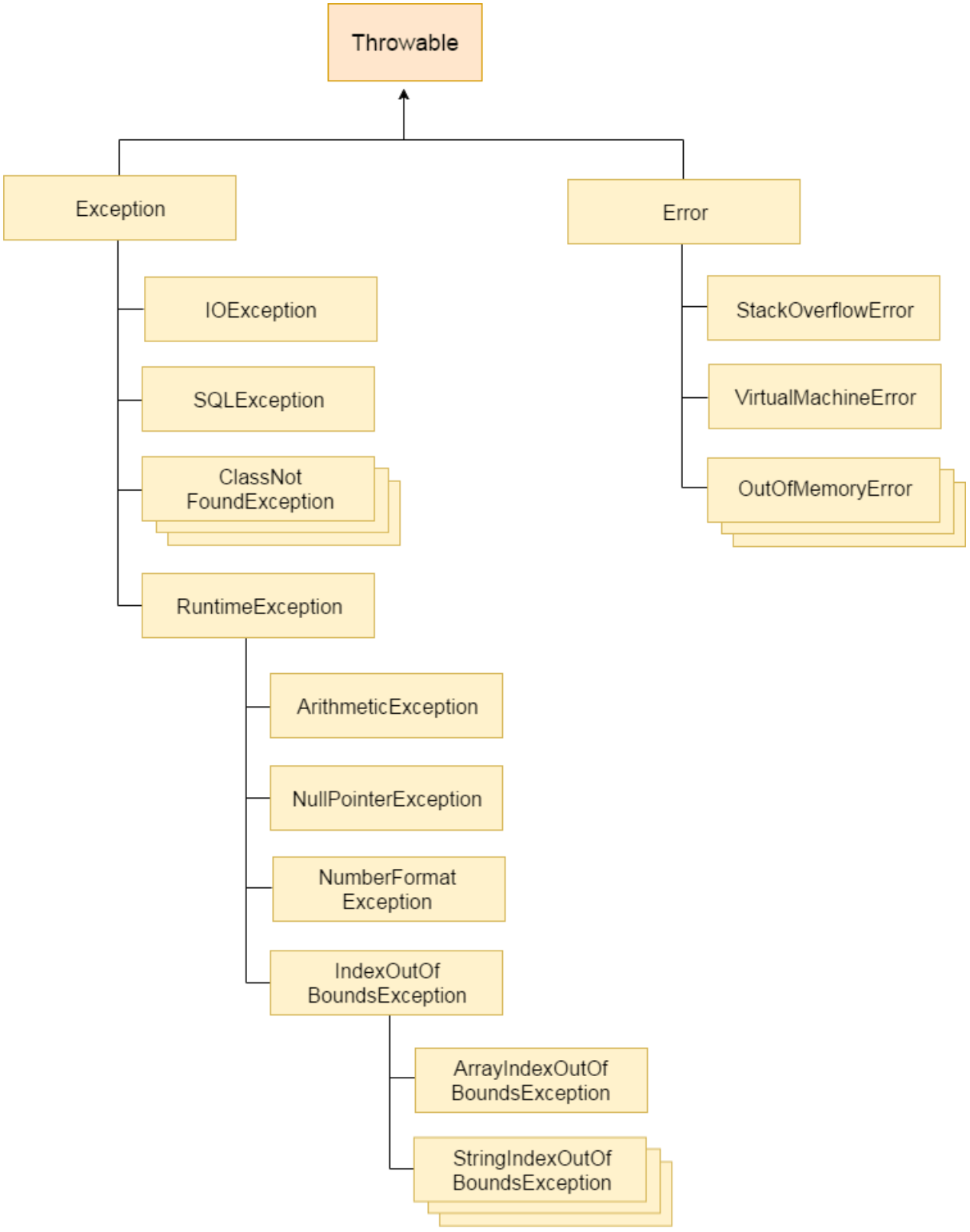
132) What is Exception Handling?

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.

---

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.

---

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. }
13.
```

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
```

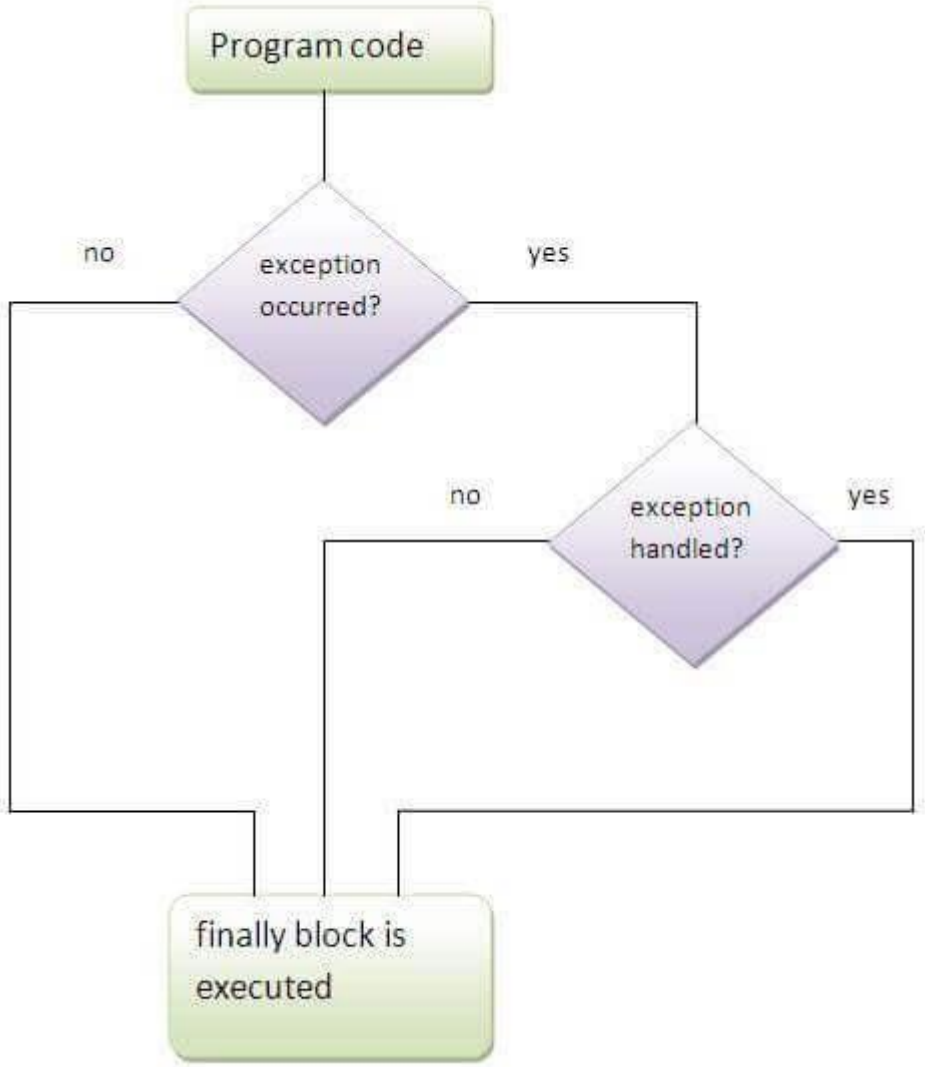
Explanation

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).



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.

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).

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.

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.          }
10.
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.

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.

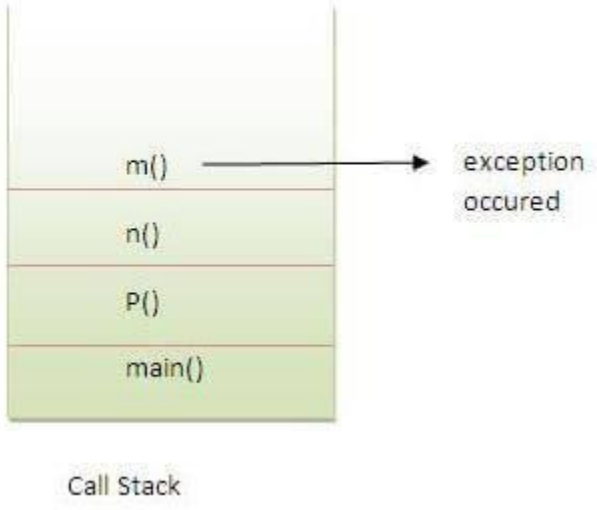
```
1.  class TestExceptionPropagation1{
2.    void m(){
3.      int data=50/0;
4.    }
5.    void n(){
6.      m();
7.    }
8.    void p(){
9.     try{
10.     n();
11.    }catch(Exception e){System.out.println("exception handled");}
12.   }
13.   public static void main(String args[]){
14.    TestExceptionPropagation1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

Test it Now

Output:

```
exception handled
```

```
normal flow...
```



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.    }
30.
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.     }
27.
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.

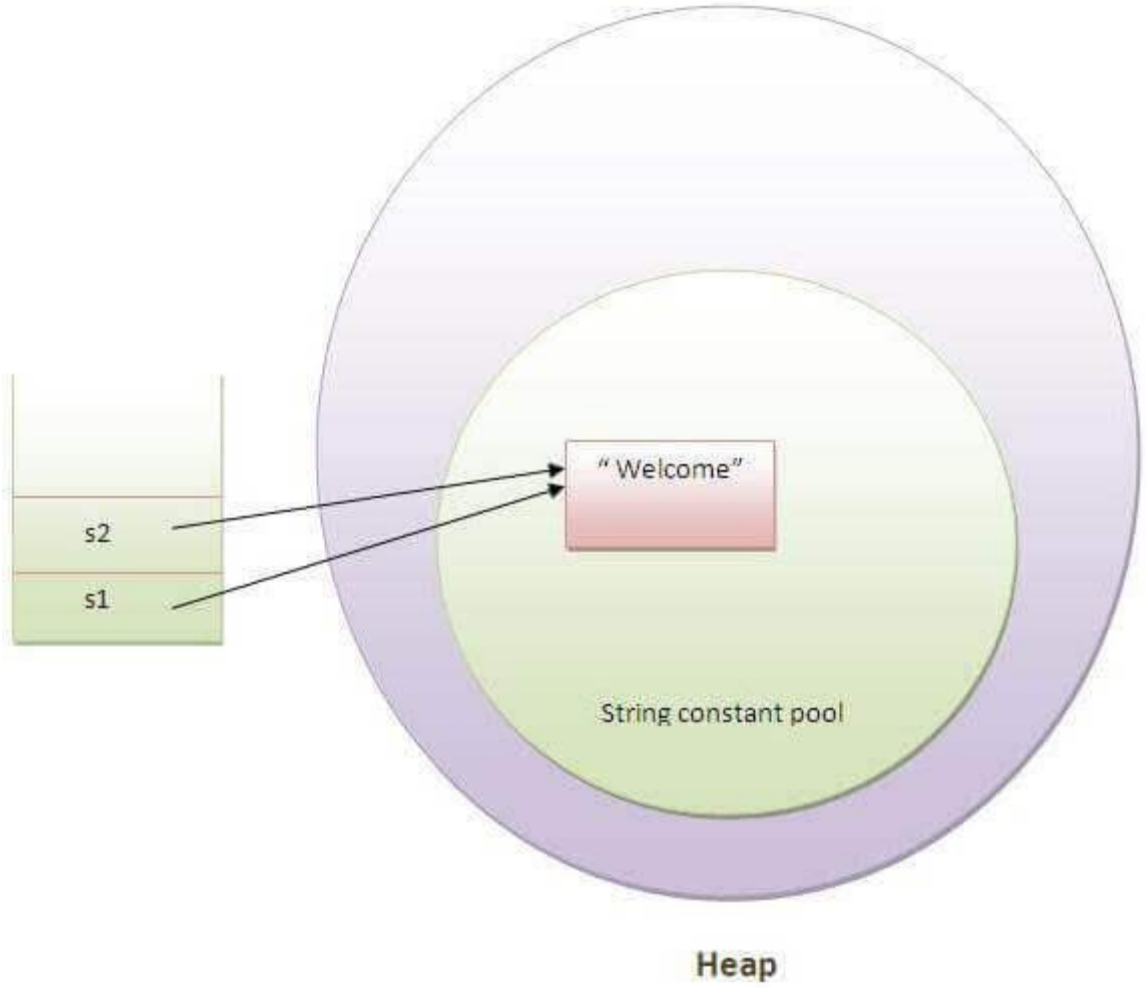
Java: String Handling Interview Questions

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 object
        s
6.    }
7.  }
```

Test it Now

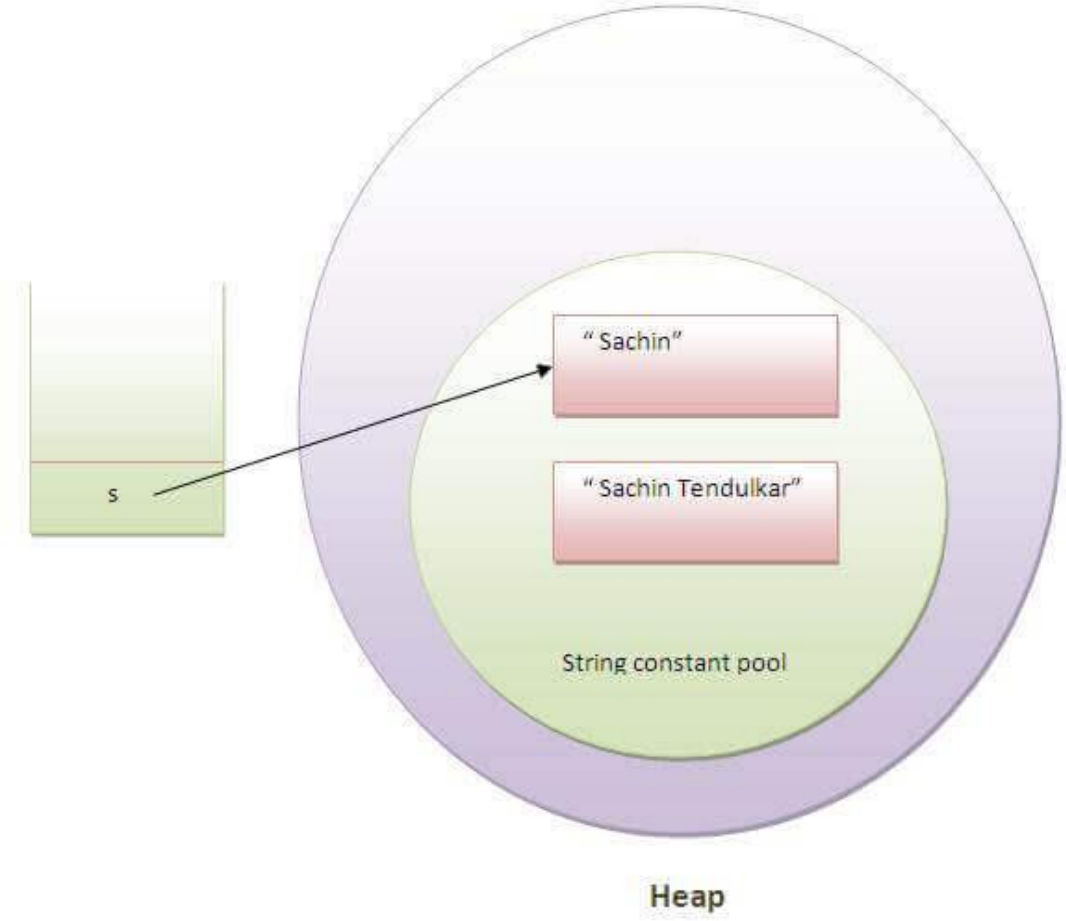
Output:

```
Sachin
```

---

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.



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.

---

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.

---

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.

---

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

```
1. public class Test
```

```
2.
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;
3.
4. public Employee(String pancardNumber){
5. this.pancardNumber=pancardNumber;
6. }
7.
8. public String getPancardNumber(){
9. return pancardNumber;
10. }
11.
12. }
```

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;
5.
6.  Student(int rollno, String name, String city){
7.  this.rollno=rollno;
8.  this.name=name;
9.  this.city=city;
10. }
11.
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");
18.
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
```

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 : "+wo
   rds.length);
8.     }
9. }
```

Output

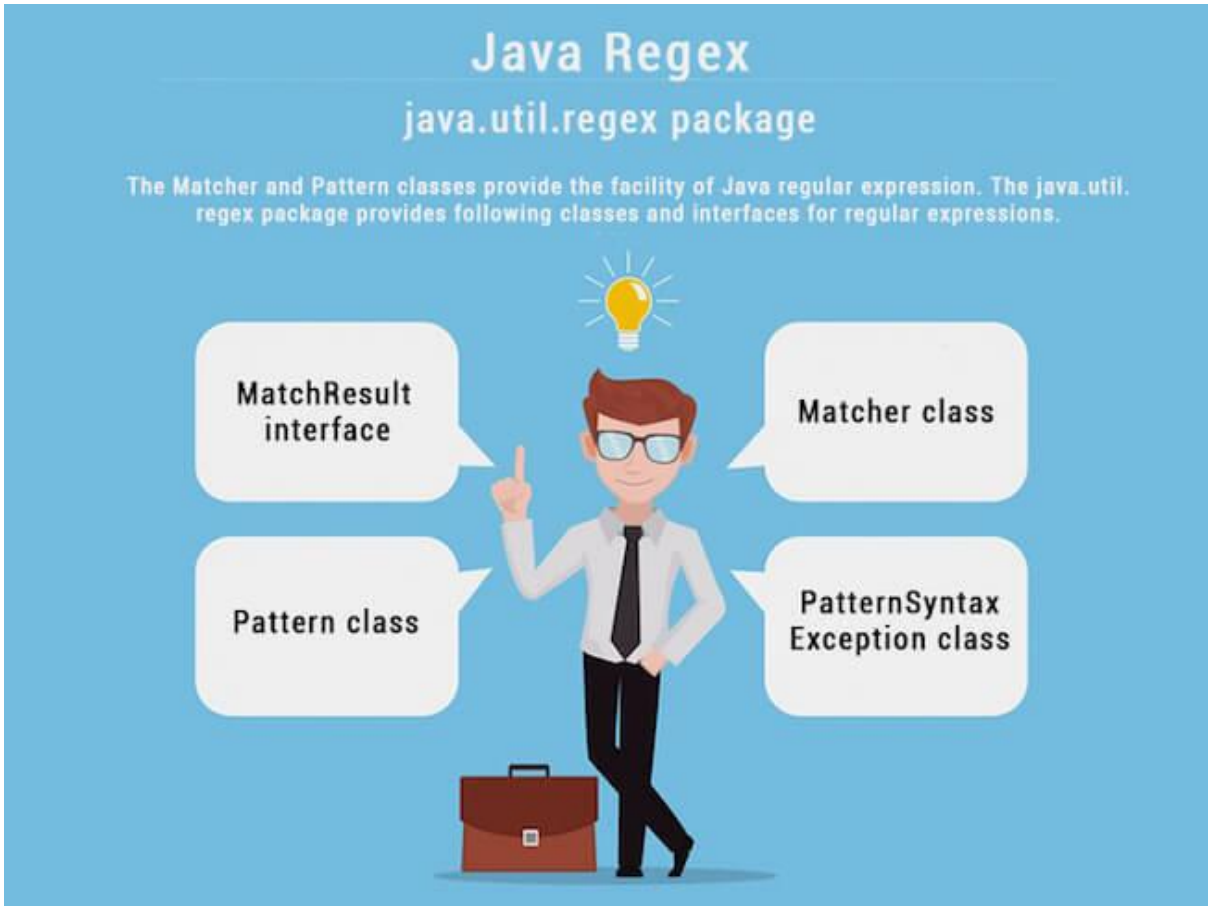
```
The Number of words present in the string are : 10
```

---

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. }
7.
```

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.

---

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	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. Its name is decided by the java compiler.
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.

---

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.

---

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

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

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.  }
7.
```

---

177) Can a class have an interface?

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

178) Can an Interface have a class?

Yes, they are static implicitly.

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.

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

```
        object is garbage collected
        object is garbage collected
```

---

181) How is garbage collection controlled?

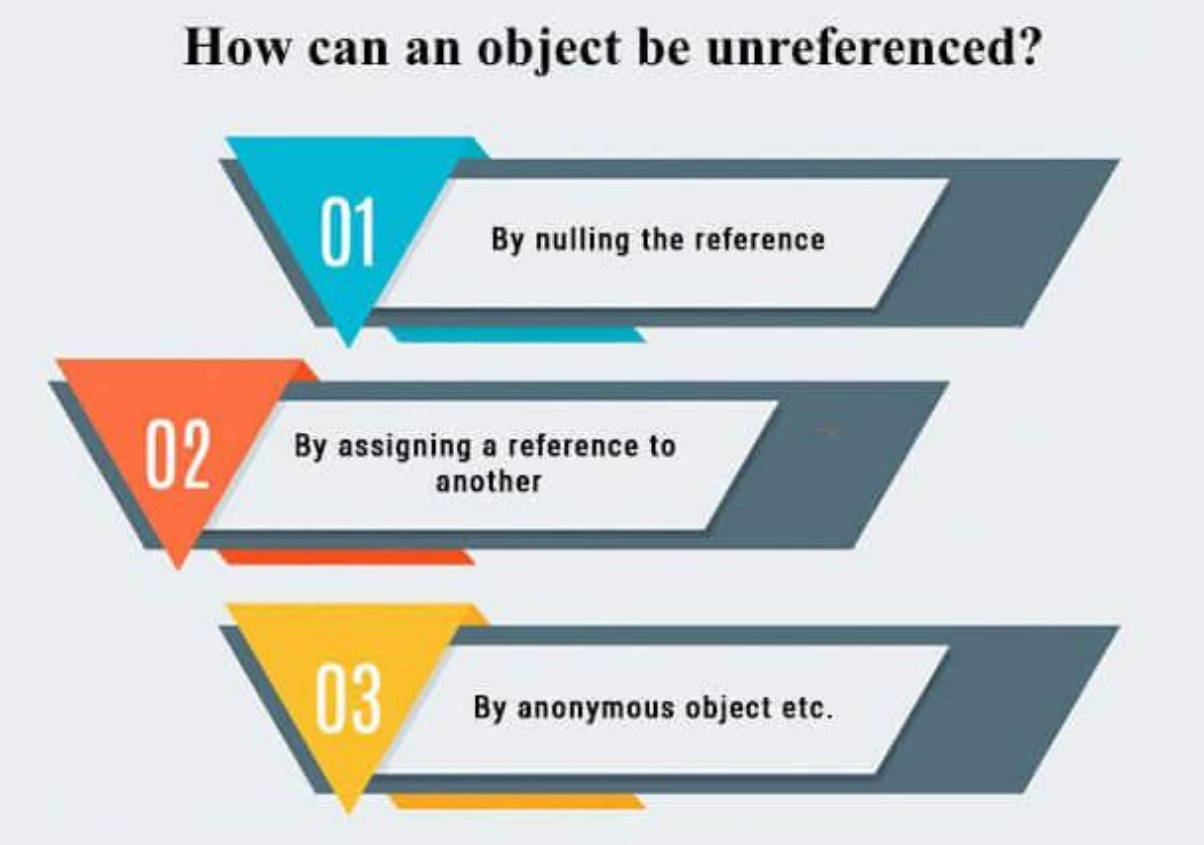
Garbage collection is managed by JVM. It is performed when there is not enough space in the memory and memory is running low. We can externally call the System.gc() for the garbage collection. However, it depends upon the JVM whether to perform it or not.

---

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. }
18.
```

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. }
```