**OOP’S**

**Q-1: Where we will use Abstract class and where Interface.**

**A:** if we want to give some common implementation that will be shared by all sub classes and give some different implementation for all sub class, in this case we will use Abstract class.

If we want to give the complete different implementation for all sub classes according their requirement we will use interface.

**Example of Abstract class**

**Abstract class Vehicle**

public abstract void speed();

public abstract void engine();

public void price()

public void co\_name()

{

S.O.P(“Mahindra”);

}

public void co\_ceo()

{

S.O.P(“Anand Mahindra”);

}

**Class Bikes extends Vehicle Class ForWheeler extends Vehicle**

public abstract void speed()

{

…………….

}

public abstract void engine()

{

…………………..

}

public void price()

{

………………..

}

public abstract void speed()

{

…………….

}

public abstract void engine()

{

…………………..

}

public void price()

{

………………..

}

Class Vehicle Access

{

public static void main(String []args)

{

Vehicle v1 = new Bikes();

v1.speed();

v1.engine();

v1. co\_name();

v1.co\_ceo();

v1.price();

Vehicle v1 = new ForWheeler ();

v1.speed();

v1.engine();

v1. co\_name();

v1.co\_ceo();

v1.price();

}

}

**Example of interface**

**Public class Calculate**

public void area();

public void perimeter();

**Class Circle implements Calculate Class Traingle implements Calculate**

public void area()

{

………………….

}

public void perimeter()

{

………………….

}

public void area()

{

………………….

}

public void perimeter()

{

………………….

}

Class Calculation

Class Demo

{

public static void main(String []args)

{

Calculate c1 = new Circle();

c1.area();

c1.perimeter();

Calculate c2 = new Triangle();

C2.area();

C2.perimeter();

}

}

**Q-2: Can Abstract class contain Constructor.**

**A:** yes

**Q-3: Can Abstract class have static method.**

**A:**No

**Q-1\*:** **How u will achavie abstraction?**

**Q-2: \*diff b/w interface and abstract class**

**A:**

(1)Abstract class can extend only one class

Interface can extend any number of interface at a time

(2)A class can extend only one abstract class.

A class can implement any number of interface.

(3)Abstract class can have both abstract and concrete method

Interface can have only abstracrt method

(4)Abstract class can have protected,public ,public abstract method

Interface can have only public abstract method(by default)

(5)Abstract class have static,final or static final variables with any access specifier

Interface can have only static final(constant) variable i,e by default

(6)Abstract class can have constructor

Interface can not have constructor

(7)Abstract class can have static methods.

Interface can not have static methods.

**Q-4: What is immutable class.How to make immutable class in java.**

**A:**immutable class is a class which once created, it’s contents can not be changed. These are the following ways to create immutable class

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables
4. Make all **mutable fields final** so that it’s value can be assigned only once.
5. Initialize all the fields via a constructor .

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members

Ex:

**public** **final** **class** Employee

{

**private final** String panNumber;

**public** Employee(String panNumber)

{

**this**.panNumber=panNumber;

}

**public** String getPanNumber()

{

**return** panNumber;

}

}

Note: The above class is immutable because:

1-The instance variable of the class is final i.e. we cannot change the value of it after creating an object.

2-The class is final so we cannot create the subclass.

3-There is no setter methods i.e. we have no option to change the value of the instance variable.

**Q-> Can you initialize the final variable later.**

**A:** yes . but by the constructor not by the setter method.

public class Demo {

//final int a=5;

private final int age;

private final String name;

Demo(int age,String name)

{

this.age=age;

this.name=name;

}

void set(int age,String name) // error

{

this.age=age;

this.name=name;

}

public static void main(String[] args) {

Demo d1 = new Demo(11,"subhag");

}

}

**Q-5:What is immutable object .**

**A:** Immutable objects are simply objects whose state (the object's data) cannot change after construction.Ex of immutable objects are String ,Integer.

Immutable Objects

are simple to constrcut,test and use

are automatically thread-safe and have no synchronization issues

don't need a copy constructor

don't need an implementation of clone

**Q-6: Is System class is final class.**

**A:** yes

**Q-7: What is Run time polymorphism? What is the use of this?**

**A: Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than 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.

**Q-8: What we use to clean the Resource.**

**A:** finally

**Q-9: How many ways of Object creation**

**A:** There are 5 ways of Object creation

1-Using New keyword

2-Using New Instance

3-Using Clone

4-Using Deserilization

5-Using ClassLoader

**1-Using New keyword->**Using new keyword is the most basic way to create an object. Use **new**keyword to create and Object of class.  
JBTClass obj = new JBTClass();

**2-Using New Instance->**Have you ever tried to connect to any DB using JDBC driver in Java, If your answer is yes then you must have seen “**Class.forName**“. We can also use it to create the object of a class. **Class.forName**actually loads the class in Java but doesn’t create any Object. To Create an Object of the Class you have to use **newInstance**method of Class class.

Class cls = Class.forName("JBTClass");

JBTClass obj = (JBTClass) cls.newInstance();

JBTClass obj1 = (JBTClass) cls.newInstance();

**Note\*:** If you want to create Object in this way class needs to have public default Constructor.

**3-Using Clone->**We can also use Clone() method to create a copy of an  existing

Object.  
JBTClass obj1 = new JBTClass();

JBTClass obj2 =  (JBTClass)obj1.clone();

**4-Using Deserilization->**Object deserialization can also be used to create an Object. It is just the opposite of serializing an Object.  
ObjectInputStream inStream = new ObjectInputStream(anInputStream );   
MyObject object = (MyObject) inStream.readObject();

**5-Using ClassLoader->**We can also use Class Loader to create Object of a Class. This way is some what same as**Class.forName** option

**Q-10: how many methods in Object class.**

**A:**The List of Object API

1. clone() - Creates and returns a copy of this object.
2. equals() - Indicates whether some other object is "equal to" this one.
3. finalize() - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
4. getClass() - Returns the runtime class of an object.
5. hashCode() - Returns a hash code value for the object.
6. notify() - Wakes up a single thread that is waiting on this object's monitor.
7. notifyAll() - Wakes up all threads that are waiting on this object's monitor.
8. toString() - Returns a string representation of the object.
9. wait() - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

**Q-11: what is enum.**

**A:**

**-** An enum is a data type (similar to class) which contains fixed set of constants.

- The enum constants are static and final implicitely.

- It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc.

Note:

1- enum improves type safety

2- enum can be easily used in switch

3- enum can be traversed

4- enum can have fields, constructors and methods

5- enum may implement many interfaces but cannot extend any class because it

internally extends java.lang.Enum class

**Ex:**

class EnumDemo

{

public enum Season { WINTER, SPRING, SUMMER, FALL } ;

public static void main(String[] args)

{

for (Season s : Season.values())

System.out.println(s);

}

}

**Q-12: What is Generic .**

**A:** Generic provide compile-time type safety that allows programmer to catch invalid types at compile times.java generic methods and generic classes enable programmer to specify –“with a single method decelaration” or “a set of related method” or ”with a single class decelaration” ,a set of related types.

Before generic,we can store any type of objects in collection i.e non-generic.now generic forces the java programmer to store specific type of object.

**Advantage of java Generic->**

There are mainly 3 advantages of generic.

**1-Type Safety->** we can hold only a single type of object in generic.it does not allow to store other type of object.

**2-Type Casting is not required->**there is no need to typecast the object

Ex->

Before Generics,we need to be type cast.

List list = new ArrayList();

list.add(“hello”);

String s = (String)list.get(0); //type casting

After Generic,we do not need to typecast the object.

List<String> list = new ArrayList<String>();

list.add(“hello”);

String s = list.get(0); //type casting

**3-compile-time checking->**it is checked at compile time.so problem will not occur at run time.

List<String> list = new ArrayList<String>();

list.add(“hello”);

list.add(32); // compile time error

**Syntax of use Generic Collection->**

Class or interface <Type>

**Generic Class->** A class that can refer to any type is known as generic.Here we are using T type parameter to create the generic class of specific type.

Ex:

public class Box<T>

{

private T t;

public void add(T t)

{

this.t = t;

}

public T get()

{

return t;

}

Public static void main(String []args)

{

Box<Integer> b1 = new Box<Integer>();

Box<String> b2= new Box<String>();

b1.add(10);

b2.add(“Ram”);

S.O.P(“Integer value is ”+ b1.get());

S.O.P(“String value is ”+ b2.get());

}

}

**Generic Method->**Like generic class,we can create generic method that can accept any type of argument. In below example we use E to denote the element.

public class Demo

{

  public static < E > void printArray(E[] elements)

 {

        for ( E element : elements){

            System.out.println(element );

  }

}

    public static void main( String args[] ) {

        Integer[] intArray = { 10, 20, 30, 40, 50 };

        Character[] charArray = { 'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };

        System.out.println( "Printing Integer Array" );

        printArray( intArray  );

       System.out.println( "Printing Character Array" );

        printArray( charArray );

    }

}

**Q-13: What is the Responsibility of JVM.**

**A:** JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is plateform dependent)

**The JVM performs following operation:**

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

**JVM provides definitions for the:**

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

**Q-14: What is marker interface.name of any 5 marker Interface? What is the use**

**of marker interface.**

**A:** Marker Interface in java is an interface with no fields or methods within it.  An interface with no methods. And no variable that means empty body. marker interface is used as a tag to inform a message to the Java compiler so that it can add special behavior to the class implementing it.

In java we have the following major marker interfaces as under:

1-Searilizable interface

2-Cloneable interface

3-Remote interface

4-ThreadSafe interface

**Use of Marker Interface->**

if the JVM sees a Class is implementing the Serializable interface it does some special operation on it and writes the state of the object into object stream. This object stream is then available to be read by another JVM. Similarly if JVM finds that a class is implementing Clonnable interface, it performs some special operation in order to support cloning. The same theory goes for RMI and Remote interface.

**Q-15: What is the difference b/w JAR and WAR.**

A: These files are simply zipped files using the java jar tool. These files are created for different purposes. Here is the description of these files:

**Java Archives (JAR)files:** The .jar files contain libraries, java classes resources and accessories files like property files. This can be added in classpath for compilation and to run java program. Generally in web applications we keep these files in lib directory of the application

**ojdbc14.jar** – This contains all the classes to connect the oracle database

**Servlet-api.jar** – contains servlet related classes

**Web Archives (WAR) files:** The war file contains the web application that can be deployed on any servlet/jsp container. The .war file contains jsp, html, javascript and other files necessary for the development of web applications. In this context, a Web application is defined as a single group of files, classes, resources, .jar files that can be packaged .

**Q-15: What is JVM Architecture**

**A:**

|  |
| --- |
|  |
|  |
|  |



### 1-Classloader:->

Classloader is a subsystem of JVM that is used to load class files.

### 2-> Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3-> Heap:

It is the runtime data area in which objects are allocated.

### 4-> Stack:

|  |
| --- |
| Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return. |
| Each thread has a private JVM stack, created at the same time as thread. |
| A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes. |

### 5->Program Counter Register:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

### 6-> Native Method Stack:

It contains all the native methods used in the application.

### 7-> Execution Engine:

|  |
| --- |
|  |
| **1) A virtual processor** |
| **2) Interpreter:**Read bytecode stream then execute the instructions. |
| **3) Just-In-Time(JIT) compiler:**It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU. |

**Q-20: What is the super class of primitive data type**

**A:** Number

**Q-30: What will happen in this case.**

**A is super class and B is sub class**

B b1 = new A(); // error

B b1 = (B)new A(); **//** exception

**Q-31: What is Pass by value and Pass by reference. And java support which.**

**A:**

**1- Pass by Value**: The method parameter values are copied to another variable and then the copied object is passed, that’s why it’s called pass by value.

**2-Pass by Reference**: An alias or reference to the actual parameter is passed to the method, that’s why it’s called pass by reference.

Java is always support Pass by Value and not pass by reference,

**Q-18: What is the class loading life cycle.**

**A:** first we write the java source code after that complile the source code after that at Run time following task is happen

|  |
| --- |
|  |
| what happens at runtime when simple java program runs |

|  |
| --- |
| **Classloader:**is the subsystem of JVM that is used to load class files. |
| **Bytecode Verifier:**checks the code fragments for illegal code that can violate access right to objects. |
| **Interpreter:**read bytecode stream then execute the instructions. |

# Q-19: What is class loader

**A:** it is the subsystem of JVM that is used to load class files. There are three type of Loader

**1-Bootstrap Class Loader**->  
Bootstrap class loader loads java’s core classes like java.lang, java.util etc. These are classes that are part of java runtime environment. Bootstrap class loader is native implementation and so they may differ across different JVMs.

**2-Extensions Class Loader->**  
JAVA\_HOME/jre/lib/ext contains jar packages that are extensions of standard core java classes. Extensions class loader loads classes from this ext folder. Using the system environment propery java.ext.dirs you can add ‘ext’ folders and jar files to be loaded using extensions class loader.

**3-System Class Loader->**  
Java classes that are available in the java classpath are loaded using System class loader.

**Q-16: What is Inner class. How many types of Inner class.**

**A: Java inner class** or nested class is a class i.e. declared inside the class or interface

### Diff b/w nested class and inner class in Java-> Inner class is a part of nested class. Non-static nested classes are known as inner classes.

**Type of Nested Class->**

**Nested Class**

**----------------------------------------**

**Non-static Nested class static Nested class**

**--------------------------------------------**

**Member Anonymous Local**

**Inner class Inner class Inner class**

**Type and Description ->**

**Member Inner Class 🡪** A class created within class and outside method.

**Anonymous Inner Class🡪**A class created for implementing interface or extending

class. Its name is decided by the java compiler.

**Local Inner Class 🡪** A class created within method.

**Static Nested Class 🡪** A static class created within class.

**Nested Interface 🡪** An interface created within class or interface.

**Q-17: What is Anonymous Inner class. What is the syntax.**

**What is the use of Anonymous Inner class.**

**A:** A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

**Ex: public class AnonymousClassDemo**

{

public static void main(String[] args)

{

Dog dog = new Dog() {

public void someDog ()

{

System.out.println("Anonymous Dog");

}

}; **// anonymous class body closes here**

**//dog contains an object of anonymous subclass of Dog.**

dog.someDog();

}

}

**class Dog**

{

public void someDog()

{

System.out.println("Classic Dog");

}

}

**Internal working of given code->**

1-Dog dog = new Dog() {, there is a brace at the end of line, not a semicolon. This curly brace opens the class definition and declares a new class that has no name (anonymous class).

**Q-24: What are the rules of Method Overriding.**

**A:** There are some rules of Method Overriding

**1-**Subclass method name must be same as super class method name.

**2-**Subclass method parameter(type, order and number) must be same as super class method parameters.

**3-**Subclass method return type must be same as super class method return type.

**Note->**if the super class method return type is class type then while overriding the method in sub class you can use same class type or its sub class as return type **(From Java 5)**

**EX:**

class Hello class A{}

{

A m1(){…….} class B extends A{}

}

class Hai extends Hello class C{}

{

A m1(){…..}

Or

B m1(){…..} //valid from java5

C m1(){…..} // invalid

}

**4-**Subclass method access modifier must be same or higher then super class method access specifier

|  |  |
| --- | --- |
| **Super Class** | **Sub Class** |
| public | public |
| protected | protected , public |
| default | default , protected , public |
| private | private , default , protected , public |

**5-** when super class method is static method then you have to override in sub class as static only.

**Q-25: What is the Co-variant Return type.**

**A:** Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type should be same class type either subclass type.

Ex:

**class** A

{

A get()

{

**return** **this**;

}

}

**class** B1 **extends** A

{

B1 get()

{

**return** **this**;

}

**void** message()

{

System.out.println("welcome to covariant return type");

}

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

{

B1 b1 = new B1();

System.out.println(b1.get());

B1.message();

}

}

**Q-27: What are the rules of exception while overriding.**

**A:** These are the some rules of method overriding

**1-**When super class method is not specified with method level exception then sub class method can do the following:

a -> Subclass method may not throw any method level exception. **Case-1**

b -> Subclass method can throw unchecked exception. **Case-1**

c-> Subclass method cannot throw checked exception. **Case-2**

**2-** When super class method is specified with method level checked exception then sub class method can do the following:

a -> Subclass method can ignore that method level exception **Case-3**

b -> Subclass method can throw the same exception **Case-4**

c -> Subclass method can throw any unchecked exception **Case-5**

d -> Subclass method can throw exception which is sub class to super class

method exception **Case-6**

e -> Subclass method can not throw exception which is super class to super

class method exception **Case-7**

f -> Subclass method can not throw exception which is non super class to

super class method exception **Case-8**

**3-** When super class method is specified with method level unchecked exception then sub class method can do the following:

a -> Subclass method can ignore that method level exception **Case-9**

b -> Subclass method can throw the same exception **Case-10**

c -> Subclass method can throw any other unchecked exception **Case-11**

d -> Subclass method cannot throw any checked exception **Case-12,13**

**EX: Some Case of Exception**

**Case-1:**

class Hello

{

void show(){}

void display(){}

}

class Hai extends Hello

{

void show(){}

void display()**throws** NullPointerException{}

}

**Case-2:**

class Hello

{

void show(){}

}

class Hai extends Hello

{

void show()**throws** ClassNotFoundException{} //error

}

**Case-3:**

import java.io.IOException;

class Hello

{

void show()**throws** IOException{}

}

class Hai extends Hello

{

void show(){}

}

**Case-4:**

import java.io.IOException;

class Hello

{

void show()**throws** IOException{}

}

class Hai extends Hello

{

void show()**throws** IOException{}

}

**Case-5:**

import java.io.IOException;

class Hello

{

void show()**throws** IOException{}

}

class Hai extends Hello

{

void show()**throws** NullPointerException{}

}

**Case-6:**

import java.io.FileNotFoundException;

import java.io.IOException;

class Hello

{

void show()**throws** IOException{}

}

class Hai extends Hello

{

void show()**throws** FileNotFoundException{}

}

Note: FileNotFoundException is the sub class of IOException that why it is not showing error

**Case-7:**

**import** java.io.\*;

class Hello

{

void show() **throws** IOException{}

}

class Hai extends Hello

{

void show()**throws** Exception{} //error

}

**Case-8:**

import java.io.\*;

class Hello

{

void show() **throws** IOException{}

}

class Hai extends Hello

{

void show()**throws** ClassNotFoundException{} //error

}

**Case-9**

class Hello

{

void show()**throws** ArrayIndexOutOfBoundsException{}

}

class Hai extends Hello

{

void show(){}

}

**Case-10:**

class Hello

{

void show()**throws** ArrayIndexOutOfBoundsException{}

}

class Hai extends Hello

{

void show()**throws** ArrayIndexOutOfBoundsException{}

}

**Case-11:**

class Hello

{

void show()**throws** ArrayIndexOutOfBoundsException{}

}

class Hai extends Hello

{

void show()**throws** RuntimeException{}

}

**Case-12:**

class Hello

{

void show()**throws** ArrayIndexOutOfBoundsException{}

}

class Hai extends Hello

{

void show()**throws** Exception{} // error

}

**Case-13:**

class Hello

{

void show()**throws** ArrayIndexOutOfBoundsException{}

}

class Hai extends Hello

{

void show()**throws** ClassNotFoundException{} //error

}

**Q-28: What is cloning. How many types of cloning. ()**

**A: T**he **object cloning** is a way to create exact copy of an object. For this purpose, clone() method of 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. Syntax of the clone() method is as follows:-> **protected** Object clone() **throws** CloneNotSupportedException

### Why use clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing to be performed that is why we use object cloning.

### Advantage of Object cloning

Less processing task.

**Ex:**

class Student **implements** Cloneable

{

int rollno;

String name;

Student(int rollno,String name){

this.rollno=rollno;

this.name=name;

}

public Object clone()**throws** CloneNotSupportedException

{

**return** **super**.clone();

}

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

**try**{

Student s1=**new** Student(101,"amit");

Student s2=(Student)s1.clone();

System.out.println(s1.rollno+" "+s1.name);

System.out.println(s2.rollno+" "+s2.name);

}**catch**(CloneNotSupportedException c){}

}

}

**Type of cloning->** Java supports two type of cloning: - **Deep and shallow cloning.** By default shallow clone is used in Java. Object class has a method clone() which does shallow cloning.

**Shallow copy->** Shallow clone is a copying the reference pointer to the object, which mean the new object is pointing to the same memory reference of the old object. The memory usage is lower.

**Deep copy->** Deep copy is a complete duplicate copy of an object.. A new memory is allocated for the object and contents are copied.

**Q-32: Can you declare variable inside interface.**

**A:** yes you can. But it is by default static and final.

**Q-34: What type of modifier applies on local variable.**

**A:1-** Local variables cannot use any of the [access level](http://javabeginnerstutorial.com/core-java-tutorial/access-modifier-in-java/) since its scope is only inside the method.

**2-Final** is the Only [Non Access Modifier](http://javabeginnerstutorial.com/core-java-tutorial/non-access-modifiers-in-java/" \o "Non Access Modifiers in Java) that can be applied to a local variable.

**3**-Local variables are not assigned a default value, hence they need to be initialized.

**Q-35: What is the use of method overriding.**

**A:** Method overriding is used to provide specific implementation of a method that is already provided by its super class.

**Q-36: What will happen.**

**Class A**

**{**

**Static void m1()**

**{**

**}**

**}**

**Class B extends A**

**{**

**Static void m1()**

**{**

**}**

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

**{**

**A a = new B();**

**a.m1();**

**}**

**}**

**Q-37: What will happen**

**Class A**

**{**

**Static void add()**

**{**

**}**

**}**

**Class B extends A**

**{**

**void add()**

**{**

**}**

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

**{**

**}**

**}**

**Q-38: What will happen**

**Class A**

**{**

**void add()**

**{**

**}**

**}**

**Class B extends A**

**{**

**Static void add()**

**{**

**}**

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

**{**

**}**

**}**

**Q-39: What will happen**

**Class A**

**{**

**protected void add()**

**{**

**System.out. println (“ A ”);**

**}**

**}**

**Class B extends A**

**{**

**void add()**

**{**

**System.out. println (“ B ”);**

**}**

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

**}**

**}**

**Q-40: What will happen**

**Class A**

**{**

**int add(int a,int b)**

**{**

**}**

**}**

**Class B extends A**

**{**

**float add(int a,int b)**

**{**

**}**

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

**{**

**}**

**}**

**Q-41: What will happen**

class A

{

public static void main(String[] args) {

System.*out*.println("welcome");

}

}

public class B1 extends A {

}

**Q🡪 What is the output of the following program.**

interface A {

public final static int MAX = 100;

}

interface B {

public final static int MAX = 100;

}

class C implements A, B {

public static void main(String args[]) {

System.out.println(MAX);

}

}

**The above program is compile or not. If it compiles what is the output.**

**Q🡪 What is the output of the following program**

**class** Demo {

**public** **void** test(**byte** b ) { System.*out*.println("byte value"); }

**public** **void** test(**int** i) {System.*out*.println("int value"); }

**public** **void** test(Integer i) { System.*out*.println("Integer value"); }

**public** **void** test(**int** ...args) { System.*out*.println("var args value"); }

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

Demo d1 = **new** Demo();

d1.test(100);

d1.test(10000);

}

}

**Q-42: What is difference b/w path and class Path**

**A: Path** variable is set for provide path for all java tools like java, javac, javap, javah, jar, appletviewer which are use at the time of java programming like for run java any program we use java tool and for compile java code use javac tool. These all tools are available in bin folder so we set path upto bin folder.

**classpath** variable is set for provide path of all java classes which is used in our application.(WorkSpace)

**Q-43: What will happen**

public class B1 {

public static void main(String[] args) {

Boolean c1 =false;

if(c1=true)

{

System.*out*.println("Hai");

}

else

{

System.*out*.println("Hello");

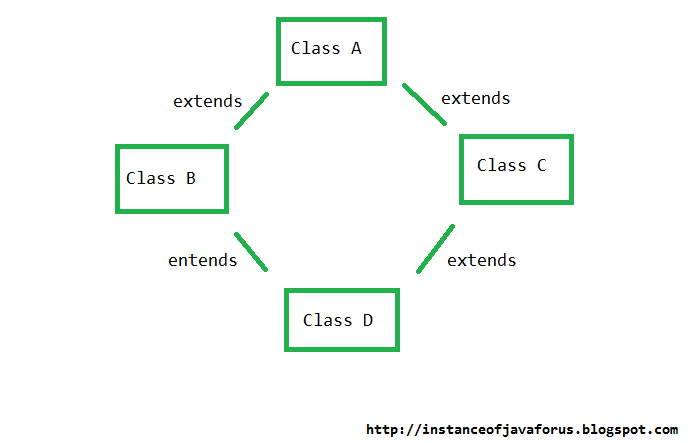
}

}

}

**Q-44: Why java does not support multiple inheritance**

**A:**  1-1) First reason is **ambiguity around Diamond problem**,

[](http://3.bp.blogspot.com/-MtQIpcmOlmE/VJe6XEVaVZI/AAAAAAAAAFo/Plki98loeOI/s1600/mutliple+inheritance+not+suooported.png)  
  
2) Second and more convincing reason to me is that **multiple inheritances does complicate the design and creates problem during casting, constructor chaining etc**

**Q-45: What the new feature come in jdk1.7**

**A:** There are some new feature of jdk1.7

**1.** String in Switch Expression  
**2.** Underscores Between Digits in Numeric Literals  
**3.** Handling multiple exceptions in a single catch block  
**4.** Try-with-resources Statement  
**5.** Automatic Type Inference in Generic object instantiation

**1. String in Switch Expression**

Earlier to **JDK 1.**7, **switch** expression takes**int** values or convertible to **int**. From JDK 1.7, switch accepts string objects also as expression.

|  |  |
| --- | --- |
|  | **public** **class** Demo  {  **public** **static** **void** main(String args[])  {  String str1 = "August";  String str2 = "";    **switch**(str1)  {  **case** "January": str2 = "1st"; **break**;  **case** "February": str2 = "2nd"; **break**;  **case** "March": str2 = "3rd"; **break**;  **case** "April": str2 = "4th"; **break**;  **case** "May": str2 = "5th"; **break**;  **case** "June": str2 = "6th"; **break**;  **case** "July": str2 = "7th"; **break**;  **case** "August": str2 = "8th"; **break**;  **case** "September": str2 = "9th"; **break**;  **case** "October": str2 = "10h"; **break**;  **case** "November": str2 = "11th"; **break**;  **case** "December": str2 = "12th"; **break**;  }  System.out.println(str1 + " is " + str2 + " month in the year");  }  } |

-

**2. Underscores Between Digits in Numeric Literals**

**Underscores** are permitted in numeric literals. You can place underscores where you feel required to increase readability; like between hundreds and thousands, thousands and lakhs etc.

This is used to group numbers in a bigger literal value (especially of long data type).

**Note:**Do not place underscore at the beginning or ending of the literal value.

|  |  |
| --- | --- |
|  | **public** **class** Demo  {  **public** **static** **void** main(String args[])  {  **int** flatCost = 48\_87\_653;    **float** buildingCost = 6\_47\_812.25\_67f;    System.out.println("Flat cost Rs. " + flatCost);  System.out.println("Building cost Rs. " + buildingCost);  }  } |

**3. Handling multiple exceptions in a single catch block**

Before JDK 1.7, it is required to write [multiple catches](http://way2java.com/exceptions/rules-of-exceptions-in-multiple-catch-blocks/) to handle different exceptions raised in the code. Now in a single catch statement, multiple exceptions can be included separated by **pipe ( | )** symbol.

Following is the earlier to **JDK 1.7** code which you are well familiar with.

|  |  |
| --- | --- |
|  | **public** **class** Demo  {  **public** **static** **void** main(String args[])  {  **int** b =0, x[] = { 10, 20, 30 };  **try**  {  **int** c = x[3]/b;  }  **catch**(ArithmeticException e)  {  System.out.println(e);  }  **catch**(ArrayIndexOutOfBoundsException e)  {  System.out.println(e);  }  }  } |

The above multiple catches can be replaced with single catch as follows in JDK 1.7.

|  |
| --- |
| **public** **class** Demo  {  **public** **static** **void** main(String args[])  {  **int** b =0, x[] = { 10, 20, 30 };  **try**  {  **int** c = x[3]/b;  }  **catch**(ArithmeticException | ArrayIndexOutOfBoundsException e)  {  System.out.println(e);  }  }  } |

-

**Note:**Following does not work:

|  |  |
| --- | --- |
|  | **Try**  {  **int** x = 10/0;  }  **catch**(ArithmeticException | RuntimeException e) { } |

Super class exception must be caught separately (it is a constraint).

**4. Try-with-resources Statement**  
(try statement defining resources)

With **JDK 1.7**, no [finally block](http://way2java.com/exceptions/try-%e2%80%93-catch-%e2%80%93-finally/) is required to close (with close() methods) the resources of files or sockets or JDBC handles (objects) etc. The **resources** (say objects) opened in try block automatically close when the execution control passes out try block (say, at the close brace of try block).

**Your Earlier code:**

|  |  |
| --- | --- |
|  | **import** java.io.\*;  **public** **class** Demo  {  **public** **static** **void** main(String args[])  {  FileReader fr = **null**;  FileWriter fw = **null**;  **try**  {  fr = **new** FileReader("abc.txt");  fw = **new** FileWriter("def.txt");    *// some file copying code*  }  **catch**(IOException e)  {  e.printStackTrace();  }  **finally**  {  **try**  {  **if**(fr != **null**) fr.close();  **if**(fw != **null**) fw.close();  }  **catch**(IOException e)  {  e.printStackTrace();  }  }  }  } |

**Your Present code with JDK 1.7:**

|  |  |
| --- | --- |
|  | **import** java.io.\*;  **public** **class** Demo  {  **public** **static** **void** main(String args[])  {  **try** (  FileReader fr = **new** FileReader("abc.txt");  FileWriter fw = **new** FileWriter("def.txt");  )  {  *// some file copying code*    } *// at this point fr and fw are closed*  **catch** (IOException e)  {  e.printStackTrace();  }  }  } |

**5. Automatic Type Inference in Generic object instantiation**

In **JDK 1.7**, empty angle brackets (known as **diamond operator**), **<>**, can be used in specifying generic type instead of writing the exact one. But remember, the compiler should be able to judge the type from the generics statement you write.

Prior JDK 7  
**Set<String> s1 = new LinkedHashSet<String>();**

**List**<**Integer**> primes **=** **new** **ArrayList**<**Integer**>();

In JDK 7  
**Set<String> s1 = new LinkedHashSet<>();**

**List**<**Integer**> primes **=** **new** **ArrayList**<>();

**6. Static blocks**

Earlier to JDK 1.7, to print [static blocks](http://way2java.com/java-introduction/static-blocks-%e2%80%93-static-initialization/) no main() method is required. But from JDK 1.7, if no main() exists, static blocks will not be executed.

# Q-46: What is difference between new operator and newInstance() method

A: Use **new** if you know at compile-time what the type of the object is that you want to create. Use **Class.forName().newInstance()** if you don't know what type of object you'll be making.

# Q-48: What is Auto Boxing and UnBoxing in java. Write the code also.

# A: Auto Boxing->The automatic conversion of primitive data types into its equivalent Wrapper type is known as AutoBoxing.

# UnBoxing->The automatic conversion of Wrapper class type into Corresponding primitive type is known as UnBoxoing

# Ex:

**public** **class** Demo

{

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

{

**int** a1=50;

Integer a2 = **new** Integer(a1); // AutoBoxing

Integer a3 = 60; // AutoBoxing

Integer b1 = **new** Integer(50);

Integer c=10;

**int** b2 = b1; // UnBoxing

**int** b3 = c; // UnBoxing

}

# }

# Q-49: Explain about Garbage Collector.

A: Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

**Advantage of Garbage Collection**

**1-**It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

**2-**It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

3-Every class inherits finalize() method from java.lang.Object, the finalize() method is called by garbage collector when it determines no more references to the object exists. In Java,

4- it is good idea to explicitly assign null into a variable when no more in use. I Java on calling System.gc() and Runtime.gc(), JVM tries to recycle the unused objects, but there is no guarantee when all the objects will garbage collected.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By annonymous 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 annonymous object:

1. **new** Employee();

## 1-finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. **protected** **void** finalize(){}

## 2-System.gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

**Note:** Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

# Q-50: if we will call explicitly finalize() method then what will happen

# A: 1-Object class finalize() method has no implementations

# Protected void finalize(){}

# 2-So you have to override finalize() method finalize() method in your class to release the resource used in you class.

# 3- When you call finalize() explicitly in this case it will be executed just like a normal method call & object would not be destroyed.

# BUT

# If you invoke finalize via System.gc then it will release the resource.

# Ex-1

# class Hello

# {

# void show()

# {

# System.out.println("show()");

# }

# void m1()

# {

# System.out.println("m1-Start()");

# 

# }

# protected void finalize()

# {

# System.out.println("Hello finalize");

# }

# }

# class Demo{

# public static void main(String args[])

# {

# Hello h1 = new Hello();

# h1.show();

# Hello h2 = new Hello();

# h1 = null;

# h2=null;

# System.gc();

# }

# }

# Ex-2:

# public class Demo5

# {

# 

# public void show()

# {

# System.out.println("welcome");

# }

# public void finalize()

# {

# System.out.println("object is garbage collected");

# }

# 

# public static void main(String args[])

# {

# Demo5 s1=new Demo5();

# Demo5 s2=new Demo5();

# s1.finalize(); // it will work as a normal method does not release resource

# s2.show();

# s1=null;

# s2=null;

# System.gc();

# }

# }

# Q-51: What is the purpose Class.forName ()method

# A: A call to Class.forName("X") causes the class named X to be dynamically loaded (at runtime). Class.forName("X") returns the Class object associated with the "X" class.  "X" is the fully qualified name of the desired class.

# Class.forName(String className) method returns the Class object associated with the class or interface with the given string name.

# className -> This is the fully qualified name of the desired class.

# Return Value ->This method returns the Class object for the class with the specified name

# the method forName() is static method of Class,when call it,it can load class from your classpath,

# Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver") // only load class

# you can load many drivers ,but which db u connecting , look follow code

# Ex:

# public class ClassDemo {

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

# try {

# // returns the Class object for the class with the specified name

# Class cls = Class.forName("java.lang.ClassLoader");

# 

# // returns the name and package of the class

# System.out.println("Class found = " + cls.getName());

# System.out.println("Package = " + cls.getPackage());

# }

# catch(ClassNotFoundException ex) {

# System.out.println(ex.toString());

# }

# }

# }

# Q-52: What does Connection pooling mean ?

# A: Opening/Closing database connections is an expensive process and hence connection pools improve the performance of execution of commands on a database for which we maintain connection objects in the pool. It facilitates reuse of the same connection object to serve a number of client requests. Every time a client request is received, the pool is searched for an available connection object and it's highly likely that it gets a free connection object. Otherwise, either the incoming requests are queued or a new connection object is created and added to the pool (depending upon how many connections are already there in the pool). As soon as a request finishes using a connection object, the object is given back to the pool from where it's assigned to one of the queued requests Since most of the requests are served using existing connection objects only so the connection pooling approach brings down the average time required for the users to wait for establishing the connection to the database.

# Use of Connection Pooling->

# It's normally used in a web-based enterprise application where the application server handles the responsibilities of creating connection objects, adding them to the pool, assigning them to the incoming requests, taking the used connection objects back, returning them back to the pool, etc.

# Q-53: What is the diff b/w pooling and caching?

# Q-54: Can a Constructor make final.

# A: No

# Q-55: What is the super class of Wrapper class.

# A: Number

Q-56: **What will happen in this code.**

**Short a=5; short a1=5;**

**a= (a+2);//error a1= (a1+2); //error**

**+2); a= (short)a+2; ; a1= (short)a1+2;**

# Q-57: Can I access the static variable in non static method

# A: No

# Q-58: In java Where compiler use and Where interpreter used.

# A: Javac is Java Compiler 🡪 Compiles your Java code into Bytecode

# JVM is Java Virtual Machine -- >Runs/ Interprets/ translates Bytecode into Native

# Machine Code

# JIT is Just In Time Compiler -- >Compiles the given bytecode instruction sequence to machine code at runtime before executing it natively. It's main purpose is to do heavy optimizations in performance.

# what exactly is the JVM:

# JVM is a virtual platform that resides on your RAM

# Its component, Class loader loads the .class file into the RAM

# The Byte code Verifier component in JVM checks if there are any access restriction violations in your code. (This is one of the principle reasons why java is secure)Next, the Execution Engine component converts the Bytecode into executable machine code

# Java uses a two step compilation process. Java source code is compiled down to "bytecode" by the Java compiler. The bytecode is executed by Java Virtual Machine (JVM). The current version of Sun HotSpot JVM uses a technique called Just-in-time (JIT) compilation to compile the bytecode to the native instructions understood by the CPU on the fly at run time.

# Some implementations of JVM might interpret the bytecode instead of JIT compiling it to machine code and running it directly. While this is still considered an "interpreter."

# Q-59: Give the example of polymorphism in terms of java.

# A: interface . because we create one interface and declare some method inside interface and give the different implementation according to the requirement .So one method deceleration and different implementation.

**Q-60: Why String is immutable and final in java.**

**A:** String is Immutable in Java because String objects are cached in String pool. Since cached String literal is shared between multiple client there is always a risk, where one client's action would affect all other client. For example, if one client changes value of String "Test" to "TEST", all other client will also see that value as explained in first example.

At the same time, ***String was made final***so that no one can compromise invariant of String class .g. Immutability, Caching, hascode calculation etc by extending and overriding behaviors.  
**String immutable Benefits**

1- [String pool](http://www.journaldev.com/797/what-is-java-string-pool) is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space because different String variables can refer to same String variable in the pool.

### 2- If String is not immutable then it would cause severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](http://www.journaldev.com/741/java-socket-server-client-read-write-example) host and port details passed as String. Since String is immutable it’s value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.

3-Since String is immutable, it is safe for multithreading and a single String instance can be shared across different threads. This avoid the usage of synchronization for thread safety, Strings are implicitly thread safe.

4- Since String is immutable, its **hashcode** is cached at the time of creation and it doesn’t need to be calculated again. This is why String is mostly used Object as HashMap keys.

**Q-61: what is aggregation and also explain “Is-A “ and “Has-A” Relationship.**

**A: IS-A Relationship:**

In object oriented programming, the concept of **IS-A** is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of. For example, Apple is a Fruit, Car is a Vehicle etc.

Inheritance is uni-directional. For example House is a Building. But Building is not a House.

Wherever you see an extends keyword or implements keyword in a class declaration, then this class is said to have IS-A relationship.

**HAS-A Relationship**

Has-A means an instance of one class “has a” reference to an instance of another class or another instance of same class.

It is also known as “composition” or “aggregation”.

There is no specific keyword to implement HAS-A relationship but mostly we are depended upon “new” keyword.

**Association**

Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects..

A Student and a Faculty are having an association.

1-A Student has a faculty.

A faculy has multiple student

2-A Student has Courses.

A Course has multiple student.

**Aggregation**

It is a specialized form of association. In aggregation, the part may have an independent lifecycle, it can exist independently. When the whole is destroyed the part may continue to exist.

**Ex:** a car has many parts. A part can be removed from one car and installed into a different car. a car has many parts. A part can be removed from one car and installed into a different car.

**Ex:** A “department” has several “professors”. Without existence of “departments” there is good chance for the “professors” to exist. Hence “professors” and “department” are loosely associated and this loose association is known as Aggregation.

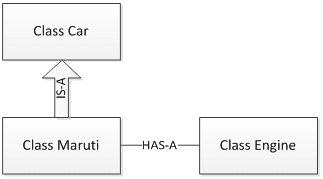
**Composition :**

It is a specialized form of Aggregation. Composition is a stronger form of aggregation. The lifecycle of the part is strongly dependent on the lifecycle of the whole. When the whole is destroyed, the part is destroyed too.

**Ex:** a building has rooms. A room can exist only as part of a building. The room cannot be removed from one building and attached to a different one..

A “university” has several “departments”. Without existence of “university” there is no chance for the “departments” to exist. Hence “university” and “departments” are strongly associated and this strong association is known as composition.

A Library contains students and books. Relationship between library and student is aggregation. Relationship between library and book is composition. A student can exist without a library and therefore it is aggregation. A book cannot exist without a library and therefore its a composition.



**Q-62:How many objects will be created in the following code?**

String s1="Welcome";

String s2="Welcome";

String s3="Welcome";

**A:** Only one object.

**Q-63: How many objects will be created in the following code?**

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

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