1. **Which two method you need to implement for key Object in Hash Map?**

In order to use any object as Key in Hash Map, it must implements equals and hash code method in Java.

1. **What is immutable object? Can you write immutable object?**

String is sequence of characters. In java String object is created from the String class. Immutable simply means unmodifiable or unchanged. Once String object is created we can’t change data or state of that object but we can create new objects. Mostly immutable also final in java which means it prevents sub class from overriding methods and variables of super class

1. **What is the difference between creating String as new () and literal?**

When we create string with new () Operator, it creates its memory in heap but not in the string pool while String created using literal are created in String pool.

String s = new String ("Test");  
   
does not put the object in String pool, we need to call String.intern() method which is used to put  them into String pool explicitly. Where as in the case of string literal the String object automatically created in String pool.

1. **What is**[**difference between StringBuffer and StringBuilder**](http://javahungry.blogspot.com/2013/06/difference-between-string-stringbuilder.html)**in Java ?**

|  |  |
| --- | --- |
| * String buffer methods are synchronized which means two threads can’t call the methods of string Buffer simultaneously. | * While String Builder is non-synchronized. Which means two threads can call the methods of string Buffer simultaneously. |
| * Thread Safe | * Not thread Safe |
| * Less efficient | * More efficient |

1. **Arraylist vs Vector in Java**

|  |  |
| --- | --- |
| Array list is non-synchronized. | Vector is synchronized. |
| Thread Safe. | Not thread Safe. |
| Fast in performance as compared to vector | slow in performance as compared to array list |
| By default Array List size is 10. It checks whether it reaches the last element then it will create the new array, copy the new data of last array to new array, then old array is garbage collected by the Java Virtual Machine (JVM). | **Automatic Increase in Capacity**  When you insert an element into the Array List,**it increases** **its Array size by 50%.** |
| There is no setSize() method or any other method in Array List which can manually set the increment size. | **Set Increment Size**  You can find the following method in Vector Class  public synchronized void setSize(int i) { //some code  } |
| Array List can only use Iterator for traversing an Array List. | Other than Hash table, Vector is the only other class which uses both [Enumeration and Iterator](http://javahungry.blogspot.com/2013/06/difference-between-iterator-and-enumeration-collections-java-interview-question-with-example.html) . |

1. **How do you handle error condition while writing stored procedure or accessing stored procedure from java?**

stored procedure should return error code if some operation fails but if stored procedure itself fail than catching SQLException is only choice.

1. **What is difference between Executor.submit() and Executer.execute() method ?**

If your task throws an exception and if it was submitted with execute this exception will go to the uncaught exception handler which is nothing but whenyou don't provided anything explicitly, the default one will just print the stack trace to System.err).

 If you submitted the task with submit any thrown exception, checked exception or not, is then part of the task's return status.

1. **What is the difference between factory and abstract factory pattern?**

Abstract Factory provides one more level of abstraction. Consider different factories each extended from an Abstract Factory and responsible for creation of different hierarchies of objects based on the type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for creation of objects in that genre.

1. **What is Singleton? is it better to make whole method synchronized or only critical section synchronized ?**

[Singleton in Java is a class with just one instance in whole Java application](http://javahungry.blogspot.com/2013/08/singleton-design-pattern-use-in-java.html), for example java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced it became very easy.

We should **create a static variable**so that it can hold one single instance of our class.  
  
         private static JavaHungrySingleton  uniqueInstance ;

We should **declare the constructor private** so that only Class itself can instantiate the object. Outside  
     classes will not be able to see the class constructor.

**private** **JavaHungrySingleton**(){}

Here in this method implementation we created the class JavaHungrySingleton.  
  
  The return type should be of the class object. But there is possibility that two threads simultaneously try to  
  access the method and we might end up with two different class objects, that’s why method is **synchronized.** In other words, by adding synchronized keyword to the method, we force every thread to wait for its turn before it can enter the method.  So that, two threads do not enter the method at the same time.  
   So we have set up the framework for the singleton method, which up to now looks like

**public** **static** **synchronized** JavaHungrySingleton **getInstance**()

{

**1**. // some code

}

 So to achieve uniqueinstance , first and foremost important thing to check whether the object of the class  
    is already in existence or not . We make sure, by putting condition

**if** (uniqueInstance ==**null** )

{

uniqueInstance=**new** JavaHungrySingleton();

}

**return** uniqueInstance ;

**public** **class** **JavaHungrySingleton**

{

**private** **static** **volatile** JavaHungrySingleton uniqueInstance;

**private** **JavaHungrySingleton**(){}

**public** **static** JavaHungrySingleton **getInstance**()

{

**if** (uniqueInstance ==**null** )

{

**synchronized**(JavaHungrySingleton.class)

{

**if** (uniqueInstance ==**null** )

{

uniqueInstance=**new** JavaHungrySingleton();

}

}

}

**return** uniqueInstance ;

}

// other useful methods here

}

**10. When do you override hashcode and equals ()?**

Especially if you want to do equality check or want to use your object as key in HashMap.

**11. What will be the problem if you don't override hashcode() method ?**

You will not be able to recover your object from hash Map if that is used as key in HashMap.

**12. Does not overriding hash code () method has any performance implication?**  
This is a good question and open to all, as per my knowledge a poor hash code function will result in frequent collision in Hash Map which eventually increase time for adding an object into Hash Map.

**13.  What will happen if you call return statement or System.exit on try or catch block? Will finally block execute?**

Finally block will execute even if you put return statement in try block or catch block but finally block won't run if you call System.exit form try or catch.

**14. Can you override private or static method in Java?**

You cannot override private or static method in Java

**15. What will happen if we put a key object in a Hash Map which is already there?**

Well if you put the same key again than it will replace the old mapping because Hash Map doesn't allow duplicate keys.

**16. If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException?**

 Answer is you can very well throw super class of RuntimeException in overridden method but you cannot do same if it’s checked Exception.

**17. Can you access non static variable in static context?**

No you cannot access static variable in non-static context in Java.

**1. What is this key word in java?**

* this must be used to access instance variable if both instance and local variable names are same.

1. package com.instanceofjava;
3. public class ThisDemo {
4. int a, b;
6. ThisDemo(int a, int b){
8. a=a;
9. b=b;
10. }
12. public static void main(String[] args){
14. ThisDemo obj= new ThisDemo(1,2);
16. System.out.println(obj.a);
17. System.out.println(obj.b);
18. }
19. }

Output:

1. 0
2. 0

Using this keyword:

1. package com.instanceofjava;
3. public class ThisDemo {
4. int a, b;
6. ThisDemo(int a, int b){
8. this.a=a;
9. this.b=b;
10. }
12. public static void main(String[] args){
14. ThisDemo obj= new ThisDemo(1,2);
16. System.out.println(obj.a);
17. System.out.println(obj.b);
18. }
19. }

Output:

1. 1
2. 2

* To call one constructor from another we need this(); and this(); call should be first statement of the constructor.

1. package com.instanceofjava;
3. public class ThisDemo {
4. int a, b;
6. ThisDemo(){
7. System.out.println("Default constructor called");
8. }
9. ThisDemo(int a, int b){
10. this();
11. this.a=a;
12. this.b=b;
13. }
15. public static void main(String[] args){
17. ThisDemo obj= new ThisDemo(1,2);
19. System.out.println(obj.a);
20. System.out.println(obj.b);
21. }
22. }

Output:

1. Default constructor called
2. 1
3. 2

**2. Can we call methods using this keyword?**Yes we can use this keyword to call current class non static methods.

1. package com.instanceofjava;
3. public class Test{
4. int a, b;
6. Test(int a, int b){
8. this.a=a;
9. this.b=b;
10. }
12. void show(){
14. System.out.println("Show() method called");
16. }
18. void print(){
20. this.show();
21. System.out.println(a);
22. System.out.println(b);
24. }
25. public static void main(String[] args){
27. Test obj= new Test(1,2);

30. obj.print()
31. }
32. }

Output:

1. Show() method called
2. 1
3. 2

**3. Can we call method on this keyword from constructor?**



**4. Can we pass this as parameter of method?**Yes we can pass this as parameter in a method

**5. Can we use this to refer static members?**Yes it’s possible to access static variable of a class using this but its discouraged and as per best practices this should be used on non-static reference.

**6 .Is it possible to use this in static blocks?**No it’s not possible to use this keyword in static block.

**7. Can we use this in static methods?**No we cannot use this in static methods. If we try to use compile time error will come. Cannot use this in a static context

8. **What are all the differences between this and super keyword?**

* This refers to current class object whereas super refers to super class object
* Using this we can access all non-static methods and variables. Using super we can access super class variable and methods from sub class.
* Using this(); call we can call other constructor in same class. Using super we can call super class constructor from sub class constructor.
* Constructor chaining example program in same class using this keyword.

1. package instanceofjava;
2. class ConstructorChaining{
3. int a,b
4. ConstructorChaining(){
5. this(1,2);
6. System.out.println("Default constructor");
8. }
10. ConstructorChaining(int x , int y){
12. this(1,2,3);
13. a=x;
14. b=y;
15. System.out.println("Two argument constructor");
17. }
19. ConstructorChaining(int a , int b,int c){
20. System.out.println("Three argument constructor")
21. }
23. public static void main(String[] args){
25. ConstructorChaining obj=new ConstructorChaining();
26. System.out.println(obj.a);
27. System.out.println(obj.b);
29. }
30. }

* Output:

1. Three argument constructor
2. Two argument constructor
3. Default argument constructor
4. 1
5. 2

* Constructor chaining example program in same class using this and super keywords.

1. package instanceofjava;
2. class SuperClass{
4. SuperClass(){
5. this(1);
6. System.out.println("Super Class no-argument constructor");
8. }
10. SuperClass(int x ){
12. this(1,"constructor chaining");
13. System.out.println("Super class one -argument constructor(int)");
15. }
17. SuperClass(int x , String str){
18. System.out.println("Super class two-argument constructor(int, String)");
19. }
21. }

1. package instanceofjava;
2. class SubClass extends SuperClass{
3. SubClass(){
4. this(1);
5. System.out.println("Sub Class no-argument constructor");
7. }
9. SubClass(int x ){
11. this("Constructor chaining");
12. System.out.println("Sub class integer argument constructor");
14. }
16. ConstructorChaining(String str){
17. //here by default super() call will be there so it will call super class constructor
18. System.out.println("Sub class String argument constructor");
19. }
21. public static void main(String[] args){
23. SubClass obj=new SubClass();

26. }
27. }

* Output:

1. Super class two-argument constructor(int, String)
2. Super class one -argument constructor(int)
3. Super Class no-argument constructor
4. Sub class String argument constructor
5. Sub class String argument constructor
6. Sub class integer argument constructor

Basically super keyword used to refer super class methods and variables.

1. package com.superkeywordinjava;
2. public Class SuperDemo{
4. int a,b;
6. }
7. package com.superkeywordinjava;
8. public Class Subdemo extends SuperDemo{
9. int a,b;
10. void disply(){
11. super.a=10;
12. super.b=20;
14. System.out.println(a);
15. System.out.println(b);
16. System.out.println(super.a);
17. System.out.println(super.b);
19. }
21. public static void main (String args[]) {
22. Subdemo obj= new Subdemo();
24. obj.a=1;
25. obj.b=2;
27. obj.disply();

30. }
31. }
32. 1
33. 2
34. 10
35. package com.superinterviewprograms;
36. public Class SuperDemo{
38. int a,b;
40. SuperDemo(int x, int y){
41. a=x;
42. b=y
43. System.out.println("Super class constructor called ");
44. }

47. }
48. package com.superinterviewprograms;
50. public Class Subdemo extends SuperDemo{
52. int a,b;
54. SubDemo(int x, int y){
55. super(10,20);
56. a=x;
57. b=y
58. System.out.println("Sub class constructor called ");
59. }
61. public static void main (String args[]) {
62. Subdemo obj= new Subdemo(1,2);
64. }
65. }
66. Super class constructor called
67. Sub class constructor called

* If our class not extending any class Yes still we can use super(); call in our class Because in java every class will extend Object class by default this will be added by JVM.

1. Super class constructor called
2. Sub class constructor called
3. package com.superinterviewprograms;
5. public Class Sample{
7. Sample(){
8. super();
9. System.out.println("Sample class constructor called ");
11. }
13. public static void main (String args[]) {
15. Sample obj= new Sample();
17. }
18. }
19. Sample class constructor called

9. [How to format text using printf() method in java](http://www.instanceofjava.com/2017/03/java-printf-formatting-table-double.html)

* Print() and println() methods are used to print text and object values or format data.
* In order to format text we also have printf() method in java

1. package printfinjava;
2. /\*\*
3. \* How to format text using java printf() method
4. \* @author www.instanceofjava.com
5. \*/
7. public class PrintfMethod {
9. public static void main(String[] args) {
10. String str="java printf double";
12. System.out.println ("String is "+str);
13. System.out.printf ("String is %s", str);
15. }
17. }

Output:

1. String is java printf double
2. String is java printf double
3. package printfinjava;
4. /\*\*
5. \* How to format text using java printf() method
6. \* @author www.instanceofjava.com
7. \*/
9. public class PrintfMethod {
11. public static void main(String[] args) {
13. double value=12.239344;
14. System.out.printf("%.2f", value);
16. }
18. }

Output:

1. 12.24
2. [**Java program to Convert arraylist to array**](http://www.instanceofjava.com/2017/03/convert-arraylist-to-array-in-java-with.html)
3. package arraysinterview;
4. import java.util.ArrayList;
5. import java.util.List;
7. public class ArrayListTOArray {
9. public static void main(String[] args) {
10. List<String> list = new ArrayList<String>();
12. list.add("array");
13. list.add("arraylist");
14. list.add("convertion");
15. list.add("javaprogram");
17. String [] str = list.toArray(new String[list.size()]);
19. for (int i = 0; i < str.length; i++) {
20. System.out.println(str[i]);
21. }
23. }
25. }

ArrayList<Element> arrayList = new ArrayList<Element>(Arrays.asList(array));

Output:

1. array
2. arraylist
3. convertion
4. javaprogram
5. **Java example program to explain about static method in java**

Static methods are class level so there are not part of object. When we override static methods its not overriding it is method hiding and whenever we call method on class name it will call corresponding class method.

1. package inheritanceInterviewPrograms;
2. /\*
3. \* @website: www.instanceofjava.com
4. \* @category: Deference between staic and final static methods in java
5. \*/
6. public class Super {

9. static void method(){
11. System.out.println("Super class method");
12. }
13. }
14. package inheritanceInterviewPrograms;
15. //  www.instanceofjava.com
16. public class Sub extends Super {
18. static void method(){
20. System.out.println("Sub class method");
21. }
23. public static void main (String args[]) {
25. Super.method();
26. Sub.method();

29. }
30. }

Output:

1. Super class method
2. Sub class method
3. **Java example program to explain about calling super class static method using sub class in java**

We can call super class static methods using sub class object or sub class name also.

1. package inheritanceInterviewPrograms;
2. //  www.instanceofjava.com
3. public class Sub extends Super {

6. public static void main (String args[]) {
8. Super.method();
9. Sub.method();

12. }
13. }

Output:

1. Super class method
2. Super class method
3. **Final static methods in java:**

* Can a method be static and final together in java?
* When we declare a method as final we can not override that method in sub class.
* In the same way when we declare a static method as final we can not hide it in sub class means we can not create same method in sub class.
* If we try to create same static method in sub class compiler will throw an error.

**Java example program to explain about final static method in java**

1. package inheritanceInterviewPrograms;
2. /\*
3. \* @website: www.instanceofjava.com
4. \* @category: Deference between staic and final static methods in java
5. \*/
6. public class Super {

9. final static void method(){
11. System.out.println("Super class method");
12. }
13. }
14. package inheritanceInterviewPrograms;
15. //  www.instanceofjava.com
16. public class Sub extends Super {
18. static void method(){  // compiler time error:
20. System.out.println("Sub class method");
22. }
23. public static void main (String args[]) {
25. Super.method();
26. Sub.method();

29. }
30. }
31. **Fanalize()**
32. [**Quicksort algorithm in java with example program**](http://www.instanceofjava.com/2016/08/quicksort-example-in-java-recursion.html)

* Quick sort uses divide and conquer algorithm.
* First will be divided in to two parts based on some pivot element. All elements which are less than pivot element will be placed left and and all elements which are greater than pivot will be in right part.
* So now pivot element is exactly in middle. if we sort left and right side elements all elements will be sorted.

1. [**8 different ways to convert int to String in java**](http://www.instanceofjava.com/2016/08/how-to-convert-integer-to-string-in-java.html)

int n=100;

* String str=n.toString();
* String str=Integer.toString();
* String str=String.valueOf(n);
* String str=Integer(n).toString();
* String str=String.format(“%d”,n);
* DecimalFormat obj=new DecimalFormat(“#”);//DesimalFormat is a class

String str=obj.format(n);

* StringBuffer sf=new StringBuffer();

sf.appen(n);

String str=sf.toString();

1. [**Java interface programming questions**](http://www.instanceofjava.com/2016/07/java-interface-programming-questions.html)

We can develop interfaces by using "interface" keyword.

A class will implements all the methods in an interface.

By default interface methods are abstract.

Interface allows public, static and final modifiers.

Interface doesn’t allow constructor

1. **Polymorphism**

Polymorphism means defining multiple methods with same name.

Two types of polymorphism   
1.Static polymorphism  
2.Dynamic polymorphism.

#### Static polymorphism:

* Defining multiple methods with same name and with different type of arguments is known as static polymorphism.
* We can achieve this static polymorphism using method overloading.

**Static polymorphism in java with example program**

1. package com.instanceofjava;
2. class A{
4. public void show(int a){
5. System.out.println("saidesh");
6. }
8. public void show(int a,int b){
9. System.out.println("ajay");
10. }
12. public void show(float a){
13. System.out.println("vinod");
14. }
15. public static void main(String args[]){
17. A a=new A();
19. a.show(10);
20. a.show(1,2);
21. a.show(1.2);
23. }
24. }

Output:

1. saidesh
2. ajay
3. vinod

**Dynamic polymorphism:**

* Defining multiple methods with same name and with same signature in super class and sub class.
* We can achieve dynamic polymorphism by using method overriding concept in java.
* Define a method in super class and override same method with same signature in sub class
* Whenever we call the method on t the object based on the object corresponding class method will be executed dynamically.

**Java example program on dynamic polymorphism: (Real time example)**

1. package MethodOverridingExamplePrograms;
2. public class Vehicle{
4. void Start(){
5. System.out.println("Vehicle started")
6. }
7. }

1. package MethodOverridingExamplePrograms;
2. public class Car extends Vehicle{
4. void Start(){
5. System.out.println("Car started")
6. }
7. }

1. package MethodOverridingExamplePrograms;
2. public class Bus extends Vehicle{
4. void Start(){
5. System.out.println("Bus started")
6. }
7. }

1. package MethodOverridingExamplePrograms;
2. public class Bike extends Vehicle{
4. void Start(){
5. System.out.println("Bike started")
6. }
7. }

1. package MethodOverridingExamplePrograms;
2. public class Test{
4. public static void startVehicle(Vehicle vh){
6. vh.start();
8. }
9. }
10. [**3 different ways to print exception message in java**](http://www.instanceofjava.com/2016/04/ways-to-find-details-of-exception-java.html)

cache(Exception e)

{

System.out.println(e);

e.printStackTrace();

System.out.println(e.getMessage());//gives only the msg like division by zero

}

1. [**Default constructor vs no arg constructor**](http://www.instanceofjava.com/2016/03/default-constructor-vs-no-argument.html)

* When we write a class without any constructor then at compilation time java compiler creates a default constructor in our class.
* The accessibility modifier of the default constructor is same as accessibility modifier of class.
* The allowed accessibility modifier are public and default.
* Default constructor added by java compiler this constructor does not have anything except super(); call.
* If our class have any constructor then java compiler does not create default constructor

**No-argument Constructor in java:**

* As a developer we can create our own constructor with no arguments is known as no-argument constructor.
* It can have all four accessibility modifiers as it is defined by developer.
* So allowed accessibility modifiers are public, private, protected and default
* It can have logic including super call.
* The common point between default and no-argument constructor
* Both does not have any arguments.
* And one more point we need to remember that in no-argument constructor also by default first statement will be super() call which is added by java compiler if it does not have.

1. [**Can we have try without catch block in java**](http://www.instanceofjava.com/2016/04/can-we-have-try-without-catch-in-java.html)

* It is possible to have try block without catch block by using finally block
* Java supports try with finally block
* As we know finally block will always executes even there is an exception occurred in try block, Except System.exit() it will executes always.
* We can place logic like connections closing or cleaning data  in finally.
* Finally block executes Even though the method have return type and try block returns something

**Java Program to write try without catch block | try with finally block**

1. package exceptionsInterviewQuestions;
2. public class TryWithoutCatch {

5. public static void main(String[] args) {

8. try {
10. System.out.println("inside try block");
12. } finally{
14. System.out.println("inside finally block");
15. }
17. }
19. }

**Output:**

1. inside try block
2. inside finally block

**Java Program to write try with finally block and try block returns some value**

1. package exceptionsInterviewQuestions;
3. public class TryWithFinally {
5. public static int method(){

8. try {
10. System.out.println("inside try block");
12. return 10;
13. } finally{
15. System.out.println("inside finally block");
16. }
18. }
20. public static void main(String[] args) {
22. System.out.println(method());
24. }
26. }

**Output:**

1. inside try block
2. inside finally block
3. 10
4. [**Unreachable Blocks in java**](http://www.instanceofjava.com/2016/04/unreachable-blocks-in-java-example.html)

* The block of statements to which the control would never reach under any case can be called as unreachable blocks.
* Unreachable blocks are not supported by java.
* Thus catch block mentioned with the reference of  "Exception" class should and must be always last catch block. Because Exception is super class of all exceptions.

1. package com.instanceofjava;
2. public class ExcpetionDemo {
3. public static void main(String agrs[])
4. {
6. try
7. {
8. //statements
9. }
10. catch(Exception e)
11. {
12. System.out.println(e);
13. }
14. catch(ArithmeticException e)//unreachable block.. not supported by java. leads to error
15. System.out.println(e);
16. }
17. }
18. **What is an exception?**
19. **What are the differences between exception and error?**
20. **How the exceptions are handled in java**
21. **What is the super class for Exception and Error**

Throwable.

1. public class Exception extends Throwable implements Serializable

1. public class Error extends Throwable implements Serializable
2. **Exceptions are defined in which java package**java.lang.Exception

# [**Difference between throw and throws in java**](http://www.instanceofjava.com/2014/12/difference-between-throw-and-throws-in.html)

* throw keyword used to throw user defined exceptions.(we can throw predefined exception too)
* If we are having our own validations in our code we can use this throw keyword.
* For Ex: BookNotFoundException, InvalidAgeException (user defined)

1. //Custom exception
2. package com.instanceofjavaforus;
3. public class InvalidAgeException extends Exception {
4. InvaidAgeException(String msg){
5. super(msg);
6. }
7. }
8. package com.instanceofjavaforus;
9. public class ThrowDemo {
10. public boolean isValidForVote(int age){
11. try{
12. if(age<18){
13. throw new InvalidAgeException ("Invalid age for voting");
14. }
15. }catch(Exception e){
16. System.out.println(e);
17. }
18. return false;
19. }
20. public static void main(String agrs[]){
21. ThrowDemo obj= new ThrowDemo();
22. obj.isValidForVote(17);
23. }
24. }

We can throw predefined exceptions also

1. package com.instanceofjavaforus;
2. public class ThrowDemo {
3. public void method(){
4. try{
6. throw new NullPointerException("Invalid age for voting");
7. }
8. }catch(Exception e){
9. System.out.println(e);
10. }
11. }
12. public static void main(String agrs[]){
13. ThrowDemo obj= new ThrowDemo();
14. obj.method();
15. }
16. }

Like this we can throw checked exceptions and unchecked exceptions also. But throw keyword is mainly used to throw used defined exceptions / custom exceptions.

**Throws:**

* The functionality of throws keyword is only to explicitly to mention that the method is proven transfer un handled exceptions to the calling place.
* Using throws keyword we can explicitly provide the information about unhand-led exceptions of the method to the end user, Java compiler, JVM.
* Using throws keyword we can avoid try-catch with respect to the statements which are proven to generate checked exceptions.

1. package com.instanceofjavaforus;
2. public class ThrowSDemo {
3. public void method(int a,int b) Throws ArithmeticException{
5. inc c= a/b;
6. }
7. public static void main(String agrs[]){
8. ThrowDemo obj= new ThrowDemo();
9. try{
10. obj.method(1,0);
11. }catch(Exception e){
12. System.out.println(e);
13. }
14. }
15. }

# **Can we write multiple catch blocks under single try block?**

* One try can have multiple catch blocks
* Whenever an exception object is identified in try block and if there are multiple catch blocks then the priority would be always given to first catch block. If the first catch block cannot handle the identified exception object then it considers the immediate next catch block.
* Let’s see a java example programs which can raise multiple exceptions. Where we are handling only arithmetic exception. But there is a chance of getting ArrayIndexOutofBounds exception also. So better to maintain multiple cache blocks in such cases.
* package exceptions;
* public class MultipleCatchBlocks {
* /\*\*
* \* @www.instanceofjava.com
* \*/
* public static void main(String[] args) {

* int x=10;
* int y=0;
* try{
* int i=x/y;
* int a[]=new int[2];
* a[3]=12;
* } catch(ArithmeticException e){
* e.printStackTrace();
* }
* }
* }

**Output:**

* java.lang.ArithmeticException: / by zero  
      at exceptions.MultipleCatchBlocks.main(MultipleCatchBlocks.java:15)

# **How to write user defined exception or custom exception in java**

* Apart from java we have Existing Exceptions. And we can also create our own Exceptions nothing but User defined Exceptions.
* User defined exceptions in java are also known as Custom exceptions.

**Program:**   
InvalidAgeException .java:

1. package com.instanceofjava;
2. class InvalidAgeException extends Exception{
3. InvalidAgeException(String s){
4. super(s);
5. }
6. }

**Program:**   
TestUsrDefinedException,java:

1. package com.instanceofjava;
2. class TestUsrDefinedException{
3. static void validate(int age)throws InvalidAgeException{
4. if(age<18)
5. throw new InvalidAgeException("Invalid age");
6. else
7. System.out.println("welcome to vote");
8. }
9. public static void main(String args[]){
10. try{
11. validate(13);
12. }
13. catch(Exception m){
14. System.out.println("Exception occured: "+m);
15. }
16. finally{
17. System.out.println("This block will be Executed")
18. }
19. }
20. }

# **What are the differences between final finally and finalize in java**

### **Final:**

* Any variable declare along with final modifier then those variables treated as final variable.
* if we declare final variables along with static will became constants.
* public final String name = "foo"; //never change this value
* If you declare method as final that method also known as final methods.Final methods are not overridden.means we can't overridden that method in anyway.
* public final void add(){  
   }  
   public class A{  
   void add(){  
   //Can't override  
   }  
    
   }
* If you declare class is final that class is also known as final classes.Final classes are not extended.means we can't extens that calss in anyway.
* public final class indhu{  
   }  
   public class classNotAllowed extends indhu {...} //not allowed

### **Finally:**

* Finally blocks are followed by try or catch.finally blocks are complasary executable blocks.But finally is useful for more than just exception handling.
* it allows the programmer to avoid having cleanup code accidentally bypassed by a return,  
   continue, or break,Closing streams, network connection, database connection. Putting cleanup  code in a finally block is always a good practice even when no exceptions are anticipated
* where finally doesn't execute e.g. returning value from finally block, calling System.exit from try block etc
* finally block always execute, except in case of JVM dies i.e. calling System.exit()   
   lock.lock();  
  try {  
    //do stuff  
  } catch (SomeException se) {  
    //handle se  
  } finally {  
    lock.unlock(); //always executed, even if Exception or Error or se  
    //here close the database connection and any return statements like that we have to write  
  }

### **Finalize():**

* finalize() is a method which is present in Java.lang.Object class.
* Before an object is garbage collected, the garbage collector calls this finalize() of object.Any unreferncebefore destorying if that object having any connections with databse or anything..It will remove  the connections and it will call finalize() of object.It will destroy the object.
* If you want to Explicitly call this garbage collector you can use System.gc() or Runtime.gc() objects are there from a long time the garbage collector will destroy that objects.
* public void finalize() {  
    //free resources (e.g. unallocate memory)  
    super.finalize();  
  }

# **Can we write return statement in try and catch blocks**

Finally block will executes always excepts system.exit().

* So if we are returning some value in finally means it will be returned always
* Finally will executed so method always returns finally return value and no need of keeping return value at end of the method.
* And in finally after return if we keep some statement those statement will be treated as dead code.

1. package com.exceptionhandlingiinterviewquestions;
3. public class TryCatchReturn{
5. int calc(){
7. try {
9. return 10;
11. } catch (Exception e) {
12. return 20;
13. }
14. finally(){
15. return 30;
16. }
18. }

21. public static void main(String[] args) {
23. TryCatchReturn obj = new TryCatchReturn();
25. System.out.println(obj.calc())
26. }
27. }

### **Output:**

1. 30

# **Is it possible to print message without using system.out.println?**

* System.out.write("www.instanceofjava.com \n".getBytes());
* System.out.format("%s", "www.instanceofjava.com \n")
* PrintStream myout =  new PrintStream(new FileOutputStream(FileDescriptor.out));  
     myout.print("www.instanceofjava.com \n");
* System.err.print("This is custom error message");
* System.console().writer().println("Hai");

# **Is it possible to write multiple exceptions in single catch block**

* It is not possible prior to java 7.
* new feature added in java 7.

1. package exceptionsFreshersandExperienced;
2. public class ExceptionhandlingInterviewQuestions{
4. /\*\*
5. \* @www.instanceofjava.com
6. \*\*/
7. public static void main(String[] args) {
9. try {
11. // your code here
13. } catch (IOException | SQLException ex) {
14. System.out.println(e);
15. }
17. }
19. }

# [**Java program to restrict a class from creating not more than three objects**](http://www.instanceofjava.com/2016/03/restrict-class-creating-multple-objects.html)

1. package com.instanceofjava;
3. public class RestrictObjectCreation{
4. private static RestrictObjectCreationobject;
5. public static int objCount = 0;
7. private RestrictObjectCreation()
8. {
9. System.out.println("Singleton(): Private constructor invoked");
11. objCount  ++;
12. }
14. public static RestrictObjectCreation getInstance()
15. {
17. if (objCount < 3)
18. {
20. object = new RestrictObjectCreation();
22. }
24. return object;
26. }
28. }
29. package instanceofjava;
31. public class Test{
32. public static void main(String args[]) {
34. RestrictObjectCreation obj1= RestrictObjectCreation.getInstance();
35. RestrictObjectCreation obj2= RestrictObjectCreation.getInstance();
36. RestrictObjectCreation obj3= RestrictObjectCreation.getInstance();
37. RestrictObjectCreation obj4= RestrictObjectCreation.getInstance();
38. RestrictObjectCreation obj5= RestrictObjectCreation.getInstance();
40. System.out.println(obj1.hashCode());
41. System.out.println(obj2.hashCode());
42. System.out.println(obj3.hashCode());
43. System.out.println(obj4.hashCode());
44. System.out.println(obj5.hashCode());
46. }
47. }

**Output:**

1. RestrictObjectCreation(): Private constructor invoked
2. RestrictObjectCreation(): Private constructor invoked
3. RestrictObjectCreation(): Private constructor invoked
4. 705927765
5. 366712642
6. 1829164700
7. 1829164700
8. 1829164700

# [**Abstract class and abstract method interview questions and programs**](http://www.instanceofjava.com/2016/02/abstract-method-class-example-programs.html)

1. **What is abstract class in java?**

* Hiding the implementation and showing the function definition to the user.
* Abstract class contains abstract methods and concrete methods(normal methods)

1. **How can we define an abstract class?**

* Using abstract keyword we can define abstract class.
* Check below code for abstract class example program.

1. package Abstraction;
2. public **abstract**class AbstractDemo {
3. //
4. //
5. }
6. **How to declare an abstract method?**

* By using abstract keyword in the method signature we can declare abstract method.

1. package Abstraction;
2. public **abstract**class AbstractDemo {
3. //
4. **abstract**void show();
5. }

1. **Can we define abstract class without abstract method?**

* Yes we can define abstract class without abstract methods.
* It is not mandatory to create at-least one abstract method in abstract class.

1. package Abstraction;
2. public **abstract**class AbstractDemo {
3. //
5. }
6. **Can we create object for abstract class?**

* Abstract class cannot be instantiated directly.
* Means we cannot create object for abstract class directly.
* Through sub class abstract class members will get memory. Whenever we create sub class object of abstract class abstract class object will be created. i.e abstract class members will get memory at that time.

1. package Abstraction;
2. public abstract class AbstractClass {
4. abstract void add(); // abstract method
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
10. public static void main(String[] args){
12. AbstractClass obj= new AbstractClass ();
13. // ERROR: Cannot instantiate the type AbstractClass
15. }
16. }
17. **What are the valid and invalid keywords or modifier with abstract class?**

* Public, protected and default are valid.
* Static, final and private are invalid.

1. **What happens if sub class not overriding abstract methods?**

* If sub class which is extending abstract class not overriding abstract method compiler will throw an error.

1. package Abstraction;
2. public abstract class AbstractClass {
4. abstract void add() throws Exception;
6. }
7. package Abstraction;
8. public class Sample extends AbstractClass {
10. //Compiler Error: The type Sub must implement the inherited abstract method Super.add()
11. }
12. **Can we escape of overriding abstract class in sub class which is extending abstract class?**

* Yes we can escape from overriding abstract method from super abstract class by making our class again as abstract.
* The class which is extending our class will get responsibility of overriding all abstract methods in our class and in super class.

1. package Abstraction;
2. public abstract class AbstractClass {
4. abstract void add() throws Exception;
6. }
7. package Abstraction;
8. public abstract class Sample extends AbstractClass {

11. }
12. package Abstraction;
13. public class Example extends Sample{
15. //Compiler Error: The type Sub must implement the inherited abstract method Super.add()
16. }

# [**Constructor in interface?**](http://www.instanceofjava.com/2016/02/java-interface-constructor-example.html)

# **Can we write constructor inside interface in java?**

* No. Interfaces does not allow constructors.
* Why interface does not have constructor? The variables inside interfaces are static final variables means constants and we cannot create object for interface so there is no need of constructor in interface that is the reason interface doesn't allow us to create constructor.

# **What will happens if we try to create constructor inside interfaces in java**

* If we try to create constructor in interface compile time error will come.
* Error description: Interfaces cannot have constructors.

1. public interface sample{
3. int a=10;
4. sample(){//Interfaces cannot have constructors.
5. }
7. }

**Interfaces in Java 8:**

* Before java 8 interfaces allows only public abstract methods.
* If we declare any method in interface with default it will be treated as public abstract method.
* Interface methods doesn't have body. The class which implements interfaces are responsible for implementing unimplemented methods of interface.
* But in [Java 8 static and default methods](http://www.instanceofjava.com/2015/02/java-8-interface-static-default-methods.html) added.

**Static methods in Java 8:**

* These methods are the parts of interface not belongs to implementation class objects.

1. package com.interfacesinJava8;
2. interface StaticInterface{
4. **Static**void print(String str){
6. System.out.println("Static method of interface:"+str);
8. }
9. }
10. package com.instanceofjava;
11. class Demo implements StaticInterface{
13. public static void main(String[] args){
15. StaticInterface.print("Java 8")
17. }
19. }

**Output:**

1. Static method of interface: Java 8

# **What is method overriding in java?**

* Defining multiple methods with same name and same signature in super class and sub class known as method overriding.
* Method overriding is type of polymorphism in java which is one of the main object oriented feature.
* Redefined the super class method in sub class is known as method overriding.
* Method overriding allows sub class to provide specific implementation that is already defined in super class.
* Sub class functionality replaces the super class method functionality (implementation).

# **Can we override private methods in java?**

* No. Its not possible to override private methods because private methods in super class will not be inherited to sub class.

# **Can we override static methods of super class in sub class?**

* NO. It’s not possible to override static methods because static means class level so static methods not involve in inheritance.
  1. **Can we change the return type of overridden method in sub class?**
* No. Return type must be same in super class and sub class.

1. package MethodOverridingExamplePrograms;
2. public class Super{
4. void add(){
5. System.out.println("Super class add method");
6. }
7. }

1. package MethodOverridingInterviewPrograms;
2. public class Sub extends Super{
4. int add(){    //Compiler Error: The return type is incompatible with Super.add()
6. System.out.println("Sub class add method");
7. return 0;
8. }
9. }

# [**Top 10 interview questions on java interfaces**](http://www.instanceofjava.com/2016/03/java-interface-interview-questions.html)

# **What is an interface in Java?**

* Before Java 8 interfaces are pure abstract classes which allow us to define public static final variables and public abstract methods (by default).
* In java 8 introduced static methods and default methods in interfaces.
* [Java 8 Interface Static and Default Methods](http://www.instanceofjava.com/2015/02/java-8-interface-static-default-methods.html)
* We can develop interfaces by using "interface" keyword.
* A class will implements all the methods in an interface.

# **Can we create non static variables in an interface?**

* No. We cannot create non static variables in an interface.
* If we try to create non static variables compile time error comes.
* By default members will be treated as public static final variables so it expects some value to be initialized.

1. package com.instanceofjava;
2. interface JavaInterface{
4. int x, y; // compile time error
5. }

# **Can we create object for an interface?**

* NO. We cannot create object for interface.
* We can create a variable for an interface

1. package com.instanceofjava;
2. interface JavaInterface{
4. void show();
5. }
6. package com.instanceofjava;
7. interface A implements JavaInterface {
9. void show(){
10. // code
11. }
12. public static void main(String args[]){
14. JavaInterface obj= new JavaInterface(); // Error: Cannot instantiate the type JavaInterface
16. }
17. }

# **How can we access same variables defined in two interfaces implemented by a class?**

* By using corresponding interface.variable name we can access variables of corresponding interfaces.

# **If same method is defined in two interfaces can we override this method in class implementing these interfaces?**

* Yes implementing the method once is enough in class.
* A class cannot implement two interfaces that have methods with same name but different return type.

# [**Private constructor in java**](http://www.instanceofjava.com/2016/02/private-constructor-java-example.html)

# **Can a constructor in Java be private?**

* Yes we can declare private constructor in java.
* If we declare constructor as private we cannot able to create object of the class.
* In singleton design pattern we use this private constructor.

# **In what scenarios we will use private constructor in java.**

* [Singleton Design pattern](http://www.instanceofjava.com/2015/05/singleton-design-pattern-in-java.html)
* It won’t allow class to be sub classed.
* It won’t allow to create object outside the class.
* If All Constant methods is there in our class we can use private constructor.
* If all methods are static then we can use private constructor.

# **Super Keyword:**

* The functionality of super keyword is only to point the immediate super class object of the current object.
* super keyword is applicable only in the non static methods and super keyword not applicable in the static methods.
* super keyword used to access the members of the super class object.
* super.member;
* It is used to store super class non static members memory reference through current sub class object for separating super class members from subclass members.
* We can call super class constructor in sub class using super() call.
* We can access super class methods and variables in sub class using super.variable\_name, super.method();

**1. By using super keyword we can access super class variables in sub class.**

* Using super keyword we can access super class variables from sub class.
* super.variable\_name.

1. package com.superkeywordinjava;
2. public Class SuperDemo{
4. int a,b;
6. }
7. package com.superkeywordinjava;
8. public Class Subdemo extends SuperDemo{
9. int a,b;
10. void disply(){
11. System.out.println(a);
12. System.out.println(b);
13. super.a=10;
14. super.b=20;
16. System.out.println(super.a);
17. System.out.println(super.b);
19. }
21. public static void main (String args[]) {
22. Subdemo obj= new Subdemo();
24. obj.a=1;
25. obj.b=2;
27. obj.disply();

30. }
31. }

**Output:**

1. 1
2. 2
3. 10
4. 20

**2. By using super keyword we can access super class methods in sub class.**

* Using super keyword we can call super class methods from sub class.
* super.method();.

1. package com.instanceofjavaforus;
2. public Class SuperDemo{
3. int a,b;
5. public void show() {
6. System.out.println(a);
7. System.out.println(b);
9. }
10. }
11. package com.instanceofjavaforus;
12. public Class Subdemo extends SuperDemo{
13. int a,b;
14. void disply(){
15. System.out.println(a);
16. System.out.println(b);
18. super.a=10;
19. super.b=20;
21. **super.show();**
23. }
25. public static void main (String args[]) {
27. Subdemo obj= new Subdemo();
29. obj.a=1;
30. obj.b=2;
32. obj.disply();
34. }
35. }

**Output:**

1. 1
2. 2
3. 10
4. 20

**3. We can call super class constructor from class constructor:**

* By using super keyword we can able to call super class constructor from sub class constructor.
* Using super();
* For the super(); call must be first statement in sub class constructor.

1. package com.superinterviewprograms;
2. public Class SuperDemo{
4. int a,b;
6. SuperDemo(int x, int y){
7. a=x;
8. b=y
9. System.out.println("Super class constructor called ");
10. }

13. }
14. package com.superinterviewprograms;
16. public Class Subdemo extends SuperDemo{
18. int a,b;
20. SubDemo(int x, int y){
21. super(10,20);
22. a=x;
23. b=y
24. System.out.println("Sub class constructor called ");
25. }
27. public static void main (String args[]) {
28. Subdemo obj= new Subdemo(1,2);
30. }
31. }

**Output:**

1. Super class constructor called
2. Sub class constructor called

**Key points:**

* Super(); call must be first statement inside constructor.
* By default in every class constructor JVM adds super(); call inside the constructor as first statement which calls default constructor of super class, if we are not calling it.
* If we want to call explicitly we can call at that time default call won’t be there.

**4. What will happen if we are calling super() in constructor but our class does not extending any class?**

* if our class not extending any class Yes still we can use super(); call in our class
* Because in java every class will extend Object class by default this will be added by JVM.
* But make sure we are using only super(); default call we cannot place parameterized super call because Object class does not have any parameterized constructor.

1. package com.superinterviewprograms;
3. public Class Sample{
5. Sample(){
6. super();
7. System.out.println("Sample class constructor called ");
9. }
11. public static void main (String args[]) {
13. Sample obj= new Sample();
15. }
16. }

**Output:**

1. Sample class constructor called

**5. What if there is a chain of extended classes and 'super' keyword is used**

1. package com.superinterviewprograms;
2. public Class A{
4. A(){
5. System.out.println("A class constructor called ");
6. }

9. }
10. package com.superinterviewprograms;
11. public Class B extends A{
13. B(){
15. System.out.println("B class constructor called ");
17. }

20. }
21. package com.superinterviewprograms;
22. public Class C extends B{
24. C(){
25. System.out.println("C class constructor called ");
26. }

29. public static void main (String args[]) {
30. C obj= new C();
32. }
33. }

**Output:**

1. A class constructor called
2. B class constructor called
3. C class constructor called

**6. Can we call super class methods from static methods of sub class?**

* No we cannot use super in static methods of sub class because super belongs to object level so we cannot use super in static methods.
* If we try to use in sub class static methods compile time error will come.

# [**Encapsulation in java with example program**](http://www.instanceofjava.com/2015/03/program-on-encapsulation.html)

* Binding the data with its related functionalities known as encapsulation
* Here data means variables and functionalities means methods.
* So keeping the variable and related methods in one place.
* That place is "class". class is the base for encapsulation.
* Let’s see example program on encapsulation, how the variables and methods defined in a class

2. public class EncapsulationDemo {
4. String name;
5. int rno;
6. String address;
8. public String getName() {
9. return name;
10. }
12. public void setName(String name) {
13. this.name = name;
14. }
15. public int getRno() {
16. return rno;
17. }
19. public void setRno(int rno) {
20. this.rno = rno;
21. }
23. public String getAddress() {
24. return address;
25. }
27. public void setAddress(String address) {
28. this.address = address;
29. }
31. public void showInfo(){
33. System.out.println("Name: "+getName());
34. System.out.println("R.No: "+getRno());
35. System.out.println("Name: "+getAddress());
37. }
38. }

# [**Top 10 Java interview questions on final keyword**](http://www.instanceofjava.com/2016/03/java-interview-questions-final-keyword.html)

**1. What is the use of final keyword in java?** 

* By using final keyword we can make
* Final class
* Final method
* Final variables
* If we declare any class as final we cannot extend that class
* If we declare any method as final it cannot be overridden in sub class
* If we declare any variable as final its value unchangeable once assigned.

**2. What is the actual use of final class in java?**

* If a class needs some security and it should not participate in inheritance in this scenario we need to use final class.
* We cannot extend final class.

**3. Is it possible to declare final variables without initialization?**

* No. It’s not possible to declare a final variable without initial value assigned.
* While declaring itself we need to initialize some value and that value cannot be change at any time.

**4. Can we create object for final class?**

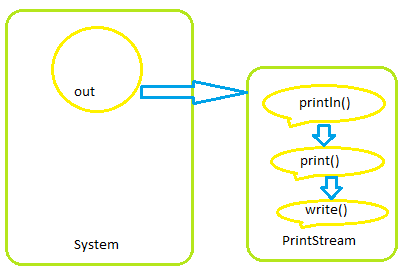
* Yes we can create object for final class.

**5. What is the most common predefined final class object you used in your code?**

* String (for example)

# [**Explain System.out.println()?**](http://www.instanceofjava.com/2014/12/1explain-systemoutprintln.html)

* System is a class Which is present in java.lang package.  
  Out is a static final field (variable) of  Printstream class  
  Println(): method of Printstream Class
* Class System{  
  public static final Printstream Out;  
  }
* Class Printstream{  
  public void println(){}}

[](http://3.bp.blogspot.com/-iZGwXwGmMWw/VHxKx93uAMI/AAAAAAAAAFs/GdPoS1ZTEBY/s1600/System.png)

# [**Non static blocks in java example**](http://www.instanceofjava.com/2016/03/non-static-blocks-in-java-example.html)

* Whenever object created non static blocks will be executed before the execution of constructor
* Non static blocks are class level block which does not have prototype

1. package nonstaticblocks;
2. public class A {
4. {
6. System.out.println("non static block executed");
8. }
10. }

**1. What is the need of Non static blocks in java?**

* To execute any logic whenever object is created irrespective of constructor used in object creation.

**2. Who will execute Non static blocks?**

* Non static blocks are automatically called by JVM for every object creation in java stack area

**3. How many Non static blocks we can create?**

* We can create any number of Non static blocks

**4. Order of execution of non static blocks**

* Order of execution of non static blocks will be order as they are defined.

1. package nonstaticblocks;
2. public class A {
4. {
5. System.out.println("first block");
6. }
8. {
9. System.out.println("second block");
10. }
12. {
13. System.out.println("third block");
14. }
15. public static void main(String[] args) {
16. A obj= new A();
17. }
18. }

**Output:**

1. first block
2. second block
3. third block

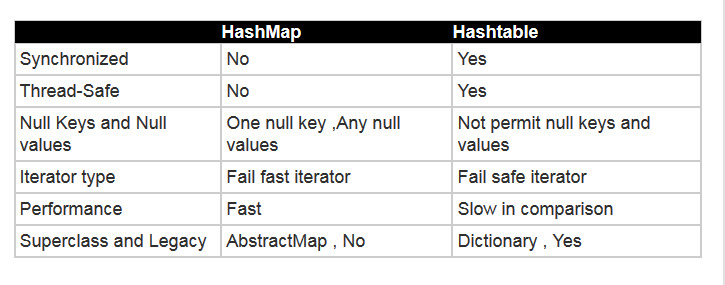
# [**Top 20 collection framework interview questions and answers in java**](http://www.instanceofjava.com/2015/07/collections-interview-questions-java.html)

**1. Why Map interface doesn’t extend Collection interface?**

* Set is unordered collection and does not allows duplicate elements.
* List is ordered collection allows duplicate elements.
* Whereas Map is key-value pair.
* It is viewed as set of keys and collection of values.
* Map is a collection of key value pairs so by design they separated from collection interface.

**2. What is difference between HashMap and Hashtable?**

* Synchronization or Thread Safe
* Null keys and null values
* Iterating the values
* Default Capacity

[](https://2.bp.blogspot.com/-3WTfMcf4CG0/VwX2BAuSxZI/AAAAAAAAAv0/8rX_HZvjVKgZLaWbeL0HVUJrv27GUwhtg/s1600/Hashmap+vs+hashtable.jpg)

**3. Differences between comparable and comparator?**

* Comparable Interface is actually from java.lang package.
* It will have a method compareTo(Object obj)to sort objects
* Comparator Interface is actually from java.util package.
* It will have a method compare(Object obj1, Object obj2)to sort objects

**4. How can we sort a list of Objects?**

* To sort the array of objects we will use  Arrays.sort() method.
* If we need to sort collection of object we will use Collections.sort().

**5. What is difference between fail-fast and fail-safe?**

* Fail fast is nothing but immediately report any failure. Whenever a problem occurs fail fast system fails.
* In java fail fast iterator while iterating through collection of objects sometimes concurrent modification exception will come there are two reasons for this.
* If one thread is iterating a collection and another thread trying to modify the collection.
* And after remove() method call if we try to modify collection object

**6. What is difference between Iterator ,ListIterator and Enumeration?**

* Enumeration interface implemented in java 1.2 version.So Enumeration is legacy interface.
* Enumeration uses elements() method.
* Iterator is implemented on all Java collection classes.
* Iterator uses iterator() method.
* Iterator can traverse in forward direction only.
* ListIterator is implemented only for List type classes
* ListIterator uses listIterator() method.

**7. What is difference between Set and List in Java?**

* A set is a collection that allows unique elements.
* Set does not allow duplicate elements
* Set allows only one null value.
* Set having classes like :
* HashSet
* LinkedHashSet
* TreeSet
* List having index. and ordered  collection
* List allows n number of null values.
* List will display Insertion order with index.
* List having classes like :
* Vector
* ArrayList
* LinkedList

**8. Differences between arraylist and vector?**

* Vector was introduced in  first version of java . that's the reason only vector is legacy class.
* ArrayList was introduced in java version1.2, as part of java collections framework.
* Vector is  synchronized.
* ArrayList is not synchronized.

**9. What are the classes implementing List interface?**

* ArrayList
* LinkedList
* Vector

**10. Which all classes implement Set interface ?**

* HashSet
* LinkedHashSet
* TreeSet

**11. How to make a collection thread safe?**

* Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment.
* By using Collections.synchronizedList(list)) we can make list classes thread safe.
* By using    
  java.util.Collections.synchronizedSet()  we can make set classes thread safe.

**12. Can a null element added to a TreeSet or HashSet?**

* One null element can be added to hashset.
* TreeSet does not allow null values

**13. Which design pattern Iterator follows?**

* Iterator design pattern

**14. Which data structure HashSet implements**

* Hashset implements hashmap internally.

**15. Why doesn't Collection extend Cloneable and Serializable?**

* List and Set and queue extends Collection interface.
* SortedMap extends Map interface.

**16. What is the importance of hashCode() and equals() methods? How they are used in Java?**

* equals() and hashcode() methods defined in "object" class.
* If equals() method return true on comparing two objects then hashcode() of those two objects must be same.

**17. What is difference between array & arraylist?**

* Array is collection of similar type of objects and fixed in size.
* Arraylist is collection of homogeneous and heterogeneous elements.

**18. What is the Properties class?**

* Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key and the value is String.

**19. How to convert a string array to arraylist?**

String[] words = {"ace", "boom", "crew", "dog", "eon"};

List<String> wordList = Arrays.asList(words);

 arrayList.toArray(); from list to array

# [**Top 10 Java Interview Questions On main() Method**](http://www.instanceofjava.com/2015/08/java-interview-questions-on-main-method.html)

**1. Can we define a class without main method?**

* No, you can’t run java class without main method.
* Before Java 7, you can run java class by using static initializers. But, from Java 7 it is not possible.

**2. Can main() method take an argument other than string array?**

* No, argument of main() method must be string array.
* But, from the introduction of var args you can pass var args of string type as an argument to main() method. Again, var args are nothing but the arrays.

1. package com.instanceofjava;
2. public class MainMethod
3. {
4. public static void main(String args[])
5. {
6. }
7. }

**3. Can we change return type of main() method?**

* No, the return type of main() method must be void only. Any other type is not acceptable.

1. package com.instanceofjava;
2. public class A
3. {
4. public static int main(String[] args)
5. {
6. return 1;    //run time error : No main method found
7. }
8. }

**4. Why main() method must be static?**

* main() method must be static.
* If main() is allowed to be non-static, then while calling the main method JVM has to instantiate it’s class.
* While instantiating it has to call constructor of that class. There will be an ambiguity if constructor of that class takes an argument.
* For example, In the below program what argument JVM has to pass while instantiating class “A”?

1. package com.instanceofjava;
2. public class A
3. {
4. public A(int i)
5. {
6. //Constructor taking one argument
7. }
8. public void main(String[] args)
9. {
10. //main method as non-static
11. }

**5. Can We Declare main() Method As Non-Static?**

* No, main() method must be declared as static so that JVM can call main() method without instantiating it’s class.
* If you remove ‘static’ from main() method signature, compilation will be successful but program fails at run time.

1. package com.instanceofjava;
2. public class A
3. {
4. public void main(String[] args)
5. {
6. System.out.println("indhu");         //Run time error
7. }
8. }

**6. Can We Overload main() method?**

* Yes, We can overload main() method. A Java class can have any number of main() methods. But to run the java class, class should have main()
* Method with signature as “public static void main(String[] args)”. If you do any modification to this signature, compilation will be successful.
* But, you can’t run the java program. You will get run time error as main method not found.

1. package com.instanceofjava;
2. public class A
3. {
4. public static void main(String[] args)
5. {
6. System.out.println("Indhu");
7. }
8. void main(int args)
9. {
10. System.out.println("Sindhu");
11. }
12. long main(int i, long d)
13. {
14. System.out.println("Saidesh");
15. return d;
16. }
17. }

**7. Can we declare main() method as private or protected or with no access modifier?**

* No, main() method must be public. You can’t define main() method as private or protected or with no access modifier.
* This is because to make the main() method accessible to JVM. If you define main() method other than public, compilation will be successful but you will get run time error as no main method found.

1. package com.instanceofjava;
2. public class A
3. {
4. private static void main(String[] args)
5. {
6. //Run time error
7. }
8. }

**8. Can we override main in Java?**

* No you cannot override main method in Java, Why because main is static method and in Java static method is bonded during compile time and you can not
* Override static method in Java.

**9. Can we make main final in Java?**

* You can make main method final in Java. JVM has no issue with that. Unlike any final method you cannot override main in Java.

**10. Can we make main synchronized in Java?**

* Yes, main can be synchronized in Java, synchronized modifier is allowed in main signature and you can make your main method synchronized in Java.

# [**Can we call super class static method from subclass in java**](http://www.instanceofjava.com/2016/02/can-we-call-superclass-static-method.html)

* If you want to call static method of a class we can call directly from another static method or by using its class name we can call static method of that class.
* let us see a program on how to call a static method of a class

1. package com.instanceofjava;
2. public class Sample{
3. public static void show(){
5. System.out.println("show() method called");
7. }
8. public static void main(String args[]){
10. show();
11. Sample.show();
13. }
14. }

**Output:**

1. show() method called
2. show() method called

* Now our question is can we call super class static method from sub class?
* Yes we can call super class static method inside sub class using super\_class\_method();
* We can also call super class static method using Sub\_class\_name.superclass\_staticMethod()

1. package com.instanceofjava;
3. public class SuperDemo{
4. public static void show(){
6. System.out.println("Super class show() method called");
8. }
10. }
11. package com.instanceofjava;
12. public class SubDemo extends SuperDemo{
13. public void print(){
15. System.out.println("Sub class print() method called");
17. }
18. public static void main(String args[]){
20. SuperDemo.show();
21. SubDemo.show();
22. }
23. }

**Output:**

1. Super class show() method called
2. Super class show() method called

* If the same static method defined in sub class also then we cannot call super class method using sub class name if we call them sub class static method will be executed.

1. package com.instanceofjava;
3. public class SuperDemo{
4. public static void show(){
6. System.out.println("Super class show() method called");
8. }
10. }
11. package com.instanceofjava;
12. public class SubDemo extends SuperDemo{
13. public static void show(){
15. System.out.println("Sub class show() method called");
17. }
18. public static void main(String args[]){
20. SuperDemo.show();
21. SubDemo.show();
22. }
23. }

**Output:**

1. Super class show() method called
2. Sub class show() method called

# [**What is serialization?**](http://www.instanceofjava.com/2014/12/what-is-serialization.html)

* Serializable means transferable
* The concept of transferring object of a class from one location to another location is known as Serialization.
* All the java classes can be divided into two.
* The classes whose objects can be transferable.  Objects of all classes  that are implementing serializable  interface can be transferable
* The classes whose objects can't be transferable.  Objects of all classes which are not implementing serializable  interface can’t be transferable.
* To transfer the object need to convert to byte stream :  serializing
* To receive Object need to convert from byte stream to Object : de-serializing
* Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.

public class Student implements java.io.Serializable

{

public String name;

public int number ;

}

import java.io.\*;

public class SerializeDemo

{

public static void main(String [] args)

{

Employee e = new Employee();

e.name = "harsha";

e.number = 101;

try

{

FileOutputStream fileOut =

new FileOutputStream("/tmp/student.ser");

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(e);

out.close();

fileOut.close();

System.out.printf("Serialized data is saved in /tmp/employee.ser");

}catch(IOException i)

{

i.printStackTrace();

}

}

}

**Deserialization**

import java.io.\*;

public class DeserializeDemo

{

public static void main(String [] args)

{

Student e = null;

try

{

FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");

ObjectInputStream in = new ObjectInputStream(fileIn);

e = (Student ) in.readObject();

in.close();

fileIn.close();

}catch(IOException i)

{

i.printStackTrace();

return;

}catch(ClassNotFoundException c)

{

System.out.println("student class not found");

c.printStackTrace();

return;

}

System.out.println("Deserialized student...");

System.out.println("Name: " + e.name);

System.out.println("Number: " + e.number);

}

}

**Transient Keyword:**

* The keyword transient in Java used to indicate that the variable should not be serialized
* By default all the variables in the object is converted to persistent state. In some cases, you may want to avoid persisting some variables because you don’t have the necessity to transfer across the network. So, you can declare those variables as transient.
* If the variable is declared as transient, then it will not be persisted. It is the main purpose of the transient keyword.

# [**What is abstract class and interfaces?**](http://www.instanceofjava.com/2014/12/what-is-abstract-class-and-interfaces.html)

**Abstract Class:**

* Abstract class means hiding the implementation  and showing the function definition to the user is known as Abstract class
* Abstract classes having Abstract methods and normal methods (non abstract methods) will be there.
* Abstract classes having methods will be anything means public ,private,protected.  
  In Abstract classes variables will be anything( public, private, protected)  
  For Abstract classes we not able to create object directly.But Indirectly we can create object using sub class object.
* A Java abstract class should be extended using keyword “extends”.
* A Java abstract class can have instance methods that implements a default behavior.If you know requirement and partially implementation you can go for Abstract classes. Abstract class can extend from a class or from an abstract class.
* Abstract class can extend only one class or one abstract class at a time. so Abstract classes can't support multiple inheritance.
* In comparison with java Interfaces, java Abstract classes are fast.  
  If you add new method to abstract class, you can provide default implementation of it. So you don’t need to change your current code.
* Abstract classes can have constructors.  
  We can run an abstract class if it has main() method.

**Interface:**

* Interface nothing but some set of rules.
* Interfaces having only Abstract methods.it is purely Achieve Abstraction.
* In Interfaces by default the methods will be public abstract methods.
* In Interfaces by default the variables will be static final .
* For Interfaces we can't create object directly or Indirectly but we can give sub class object reference to interface .
* Java interface should be implemented using keyword “implements”.
* methods of a Java interface are implicitly abstract and cannot have implementations.
* If u don't know Any requirement and any implementation you can go for Interfaces.
* Interface can extend only from an interface
* Interface can extend any number of interfaces at a time. so interfaces can support multiple inheritance(syntactical not implementation of multiple inheritance).
* In comparison with java abstract classes, java interfaces are slow as it requires extra indirection.
* if  you add new method to interface, you have to change the classes which are implementing that interface
* Interface having no constructor.  
  we can’t run an interface because they can’t have main method implementation.

# [**How to call garbage collector explicitly?**](http://www.instanceofjava.com/2014/12/how-to-call-garbage-collector-explicitly.html)

* When there are no more references to an object, the object is finalized and when the Garbage Collections starts these finalized objects gets collected this will done automatically by jvm.
* But if we want to call  Garbage collection explicitly, There are methods  
  1.System.gc();  
  2.Runtime.gc();
* How to prove?
* The java.lang.Runtime.freeMemory() method returns the amount of free memory in the Java Virtual Machine. Calling the gc method may result in increasing the value returned by freeMemory
* Class GcDemo{  
    
  public static void main(String args[]){  
  System.out.println(Runtime.getRuntime().freeMemory());  
    
      for (int i=0;i<= 100000;i++) {  
      Double d = new Double(225);  
      }  
      System.out.println(Runtime.getRuntime().freeMemory());  
      System.gc();  
      System.out.println(Runtime.getRuntime().freeMemory());  
  }

# [**Five different ways to create objects in java?**](http://www.instanceofjava.com/2014/12/four-different-ways-to-create-objects.html)

There are four different ways to create objects in java:

1. Using new keyword
2. Using Class.forName():
3. Using clone():
4. Using Object Deserialization:
5. Using newIntance() method

**Using new keyword:**

    This is the most common way to create an object in java. Almost 99% of objects are created in this way.

Object object=new Object ();

**Using Class.forName():**

* If we know the name of the class & if it has a public default constructor we can create an object in this way.
* Syntax:
* Myobject obj=(MyObject) class.forName("object").newInstance();

**Using clone():**

* The clone() can be used to create a copy of an existing object.
* Syntax:
* MyObject obj=new MyObject();
* MyObject object=(MyObject )obj.clone();

**Using Object Deserialization:**

* Object deserialization is nothing but creating an object from its serialized form.
* Syntax:
* objectInputStream istream=new objectInputStream(some data);
* MyObject object=(MyObject) instream.readObject();

**Using newInstance() method**  
  
Object obj = DemoClass.class.getClassLoader().loadClass("DemoClass").newInstance();

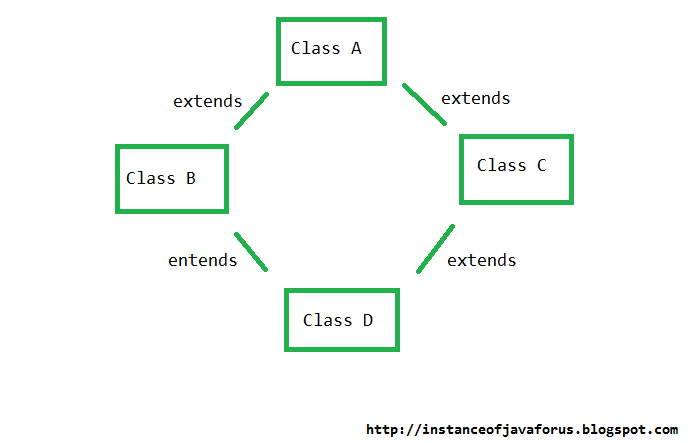
# [**Why Java does not supports multiple inheritance?**](http://www.instanceofjava.com/2014/12/why-java-does-not-supports-multiple.html)

* The concept of getting the properties from multiple class objects to sub class object with same priorities is known as multiple inheritance.
* Java Doesn't Support multiple Inheritance.

### **Diamond problem:**

* In multiple inheritance there is every chance of multiple properties of multiple objects with the same name available to the sub class object with same priorities leads for the ambiguity.

1. //Multiple inheritance program
2. Class A{
3. }
4. Class B extends A{
5. public void show(){
6. }
7. }
8. Class C extends A{
9. public void show(){
10. }
11. }
12. Class D extends B,C{  // not supported by java leads to syntax error.
13. }

[](http://3.bp.blogspot.com/-MtQIpcmOlmE/VJe6XEVaVZI/AAAAAAAAAFo/Plki98loeOI/s1600/mutliple+inheritance+not+suooported.png)

* We have two classes B and c which are inheriting A class properties.
* Here Class D inheriting B class and C class so properties present in those classes will be available in java.
* But both classes are in same level with same priority.
* If we want to use show() method that leads to ambiguity
* This is called diamond problem.
* Because of multiple inheritance there is chance of the root object getting created more than once.
* Always the root object i.e object of object class hast to be created only once.
* Because of above mentioned reasons multiple inheritance would not be supported by java.
* Thus in java a class cannot extend more than one class simultaneously. At most a class can extend only one class.
* So these are the reasons that java does not supports multiple inheritance.

# **What is JVM and is it platform independent?**

Java Virtual Machine (JVM) is the heart of java programming language. JVM is responsible for converting byte code into machine readable code. JVM is not platform independent, thats why you have different JVM for different operating systems. We can customize JVM with Java Options, such as allocating minimum and maximum memory to JVM. It’s called virtual because it provides an interface that doesn’t depend on the underlying OS.

# **What is the difference between JDK and JVM?**

Java Development Kit (JDK) is for development purpose and JVM is a part of it to execute the java programs.

JDK provides all the tools, executables and binaries required to compile, debug and execute a Java Program. The execution part is handled by JVM to provide machine independence.

# **Which class is the superclass of all classes?**

java.lang.Object is the root class for all the java classes and we don’t need to extend it.

# **Why Java is not pure Object Oriented language?**

Java is not said to be pure object oriented because it support primitive types such as int, byte, short, long etc. I believe it brings simplicity to the language while writing our code. Obviously java could have wrapper objects for the primitive types but just for the representation, they would not have provided any benefit.

As we know, for all the primitive types we have wrapper classes such as Integer, Long etc that provides some additional methods.

# **What is difference between path and classpath variables?**

PATH is an environment variable used by operating system to locate the executables. That’s why when we install Java or want any executable to be found by OS, we need to add the directory location in the PATH variable. If you work on Windows OS, read this post to learn [how to setup PATH variable on Windows](http://www.journaldev.com/476/java-tutorial-1-setting-up-java-environment-on-windows).

Classpath is specific to java and used by java executables to locate class files. We can provide the classpath location while running java application and it can be a directory, ZIP files, JAR files etc.

# **What is the importance of main method in Java?**

main() method is the entry point of any standalone java application. The syntax of main method is public static void main(String args[]).

main method is public and static so that java can access it without initializing the class. The input parameter is an array of String through which we can pass runtime arguments to the java program. Check this post to learn [how to compile and run java program](http://www.journaldev.com/481/java-hello-world-program).

# **What is overloading and overriding in java?**

When we have more than one method with same name in a single class but the arguments are different, then it is called as method overloading.

Overriding concept comes in picture with inheritance when we have two methods with same signature, one in parent class and another in child class. We can use @Override annotation in the child class overridden method to make sure if parent class method is changed, so as child class.

# **Can we overload main method?**

Yes, we can have multiple methods with name “main” in a single class. However if we run the class, java runtime environment will look for main method with syntax as public static void main(String args[]).

# **Can we have multiple public classes in a java source file?**

We can’t have more than one public class in a single java source file. A single source file can have multiple classes that are not public.

# **What is Java Package and which package is imported by default?**

Java package is the mechanism to organize the java classes by grouping them. The grouping logic can be based on functionality or modules based. A java class fully classified name contains package and class name. For example, java.lang.Object is the fully classified name of Object class that is part of java.lang package.

java.lang package is imported by default and we don’t need to import any class from this package explicitly.

# **What are access modifiers?**

Java provides access control through public, private and protected access modifier keywords. When none of these are used, it’s called default access modifier.  
A java class can only have public or default access modifier. Read [Java Access Modifiers](http://www.journaldev.com/2345/java-access-modifiers) to learn more about these in detail.

# **What is final keyword?**

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can’t extend it.

We can use final keyword with methods to make sure child classes can’t override it.

final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on.

Java interface variables are by default final and static.

# **What is static keyword?**

static keyword can be used with class level variables to make it global i.e all the objects will share the same variable.

static keyword can be used with methods also. A static method can access only static variables of class and invoke only static methods of the class.

# **What is finally and finalize in java?**

finally block is used with try-catch to put the code that you want to get executed always, even if any exception is thrown by the try-catch block. finally block is mostly used to release resources created in the try block.

finalize() is a special method in Object class that we can override in our classes. This method get’s called by garbage collector when the object is getting garbage collected. This method is usually overridden to release system resources when object is garbage collected.

# **Can we declare a class as static?**

We can’t declare a top-level class as static however an inner class can be declared as static. If inner class is declared as static, it’s called static nested class.  
Static nested class is same as any other top-level class and is nested for only packaging convenience.

# **What is static import?**

If we have to use any static variable or method from other class, usually we import the class and then use the method/variable with class name.

import java.lang.Math;

//inside class

double test = Math.PI \* 5;

We can do the same thing by importing the static method or variable only and then use it in the class as if it belongs to it.

import static java.lang.Math.PI;

//no need to refer class now

double test = PI \* 5;

Use of static import can cause confusion, so it’s better to avoid it. Overuse of static import can make your program unreadable and unmaintainable.

# **What is try-with-resources in java?**

One of the Java 7 features is try-with-resources statement for automatic resource management. Before Java 7, there was no auto resource management and we should explicitly close the resource. Usually, it was done in the finally block of a try-catch statement. This approach used to cause memory leaks when we forgot to close the resource.

From Java 7, we can create resources inside try block and use it. Java takes care of closing it as soon as try-catch block gets finished

1. **What is static block?**

Java static block is the group of statements that gets executed when the class is loaded into memory by Java ClassLoader. It is used to initialize static variables of the class. Mostly it’s used to create static resources when class is loaded.

# **What is an interface?**

Interfaces are core part of java programming language and used a lot not only in JDK but also java design patterns, most of the frameworks and tools. Interfaces provide a way to achieve abstraction in java and used to define the contract for the subclasses to implement.

Interfaces are good for starting point to define Type and create top level hierarchy in our code. Since a java class can implements multiple interfaces, it’s better to use interfaces as super class in most of the cases.

1. **What is an abstract class?**

Abstract classes are used in java to create a class with some default method implementation for subclasses. An abstract class can have abstract method without body and it can have methods with implementation also.

abstract keyword is used to create a abstract class. Abstract classes can’t be instantiated and mostly used to provide base for sub-classes to extend and implement the abstract methods and override or use the implemented methods in abstract class.

1. **What is the difference between abstract class and interface?**

abstract keyword is used to create abstract class whereas interface is the keyword for interfaces.

Abstract classes can have method implementations whereas interfaces can’t.

A class can extend only one abstract class but it can implement multiple interfaces.

We can run abstract class if it has main() method whereas we can’t run an interface.

# **Can an interface implement or extend another interface?**

Interfaces don’t implement another interface, they extend it. Since interfaces can’t have method implementations, there is no issue of diamond problem. That’s why we have multiple inheritance in interfaces i.e an interface can extend multiple interfaces.

# **What is Marker interface?**

A marker interface is an empty interface without any method but used to force some functionality in implementing classes by Java. Some of the well known marker interfaces are Serializable and Cloneable.

# **What are Wrapper classes?**

Java wrapper classes are the Object representation of eight primitive types in java. All the wrapper classes in java are immutable and final. Java 5 autoboxing and unboxing allows easy conversion between primitive types and their corresponding wrapper classes.

# **What is Enum in Java?**

Enum was introduced in Java 1.5 as a new type whose fields consists of fixed set of constants. For example, in Java we can create Direction as enum with fixed fields as EAST, WEST, NORTH, SOUTH.

enum is the keyword to create an enum type and similar to class. Enum constants are implicitly static and final.

1. **What is Java Annotations?**

Java Annotations provide information about the code and they have no direct effect on the code they annotate. Annotations are introduced in Java 5. Annotation is metadata about the program embedded in the program itself. It can be parsed by the annotation parsing tool or by compiler. We can also specify annotation availability to either compile time only or till runtime also. Java Built-in annotations are @Override, @Deprecated and @SuppressWarnings.

1. **What is Classloader in Java?**

Java Classloader is the program that loads byte code program into memory when we want to access any class. We can create our own classloader by extending ClassLoader class and overriding loadClass(String name) method

# **What are different types of classloaders?**

There are three types of built-in Class Loaders in Java:

* 1. Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.
  2. Extensions Class Loader – It loads classes from the JDK extensions directory, usually $JAVA\_HOME/lib/ext directory.
  3. System Class Loader – It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.

# **What is break and continue statement?**

We can use break statement to terminate for, while, or do-while loop. We can use break statement in switch statement to exit the switch case. You can see the example of break statement at [java break](http://www.journaldev.com/588/java-switch-case-string). We can use break with label to terminate the nested loops.

The continue statement skips the current iteration of a for, while or do-while loop. We can use continue statement with label to skip the current iteration of outermost loop.

# **What is this keyword?**

this keyword provides reference to the current object and it’s mostly used to make sure that object variables are used, not the local variables having same name.

//constructor

public Point(int x, int y) {

this.x = x;

this.y = y;

}

We can also use this keyword to invoke other constructors from a constructor.

public Rectangle() {

this(0, 0, 0, 0);

}

public Rectangle(int width, int height) {

this(0, 0, width, height);

}

public Rectangle(int x, int y, int width, int height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

# **What does super keyword do?**

super keyword can be used to access super class method when you have overridden the method in the child class.

We can use super keyword to invoke super class constructor in child class constructor but in this case it should be the first statement in the constructor method.

package com.journaldev.access;

public class SuperClass {

public SuperClass(){

}

public SuperClass(int i){}

public void test(){

System.out.println("super class test method");

}

}

Use of super keyword can be seen in below child class implementation.

package com.journaldev.access;

public class ChildClass extends SuperClass {

public ChildClass(String str){

//access super class constructor with super keyword

super();

//access child class method

test();

//use super to access super class method

super.test();

}

@Override

public void test(){

System.out.println("child class test method");

}

}

# **What is default constructor?**

No argument constructor of a class is known as default constructor. When we don’t define any constructor for the class, java compiler automatically creates the default no-args constructor for the class. If there are other constructors defined, then compiler won’t create default constructor for us.

# **Can we have try without catch block?**

Yes, we can have try-finally statement and hence avoiding catch block.

# **What is Garbage Collection?**

Garbage Collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. In Java, process of deallocating memory is handled automatically by the garbage collector.

We can run the garbage collector with code Runtime.getRuntime().gc() or use utility method System.gc(). For a detailed analysis of Heap Memory and Garbage Collection

1. **What is Serialization and Deserialization?**

We can convert a Java object to an Stream that is called Serialization. Once an object is converted to Stream, it can be saved to file or send over the network or used in socket connections.

The object should implement Serializable interface and we can use java.io.ObjectOutputStream to write object to file or to any OutputStream object. Read more at [Java Serialization](http://www.journaldev.com/927/how-to-write-object-to-file-in-java).

The process of converting stream data created through serialization to Object is called deserialization.

1. **What is the use of System class?**

Java System Class is one of the core classes. One of the easiest way to log information for debugging is System.out.print() method.

System class is final so that we can’t subclass and override it’s behavior through inheritance. System class doesn’t provide any public constructors, so we can’t instantiate this class and that’s why all of it’s methods are static.

Some of the utility methods of System class are for array copy, get current time, reading environment variables.

# **What is instanceof keyword?**

We can use instanceof keyword to check if an object belongs to a class or not. We should avoid it’s usage as much as possible. Sample usage is:

public static void main(String args[]){

Object str = new String("abc");

if(str instanceof String){

System.out.println("String value:"+str);

}

if(str instanceof Integer){

System.out.println("Integer value:"+str);

}

}

Since str is of type String at runtime, first if statement evaluates to true and second one to false.

# **Can we use String with switch case?**

One of the Java 7 feature was improvement of switch case of allow Strings. So if you are using Java 7 or higher version, you can use String in switch-case statements.

# **Java is Pass by Value or Pass by Reference?**

This is a very confusing question, we know that object variables contain reference to the Objects in heap space. When we invoke any method, a copy of these variables is passed and gets stored in the stack memory of the method. We can test any language whether it’s pass by reference or pass by value through a simple generic swap method.

# **What is difference between Heap and Stack Memory?**

Major difference between Heap and Stack memory are as follows:

* Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
* Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.
* Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally.

# **Java Compiler is stored in JDK, JRE or JVM?**

The task of java compiler is to convert java program into bytecode, we have javac executable for that. So it must be stored in JDK, we don’t need it in JRE and JVM is just the specs.