[**https://javabeginnerstutorial.com/core-java-tutorial/collection-in-java/**](https://javabeginnerstutorial.com/core-java-tutorial/collection-in-java/)

[**https://hackr.io/blog/java-interview-questions**](https://hackr.io/blog/java-interview-questions)

**Q) What JDK, JRE and JVM?**

| **JDK** | **JRE** | **JVM** |
| --- | --- | --- |
| **Java Development Kit is a Kit which provides the environment to develop and execute(run) the Java program.** | **Java Runtime Environment is an installation package which provides environment to only run the java program/application onto your machine** | **Java virtual machine contains JRE and JDK.Java program you can run using JRF and JDK goes into JVM.JVM executes program line by line it’s known Interpreter.** |
| **Development Tools(to provide an environment to develop your java programs)** | **JRE provides the runtime environment to execute/run Java bytecode** | **JVM is responsible for executing java programs line by line using an interpreter.** |
| **JRE (to execute your java program).** | **JRE contains a set of lib+other files that JVM uses at runtime.** |  |

**2. Difference between final, finally finalize?**

**1.final: final is a modifier applicable for classes, method and variable. If a class declared as final then we can’t extend that class we can’t create a child class for that class.**

**2.if a method declared as final then we can’t override that method in the child class.**

**3.if a variable declared as final then it will become constant and we can’t perform re-assignment for that variable.**

**2.finally: finally is block always associated with try and catch block to maintain cleanup code and either exception is handled or not it always executes.**

**3.finalize: finalize() is a method which is always invoked by the garbage collector just before destroying an object to perform cleanup activities.**

**3. Difference between String and StringBuffer?**

| **String** | **StringBuffer** |
| --- | --- |
| **String is immutable** | **StringBuffer is mutable** |
| **String is slower than StringBuffer when we perform concatenations.** | **StringBuffer is faster than String when we concatenate String** |
| **String is not synchronized** | **StringBuffer is synchronized** |
| **String is stored in constant string pool** | **StringBuffer is stored in Heap memory** |

**4. Difference between == operator and equals() method in java?**

**We use == operator for reference comparison and equals() method we use for content comparison.**

**5. What is the difference between StringBuffer and StringBuilder?**

| **StringBuffer** | **StringBuilder** |
| --- | --- |
| **Every method present in StringBuffer is synchronized** | **No Method present in StringBuilder is synchronized.** |
| **At a time only one thread allow to operate on StringBuffer object StringBuffer is thread-safe** | **At a time multiple threads are allow to operate on StringBuffer object so it is not thread-safe** |
| **It increase waiting time of thread so that performance is low as compare to StringBuilder** | **Thread are not required to wait to operate on StringBuilder object so performance is high** |
| **Introduced in 1.0 version** | **Introduced in 1.5 version** |

**6.Difference between Interface and abstract class?**

| **Interface** | **Abstract class** |
| --- | --- |
| **Inside interface every method is always public and abstract whether we are declaring or not.interface is 100% pure Abstract class.** | **Every method present in abstract class need not be public and Abstract. We can create concrete method and abstract method** |
| **We can not declare private and protected access modifiers in Interface** | **In abstract class we can not declare final, static, synchronized, natice access modifiers.** |
| **Inside interface we can’t declare constructors** | **Inside abstract class we can declare constructors** |
| **Inside interface we can’t declare instance and static block otherwise we will get compile time error** | **Inside abstract class we can declare instance and static block** |
| **For interface variable compulsory we should perform initialization at the time of declaration otherwise we will get compile time error** | **For abstract class variable it is not required to perform initialization at the time of declaration** |
| **Every variable present inside interface is always public Static, final whether we are declare or not** | **The variables present inside abstract class need not be public, static, and final** |
| **Interface support multiple inheritance** | **Abstract class does not support multiple inheritance** |
|  |  |

**7. What is a constructor in java?**

**Constructor refers to a block of code which is used to initialize an object. it must have the same name as that of class.it has no return type and it is automatically called when an object is created.**

**There are two type of constructor**

**1.Default constructor**

**2.Parematerized Constructor**

**8. What is a checked Exception and unchecked Exception?**

**The exception which is checked by the compiler for a smooth exception of the program at runtime is called Checked Exception.**

**Eg: IOException**

**The Exception which are not checked by the compiler are called Unchecked Exception**

**Eg: ArithmeticException , RuntimeException**

**Q> What is OOPs Concept**

Answer : Object oriented programming provides great flexibility, reusability, modularity and security to an application.

**Q> What is Object ?**

Answer : Object is a real world entity that has state and behavior. an object can be defined as an instance of class and object contains an address and creates some space in memory.

**Example**: a dog has an object because it has state and behavior.state represents the properties of an Object like a color and name, breed and behavior represent the action and functionality like an eating,Walking,Running ,Wagging.

1> **Realtime Example**: pencil a pencil is an object it name as natraj

State : color,name, color is black and name is natraj

Behavior : action and functionality writing is behavior

2>**Realtime Example** :Phone phone is an object it name as samsung phone

State : color,name,os,brand,price weight

Color:black name : samsung os: android brand:samsung weight:130gm

Behavior : call,send sms,sharing

**Create Object :** An Object of class is created in three steps.

1> Declaration of a reference variable

2> Creation of an Object

3> Linking the object and reference the reference

Create Object : new Classname();

Example : new Phone(); this statement tells to JVM to Allocate memory space for a new Phone() object

On the heap

Phone obj = new Phone();

**subQ> what is object reference ?**

Answer : An object reference is a unique hexadecimal number that represents a memory address of the object. It is useful to access members of objects. When a new object is created, a new reference number is allocated to it. It means that every object in Java will have a unique reference.

**subQ>What is a new keyword in Java?**

Answer : In Java, a new operator is a special keyword which is used to create an object of the class. It allocates the memory to store an object during runtime and returns a reference to it. This reference is the address of the object in the heap memory allocated by the new operator. This reference (memory address) is then stored in a variable called object reference variable that can be accessed from anywhere in the application. See below image.

**Q> what is Class**

Answer :class is Collection of an object it represents common properties,state behavior and variable.it is logical entity

That's called class

**Realtime example 2:**

Consider two objects: one boy and one girl. The boy has some properties like hairColor="black", eyeColor="black", skinColor="Fair" height="5.10 inch", weight="65 kg" and actions are read(), play(), sleep(), walk().

Now, The girl has some properties like hairColor="Brown", eyeColor="brown", skinColor="milky white", height="5.4 inch", weight="50 kg" and actions are read(), play(), sleep(), walk(). but the type of both is the same. i,e Person. So the class name is 'Person'.

**Declaration of class**

Syntax :

modifierName class className

{

// class body.

}

Example :

Public class Phone

{

// class body.

}

Class is blueprint of component template it contains following things

1>**Modifiers**(public private protected default

2>**Class** name (naming conversion)

3>**Body**: 1. Field declarations;

2. Constructor declarations;

3. Method declarations;

4. Instance block declarations;

5. Static block declarations;

4>**Fields**: Fields are the variables that provide the state/properties of the class and its object. It may be a Local variable, Instance variable or Static variable.

5>**Constructor**: Constructors are used to initialize the values of variables and new objects. A constructor can be divided into two types such as Default constructor and User-defined constructor.

6>**Methods**: methods are used to implement the actions or behavior of the class and its object. It may be an Instance method or a Static method.

7>**Blocks**: blocks are mostly used to change the default values of variables. It may be an instance block or static block.

Interface: It is used to achieve the multiple inheritances in Java.

**Q>Difference between Class and Object in Java**

1. A class is a user-defined data type whereas an object is an instance of class data type.

2. A class generates objects whereas an object gives life to a class.

3. Classes do not occupy memory location but objects occupy memory location.

4. Classes cannot be manipulated due to not being available in the memory location but objects can be manipulated.

**Q)Difference between ‘throw’ and ‘throws’ in Java Exception Handling?**

Answer :***throw*** keyword is used to throw Exception from any method or static block whereas ***throws*** is used to indicate which Exception can possibly be thrown by this method.

**Q)What is Method overloading?**

Method Overloading means to have two or more methods with the same name in the same class with different arguments.

Method Overloading is a **Compile time polymorphism**. In method overloading, more than one method shares the same method name with different signatures in the class. In method overloading, return type can or can not be the same, but we must have to change the parameter because in java, we can not achieve the method overloading by changing only the return type of the method.

**Note:**

* Overloaded methods MUST change the argument list
* Overloaded methods CAN change the return type
* Overloaded methods CAN change the access modifier
* Overloaded methods CAN declare new or broader checked exceptions
* A method can be overloaded in the same class or in a subclass

**Example :**

**public class MethodOverLoadingEx{**

**public int sum(int a,int b){**

**return a+b;**

**}**

**public int sum(int a,int b,int c){**

**return a+b+c;**

**}**

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

**MethodOverLoadingEx obj = new MethodOverLoadingEx ();**

**obj.sum(10,10);**

**ob.sum(20,20,20);**

**}**

**}**

**Q>)What is Inheritance?**

Inheritance is the process by which objects of one class acquire the properties & objects of another class. The two most common reasons to use inheritance are: a) To promote code reuse. b) To use polymorphism.

### **Q)What are the different types of inheritance in Java?**

1.**Single Inheritance**: In single inheritance, one class inherits the properties of another i.e there will be only one parent as well as one child class.

2.**Multilevel Inheritance**: When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.

3.**Hierarchical Inheritance:** When a class has more than one child class (subclasses) or in other words, more than one child class have the same parent class, then such kind of inheritance is known as hierarchical.

4.**Hybrid Inheritance:** Hybrid inheritance is a combination of two *or more types* of inheritance

**Q)What is Method overriding?**

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass

Method Overriding is a **Run time polymorphism**. In method overriding, derived class provides the specific implementation of the method that is already provided by the base class or parent class. In method overriding, return type must be the same or co-variant (return type may vary in the same direction as the derived class).

Note:

* You cannot override a method marked final
* You can’t override a method marked public and make it protected
* You cannot override a method marked static
* You cannot override a method marked static
* Static methods cannot be overridden. Overloaded methods can still be overridden.

**public class Animal{**

**public void eat(){ println(“eating”); }**

**}**

**public class Dog extends Animal{**

**public void eat(){println(“dog is eating”);}**

**}**

**public class MethodOverriding(){**

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

**Dog obj = new Dog();**

**obj.eat() // child class eat method will call because jvm runtime environment**

**Animal a1 =new Animal()**

**a1.eat(); // parent class eat method will be call**

**}}**

**Q)What is Abstract class?**

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.

If even a single method is abstract, the whole class must be declared abstract.

Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

You can’t mark a class as both abstract and final.

Non-abstract methods can access a method that you declare as abstract.

**Q)What is Abstraction ?**

Abstraction is an oops concept that hides the data value or data members and shows only essential data that need to be shown to the user.Avoids code duplication and increases reusability.

Ex: A car is viewed as a car rather than its individual components

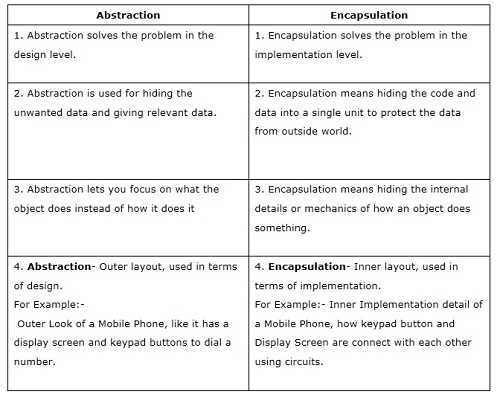
**Q)what is Encapsulation and Abstraction**

**Encapsulation** is basically to bind data members & member functions into a single unit called Class

**Abstraction** is basically to hide complexity of implementation & provide ease of access to the users

Example Of Abstraction and Encapsulation

smartphone is an abstract where the inner implementation details are encapsulated



Note:increase security of an application or program as only important details are provided to the user

**Abstract class and Abstract Method Example:->**

**abstract class Animal{**

**abstract void eat();**

**abstract void sleep();**

**void show(){ System.out.println(“parent: concrete method”);}**

**}//animal end brackets**

**class Dog extends Animal{**

**@Override**

**void eat(){**

**System.out.println(“dog is eating”)**

**}**

**@Override**

**void sleep(){**

**System.out.println(“dog is sleeping”)**

**}**

**}// Dog end brackets**

**public class AbstractClassEx{**

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

**// we can reference variable of abstract class but we can not create instance or object of abstract class**

**Animal a = new Dog(); // a is reference variable of abstract class**

**a.show();**

**a.eat();**

**a.sleep();**

**}**

**}**

**Encapsulation Example:->**

**class User{**

**private String name;**

**private int roleno;**

**//getter and setter**

**public void setName(String name){**

**This.name =name**

**}**

**public void setRoleNo(Int roleNo){**

**this.roleno = roleNo;**

**}**

**public String getName(){**

**return name;}}**

**Q) What is Interface?**

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method bodies. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

**Note:**

* Java Interface also represents the IS-A relationship.
* It cannot be instantiated just like the abstract class.
* Since Java 8, we can have **default and static methods** in an interface.
* Since Java 9, we can have **private methods** in an interface.
* Variables declared in interface are **public, static and final** by default.
* Does not allow **private** and **protected** access modifiers

**Example of Interface:->**

**Public interface Animal{**

**String name = “sai”;**

**void eat();**

**void sleep();**

**/**/java 8 we can create *static* and *default* method using **static** and **default** keyword

**static void show(){**

**System.*out*.println("interface static method block in java 8 features");**

**}**

**default void details(){**

**System.*out*.println("interface default method block in java 8 features:"+*name*);**

**}// interface end brackets**

**public class Cat implements Animal{**

**@Override**

**public void eat() {System.*out*.println("dog is eating");}**

**@Override**

**public void sleep() { System.*out*.println("dog is sleeping");}} //** cat end brackets

**public class InterfaceMainClass{**

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

**Animal.*show*();** // static method block we call using interface name

**Animal a = new Dogs();** // we can create reference variable of interface but we can;t create object of interface

**a.eat();**

**a.details();**

**a.sleep();**

**}**

**} //** InterfaceMainClass end brackets

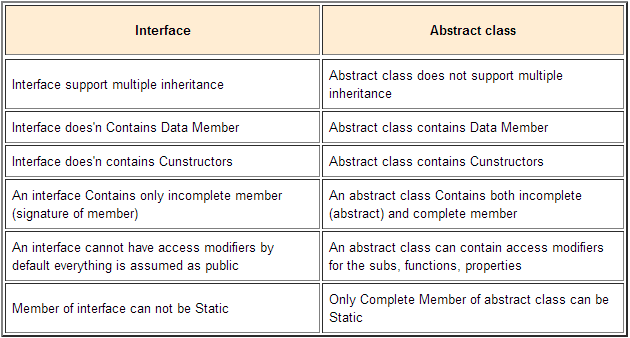
**Q)Encapsulation vs Data Abstraction**

1.[Encapsulation](http://contribute.geeksforgeeks.org/encapsulation-in-java/) is data hiding(information hiding) while Abstraction is detail hiding(implementation hiding).

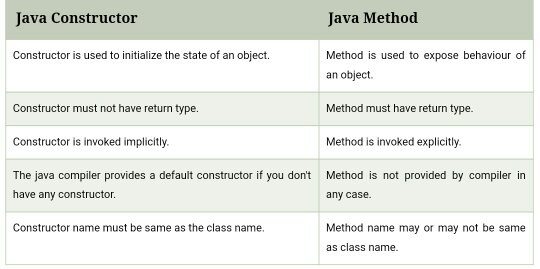
2.While encapsulation groups together data and methods that act upon the data, data abstraction deals with exposing the interface to the user and hiding the details of implementation.

**Q)Difference between Abstract and Interfaces?**

Abstract classes can have non abstract methods. It can have instance variables. We have provided default implementation to abstract class methods. A class can extend only one abstract class.A class can implement multiple interfaces.



**Q)What difference between Constructor and Method?**



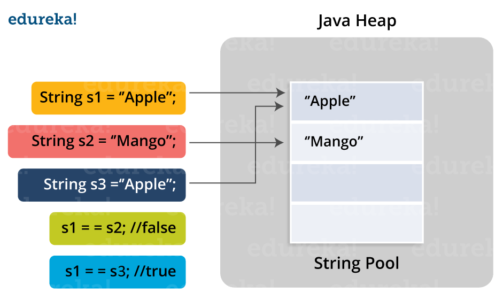
**Q)Difference between == and .equals() method in Java?**

We can use == operators for reference comparison (address comparison) and .equals() method for content comparison. \* In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

**Q)Why are strings Immutable?**

Once a value is assigned to a string it cannot be changed. And if changed, it creates a new object of the String. This is not the case with StringBuffer.

**Q)What is String Pool in Java?**

String Pool is a storage area in Java heap.

**Example Of String Pool**

**public class StringPool{**

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

**String s1 = “sainath”;**

**String s2 =”sainath”;**

**String s3 = new String("sainath");**

**String s4 = new String("sainath").intern();**

**System.out.println(s1 == s2); // true**

**System.out.println(s1 == s3); // false**

**System.out.println(s1 == s4); // true**

**}**

**}**

**Note:**When double equals operator is used to compare two objects, it returns true when they are referring to the same object, otherwise false . ... When string is creating using new operator, it gets created in the heap space

S1 and s2 is referring same object

**Q)What is String.intern()? When and why should it be used?**

1.String.intern() method can be used to deal with String duplication problems in Java. By carefully using the intern() method you can save a lot of memories consumed by duplicate String instances. A string is duplicate if it contains the same content as another string but occupies different memory locations.

2.By calling the intern() method on a string object, for instance “abc”, you can instruct JVM to put this String in the pool and whenever someone else creates "abc", this object will be returned instead of creating a new object. This way, you can save a lot of memory in Java, depending upon how many Strings are duplicated in your program.

3.When the intern method is invoked, if the String pool already contains that String object such that equals() returns true, it will return the String object from the pool, otherwise it will add that object to the pool of unique String.

**Q)What is String pool in Java?**

* String Pool in java is a pool of Strings stored in Java Heap Memory.
* When we use double quotes to create a String, it first looks for String with the same value in the String pool, if found it just returns the reference, else it creates a new String in the pool and then returns the reference
* However using a new operator, we force String class to create a new String object in heap space. We can use the intern() method to put it into the pool or refer to other String objects from the string pool having the same value.
* For example, how many strings are getting created in below statement; String str = new String("Cat");
* In the above statement, either 1 or 2 strings will be created. If there is already a string literal “Cat” in the pool, then only one string “str” will be created in the pool. If there is no string literal “Cat” in the pool, then it will be first created in the pool and then in the heap space, so total 2 string objects will be created.

**Q)*Final* modifier?**

Final modifiers - once declared cannot be modified. A blank final variable in Java is a final variable that is not initialized during declaration.

* final Classes- A final class cannot have subclasses.
* final Variables- A final variable cannot be changed once it is initialized.
* final Methods- A final method cannot be overridden by subclasses.

**Q)*Finalize* Keywords?**

Finalize is a method used to perform clean up processing just before the object is garbage collected.

**Q)*Finally* keyword?**

finally is a code block and is used to place important code, it will be executed whether an exception is handled or not.

**Q)What does the keyword synchronized mean?**

* When you have two threads that are reading and writing to the same 'resource', say a variable named 'test', you need to ensure that these threads access the variable in an atomic way. Without the synchronized keyword, your thread 1 may not see the change thread 2 made to test.
* synchronized blocks the next thread's call to method as long as the previous thread's execution is not finished. Threads can access this method one at a time.

**Q)What does the keyword volatile mean?**

* Suppose two threads are working on a method. If two threads run on different processors each thread may have its own local copy of the variable. If one thread modifies its value the change might not reflect in the original one in the main memory instantly.
* Now the other thread is not aware of the modified value which leads to data inconsistency.Essentially, volatile is used to indicate that a variable's value will be modified by different threads. “volatile” tells the compiler that the value of a variable must never be cached as its value may change outside of the scope of the program itself.
* The value of this variable will never be cached thread-locally: all reads and writes will go straight to "main memory"
* An access to a volatile variable never has the potential to block: we're only ever doing a simple read or write, so unlike a synchronized block we will never hold on to any lock.

**Q)What is Autoboxing and Unboxing?**

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called unboxing.

**Q)What is Optionals in Java?**

Optional is a container object which is used to contain not-null objects. Optional object is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as ‘available’ or ‘not available’ instead of checking null values.

**Q)Fail-fast & Fail-Safe?**

1.Fail-fast Iterators throws ConcurrentModificationException when one Thread is iterating over collection object and another thread structurally modifies Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure.

2.On the other hand [fail-safe](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) Iterators work on copy of collection instead of original collection.

**Q)Multi threading?**

Multiple tasks are running concurrently in a program.

**Q)What is Encapsulation?**

* Encapsulation involves binding code and data together as a single unit.
* Encapsulation is a technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.
* For instance, a class can be an encapsulated class if all the variables in it are defined as Private and by providing getter and setter methods.

**Q)What is Polymorphism?**

* + Polymorphism is when an object takes on multiple forms. For instance, String is a subclass of Object class.
  + Polymorphism manifests itself in Java in the form of multiple methods having the same name.
  + In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
  + In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).
  + Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.
  + 2 forms of polymorphism:
    - Compile time polymorphism: The flow of control is decided during the compile time itself. By overloading.
    - Runtime polymorphism: is done using inheritance and interface. The flow of control is decided during the runtime. Overriding: Overriding will have the same method name with the same parameters. One will be the parent class method and the other will be the child class method. Overloading occurs when the same method name is declared but with different parameters.

**Q).What is an Exception?**

An unwanted, unexpected event that disturbs normal flow of the program is called

Exception.Example: FileNotFondException.

**Q).What is the purpose of Exception Handling?**

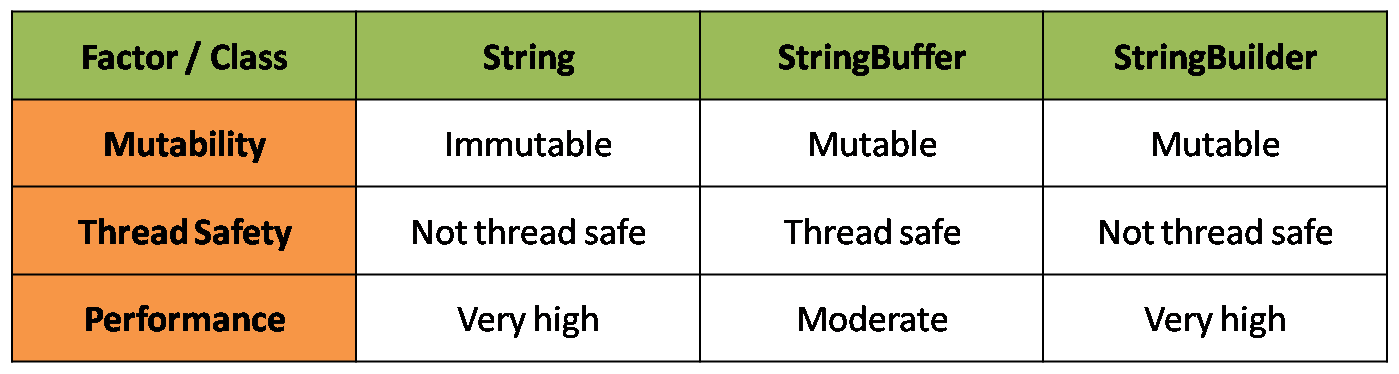
Ans.The main purpose of Exception Handling is for graceful termination of the program.

**Q).What is the meaning of Exception Handling?**

Exception Handling doesn’t mean repairing an Exception, we have to define alternative

way to continue the rest of the code normally.

**Q)String vs StringBuffer And StringBuilder?**

****

**Q). What is inner class and when should we go for inner classes?**

Some times we can declare a class inside another class such type of classes are called

inner classes

**Example Of Inner classes:->**

**public class Car{**

**void show(){//code here}**

**// inner class declarations**

**class Engine{**

**//code here**

**} //end inner class**

**}//end outer clas**s

**Note**:Without existing Car object there is no chance of existing Engine object, hence Engine class

has been declared inside the Car class.

**Collection Question:->**

**Q)What is the Collection API ? What is Collection framework**

It defines a set of classes and interfaces which can be used for representing a group of objects as a single entity.

**Q)What is the difference between Collections and Collection?**

**Collection** is an interface which can be used for representing a group of individual objects as a single entity and it acts as the root interface of the collection framework.

**Collections** is an utility class to define several utility methods for Collection implemented class objects.

**Q)Explain about List interface?**

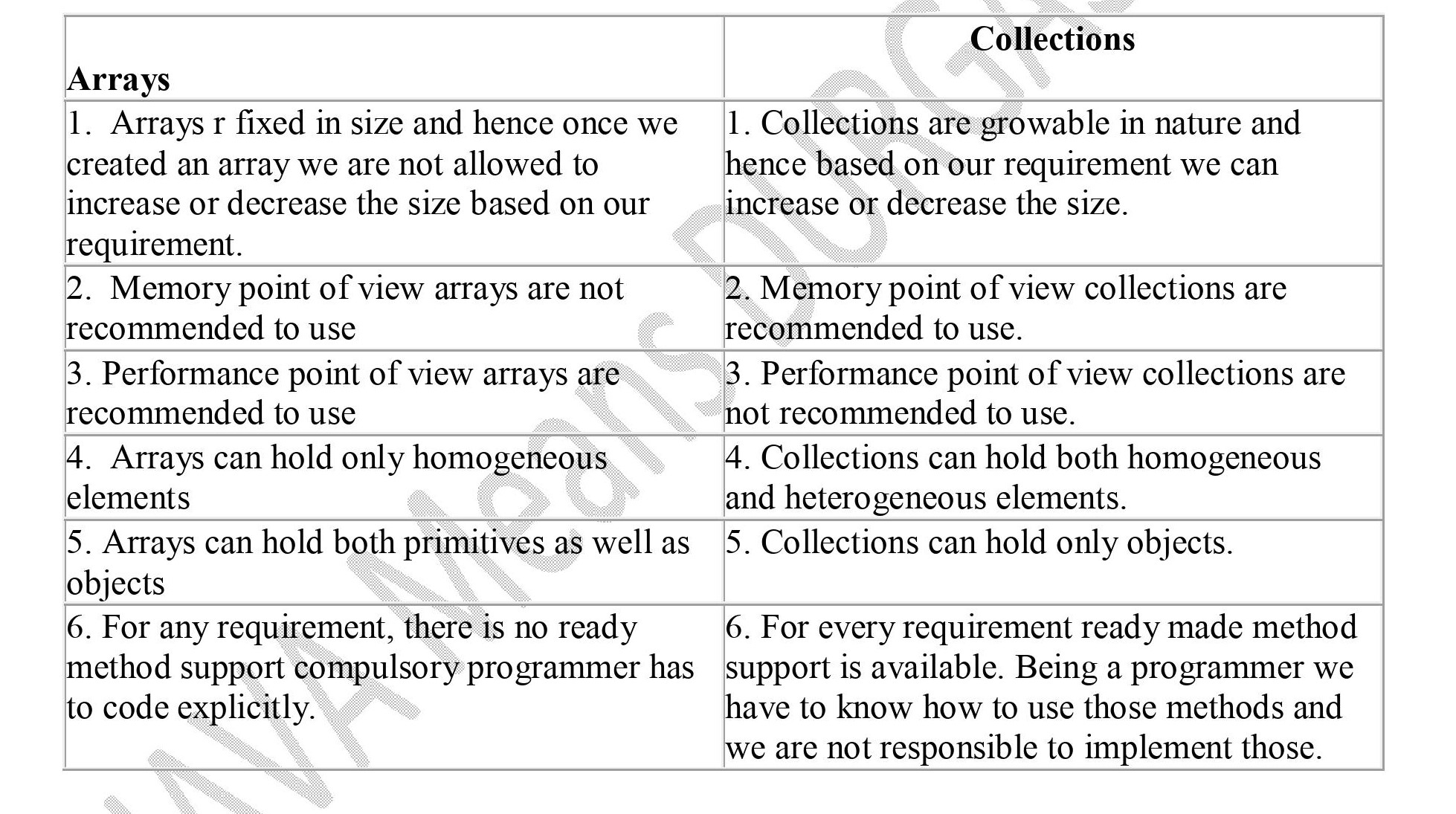
List interface is a child interface of Collection interface. This can be used to represent group of individual objects in as a single entity where Duplicates are allowed Insertion order is preserved

1.Duplicates are allowed 2.Insertion order is preserved

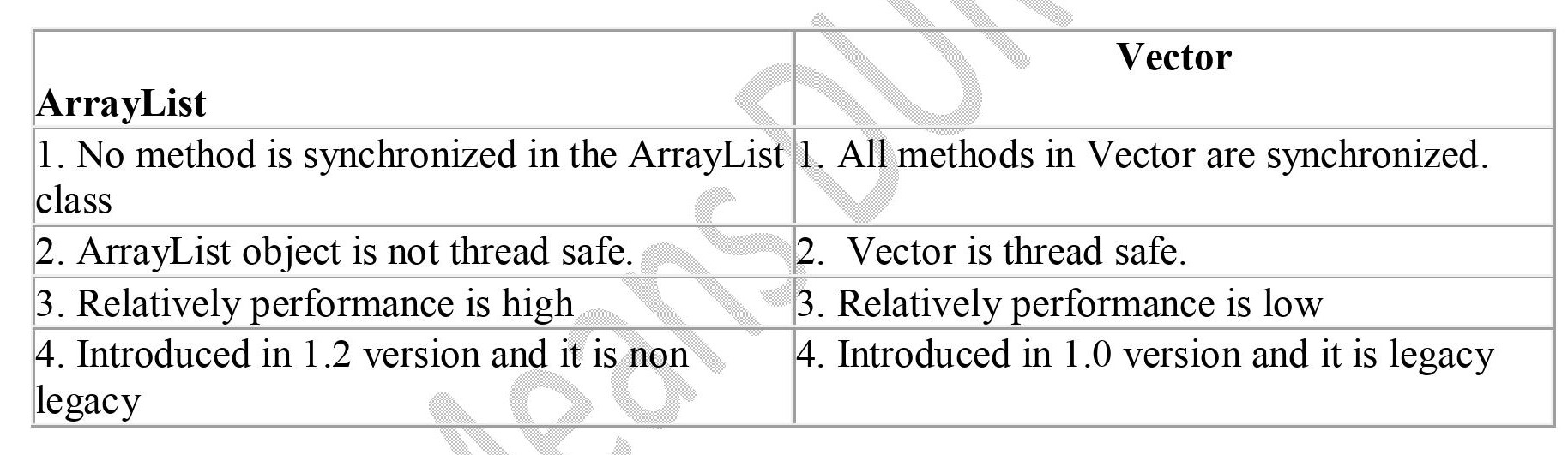
**Q)Explain about Set interface?**

Set is a child interface of Collection interface. it can be used to represent a group of individual objects as a single entity where 1.Duplicate objects are not allowed. 2. Insertion order is not preserved

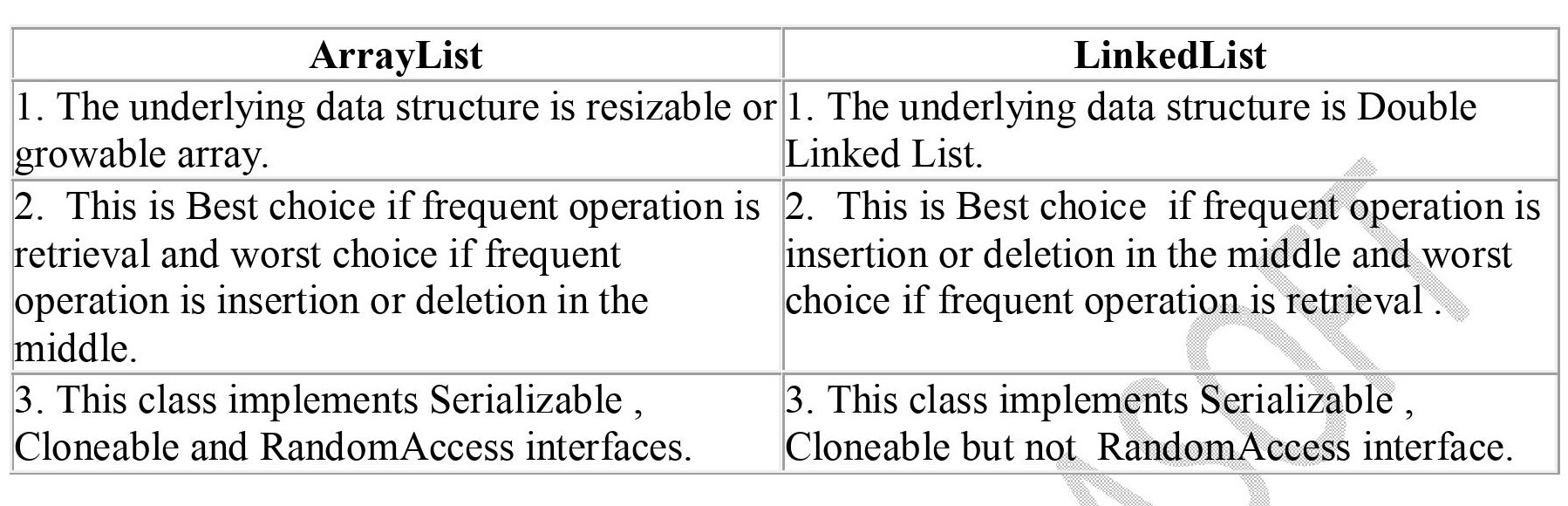
**Q)What are differences between arrays and collections? arrays and Vector? arrays and ArrayList?**

****

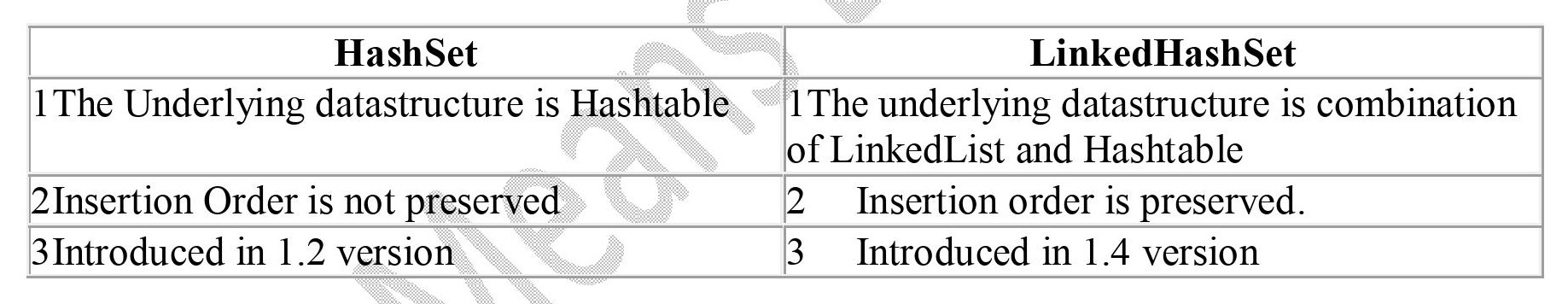
**Q) What is the difference between ArrayList and Vector?**

****

**Q)What is the difference between ArrayList and Linked List?**

****

**Q)Differences between HashSet and LinkedHashSet?**

****

**Q)What is a Comparable interface?**

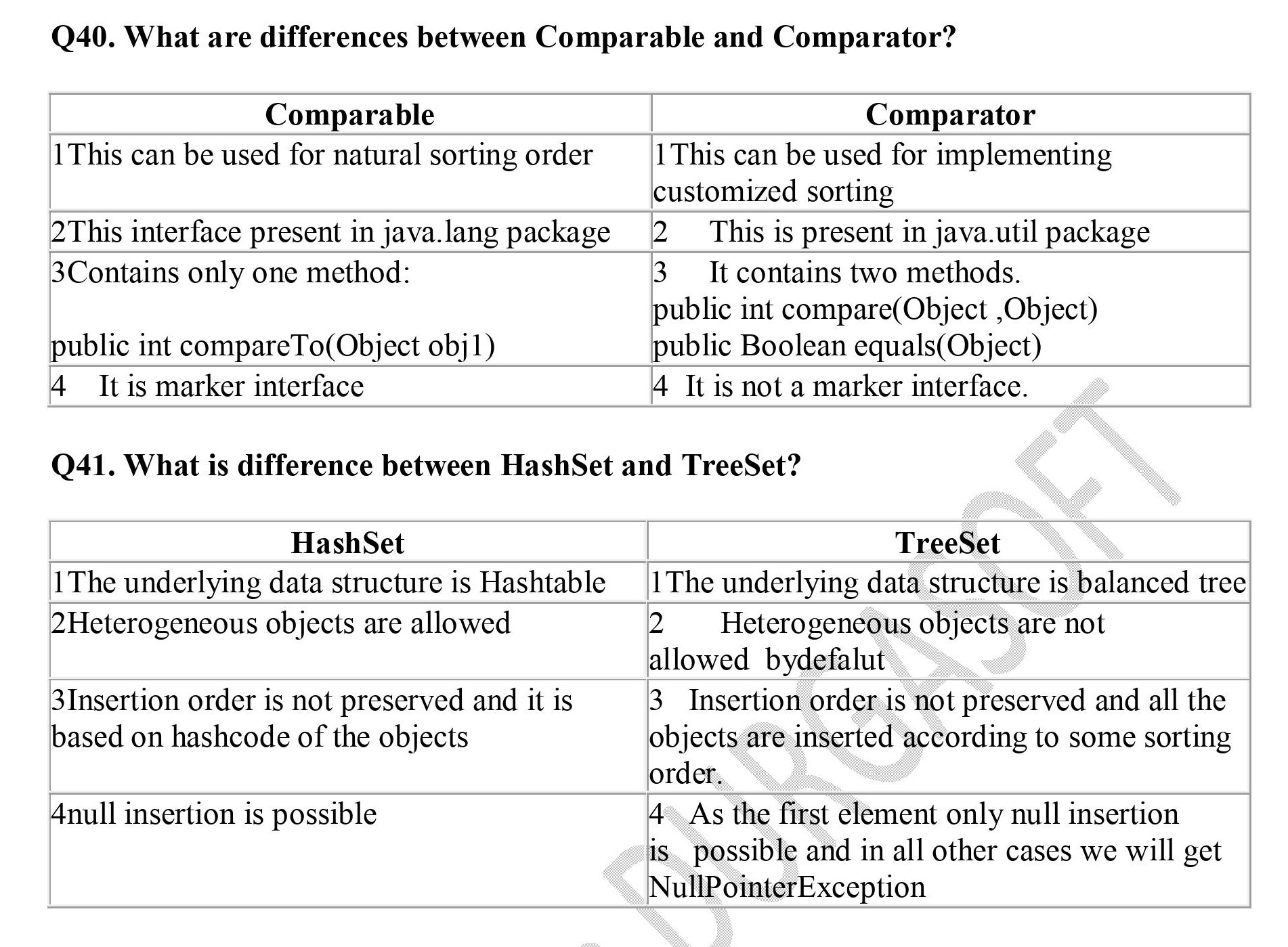
This interface can be used for defining natural sorting order of the objects.

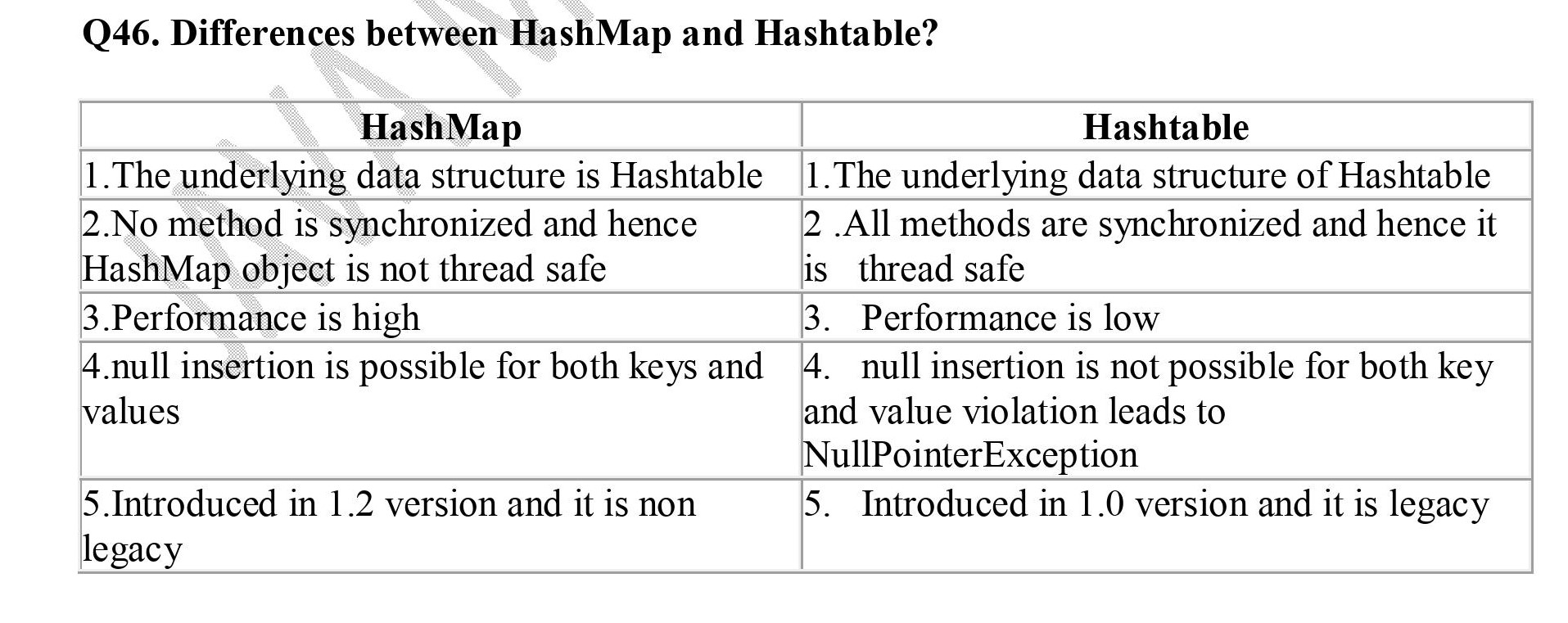
It is present in java.lang package

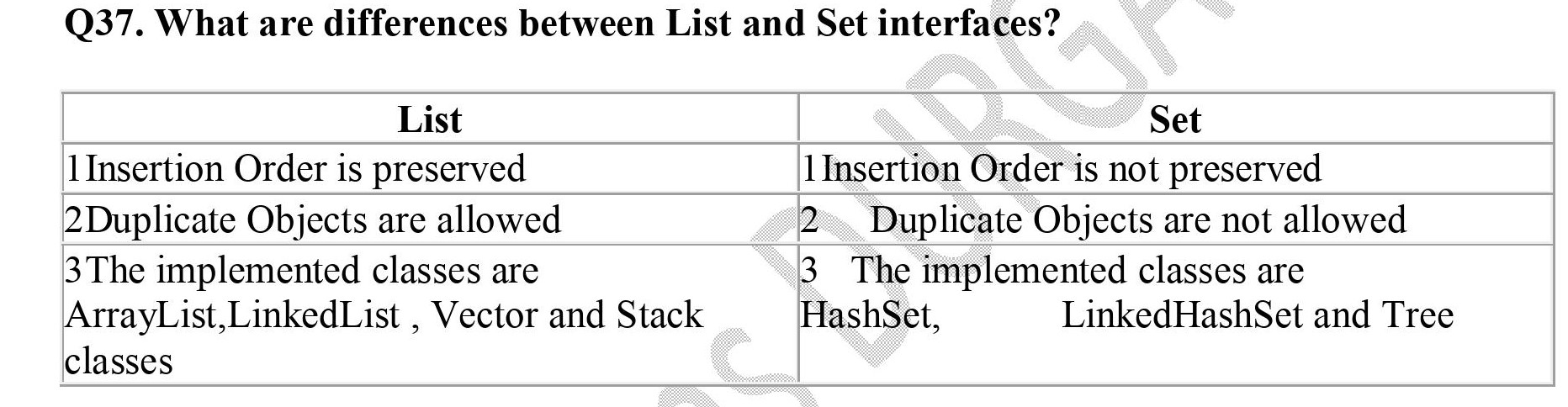
It contains a method public int compareTo(Object obj1)

**Q)What is a Comparator interface?**

1.This interface can be used for implementing customized sorting order. 2.It is present in java.util package 3. It contains two methods of public int compare(Object ,Object) o public boolean equals(Object)







**Q)Explain about ArrayList class?**

ArrayList is a Collection which can be used to represent a group of objects as a single entity.

* it is a implemented class for List interface.Introduced in 1.2 version
* The underlying data structure is resizable or growable array
* Insertion order is preserved
* Duplicates are allowed Heterogeneous objects are allowed
* null insertion is possible
* This class implements RandomAccess , Serializable , Cloneable interfaces Best choice for retrieval purpose and worst if our frequent operation is insertion or deletion in the middle.

**Q)Explain about LinkedList class?**

LinkedList is a Collection implemented class which can be used for representing a group of objects as a single entity.LinkedList is the implementation class for List interface

* Introduced in 1.2 version Underlying data Structure is DoubleLinkedList
* Allows duplicates
* Insertion order is preserved. Allows heterogeneous objects
* null insertion is possible
* LinkedList class implements Serializable and Cloneable interface but not RandomAccess interface.
* Best choice if frequent operation is insertion or deletion and objects in middle but worst choice if frequent operation is retrieval.

**MultiThreading Question**

**Q)What is Multitasking?**

Ans. Executing several tasks simultaneously is called multitasking.

**Q)Who uses Thread priority?**

Thread Scheduler uses priorities while allocating CPU. The Thread which is having

highest priority will get a chance first for execution.

**Q)What is the difference between process-based and Thread-based Multitasking?**

**1.Process-based multitasking:-** Executing several task simultaneously where each task is

A separate independent process such as multitasking is called process based Multitasking.

Example:-While typing a program in the editor we can listen to MP3 audio songs.

At the same time we download a file from the net. all these task are executing

simultaneously and each task is a separate independent program. hence it is process based

multitasking. It is best suitable at operating system level.

**2.Thread-based multitasking:-**

Executing several task simultaneously where each task is a separate independent part of the

same program is called Thread-based multitasking. and every independent part is called a

thread. This type of multitasking is best suitable at programmatic level.

**Q)What is Multithreading and explains its application areas?**

Executing several thread simultaneously where each thread is a separate independent

part of the same program is called multithreading. Java language provides inbuilt support for

multithreading by defining a reach library, classes and interfaces like Thread, ThreadGroup,

Runnable etc. The main important application area of multithreading are video games

implementation, animation development, multimedia graphics etc.

**Q)Explain about Thread Scheduler?**

If multiple threads are waiting for getting the chance for executing then which thread will get a chance is first decided by Thread Scheduler. It is the part of JVM and its behavior is vendor dependent and we can’t expect exact output.Whenever the situation comes to multithreading the guarantee behavior is very- very low.

**Q)Explain the life cycle of a Thread?**

Once we create a Thread object then the Thread is said to be in a New/Born state once we call t.start() method then the Thread will be entered into a ready/Runnable state that is Thread is ready to execute. If Thread Scheduler allocates CPU now the Thread will enter into the Running state and start execution of run() method. After completing the run() method the Thread entered into Dead State.

What is instance method

A. An instance method in Java is basically a method of the class. In other words, a non-static method which is declared inside a class is an instance method. This kind of method requires an object of its class to be created before it can be called. To invoke an instance method, we have to create the object of the class.

**What is instance block**

**A. Instance blocks in Java- Instance block is an element of a class which serves the same two purposes as served by the constructors. These two purposes are- Just like constructors, instance blocks can be used to initialize the instance variables during object creation.**

**What is static block**

**A. In a Java class, a static block is a set of instructions that is run only once when a class is loaded into memory. A static block is also called a static initialization block. This is because it is an option for initializing or setting up the class at run-time.**

**what is inheritance**

**A. Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of** [**OOPs**](https://www.javatpoint.com/java-oops-concepts) **(Object Oriented programming system).The idea behind inheritance in Java is that you can create new** [**classes**](https://www.javatpoint.com/object-and-class-in-java) **that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.Inheritance represents the IS-A relationship which is also known as a *parent-child* relationship.**

**The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.**

**class Employee{**

**float salary=40000;**

**}**

**class Programmer extends Employee{**

**int bonus=10000;**

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

**Programmer p=new Programmer();**

**System.out.println("Programmer salary is:"+p.salary);**

**System.out.println("Bonus of Programmer is:"+p.bonus);**

**}**

**}**

**Output :**

**Programmer salary is:40000.0**

**Bonus of programmer is:10000**

## **Types of inheritance in java**

**On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.**

**In java programming, multiple and hybrid inheritance is supported through interface only.**

#### **Note: Multiple inheritance is not supported in Java through class.**

**When one class inherits multiple classes, it is known as multiple inheritance. For Example:**

## **Single Inheritance Example**

**When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.**

**File: TestInheritance.java**

**class Animal{**

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

**}**

**class Dog extends Animal{**

**void bark(){System.out.println("barking...");}**

**}**

**class TestInheritance{**

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

**Dog d=new Dog();**

**d.bark();**

**d.eat();**

**}}**

**Output:**

**barking...**

**eating...**

## **Multilevel Inheritance Example**

**When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.**

**File: TestInheritance2.java**

**class Animal{**

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

**}**

**class Dog extends Animal{**

**void bark(){System.out.println("barking...");}**

**}**

**class BabyDog extends Dog{**

**void weep(){System.out.println("weeping...");}**

**}**

**class TestInheritance2{**

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

**BabyDog d=new BabyDog();**

**d.weep();**

**d.bark();**

**d.eat(); }}**

**Output:**

**weeping...**

**barking...**

**eating…**

**Hierarchical Inheritance Example**

**When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.**

**File: TestInheritance3.java**

**class Animal{**

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

**}**

**class Dog extends Animal{**

**void bark(){System.out.println("barking...");}**

**}**

**class Cat extends Animal{**

**void meow(){System.out.println("meowing...");}**

**}**

**class TestInheritance3{**

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

**Cat c=new Cat();**

**c.meow();**

**c.eat();**

**//c.bark();//C.T.Error**

**}}**

**Output:**

**meowing...**

**eating…**

## **Q) Why multiple inheritance is not supported in java?**

**To reduce the complexity and simplify the language, multiple inheritance is not supported in java.**

**Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.**

**Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.**

**class A{**

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

**}**

**class B{**

**void msg(){System.out.println("Welcome");}**

**}**

**class C extends A,B{//suppose if it were**

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

**C obj=new C();**

**obj.msg();//Now which msg() method would be invoked?**

**}**

**}**

**Output:**

**Compile Time Error**

**Rules for method overriding:**

**1. Overriding and Access-Modifiers :** The [access modifier](https://www.geeksforgeeks.org/access-modifiers-java/) for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass. Doing so, will generate compile-time error.

class Parent {

// private methods are not overridden

private void m1()

{

System.out.println("From parent m1()");

}

protected void m2()

{

System.out.println("From parent m2()");

}

}

class Child extends Parent {

// new m1() method

// unique to Child class

private void m1()

{

System.out.println("From child m1()");

}

// overriding method

// with more accessibility

@Override

public void m2()

{

System.out.println("From child m2()");

}

}

// Driver class

class Main {

public static void main(String[] args)

{

Parent obj1 = new Parent();

obj1.m2();

Parent obj2 = new Child();

obj2.m2();

}

}

**Output:**

From parent m2()

From child m2()

**2. Final methods can not be overridden :** If we don’t want a method to be overridden, we declare it as [final](https://www.geeksforgeeks.org/final-keyword-java/). Please see [Using final with Inheritance](https://www.geeksforgeeks.org/using-final-with-inheritance-in-java/) .

// A Java program to demonstrate that

// final methods cannot be overridden

class Parent {

// Can't be overridden

final void show() {}

}

class Child extends Parent {

// This would produce error

void show() {}

}

**Output:**

13: error: show() in Child cannot override show() in Parent

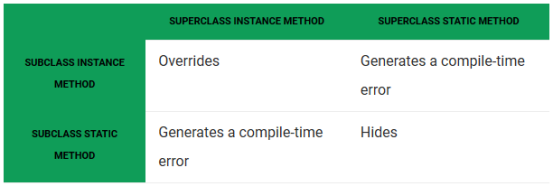
void show() { }

^

overridden method is final

**3. Static methods can not be overridden(Method Overriding vs Method Hiding) :** When you defines a static method with same signature as a static method in base class, it is known as [method hiding](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/).

The following table summarizes what happens when you define a method with the same signature as a method in a super-class.



// Java program to show that

// if the static method is redefined by

// a derived class, then it is not

// overriding, it is hiding

class Parent {

// Static method in base class

// which will be hidden in subclass

static void m1()

{

System.out.println("From parent "

+ "static m1()");

}

// Non-static method which will

// be overridden in derived class

void m2()

{

System.out.println("From parent "

+ "non-static(instance) m2()");

}

}

class Child extends Parent {

// This method hides m1() in Parent

static void m1()

{

System.out.println("From child static m1()");

}

// This method overrides m2() in Parent

@Override

public void m2()

{

System.out.println("From child "

+ "non-static(instance) m2()");

}

}

// Driver class

class Main {

public static void main(String[] args)

{

Parent obj1 = new Child();

// As per overriding rules this

// should call to class Child static

// overridden method. Since static

// method can not be overridden, it

// calls Parent's m1()

obj1.m1();

// Here overriding works

// and Child's m2() is called

obj1.m2();

}

}

**Output:**

From parent static m1()

From child non-static(instance) m2()

**4. Private methods can not be overridden :**[Private methods](https://www.geeksforgeeks.org/can-override-private-methods-java/) cannot be overridden as they are bonded during compile time. Therefore we can’t even override private methods in a subclass.

**5. The overriding method must have same return type (or subtype) :** From Java 5.0 onwards it is possible to have different return type for a overriding method in child class, but child’s return type should be sub-type of parent’s return type. This phenomena is known as [covariant return type](https://www.geeksforgeeks.org/covariant-return-types-java/).

**6. Invoking overridden method from sub-class :** We can call parent class method in overriding method using [super keyword](http://quiz.geeksforgeeks.org/super-keyword/).

// A Java program to demonstrate that overridden

// method can be called from sub-class

// Base Class

class Parent {

void show()

{

System.out.println("Parent's show()");

}

}

// Inherited class

class Child extends Parent {

// This method overrides show() of Parent

@Override

void show()

{

super.show();

System.out.println("Child's show()");

}

}

// Driver class

class Main {

public static void main(String[] args)

{

Parent obj = new Child();

obj.show();

}

}

**Output:**

Parent's show()

Child's show()

**7. Overriding and constructor :** We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).

### **Example of method overriding**

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

//Java Program to illustrate the use of Java Method Overriding

//Creating a parent class.

class Vehicle{

//defining a method

void run(){System.out.println("Vehicle is running");}

}

//Creating a child class

class Bike2 extends Vehicle{

//defining the same method as in the parent class

void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){

Bike2 obj = new Bike2();//creating object

obj.run();//calling method

}

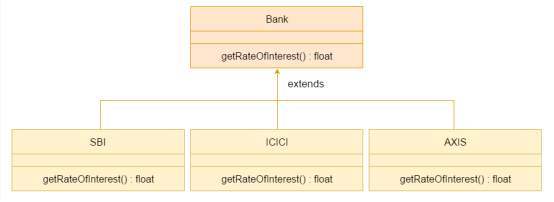
}

Output:

Bike is running safely

### **A real example of Java Method Overriding**

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



//Java Program to demonstrate the real scenario of Java Method Overriding

//where three classes are overriding the method of a parent class.

//Creating a parent class.

class Bank{

int getRateOfInterest(){return 0;}

}

//Creating child classes.

class SBI extends Bank{

int getRateOfInterest(){return 8;}

}

class ICICI extends Bank{

int getRateOfInterest(){return 7;}

}

class AXIS extends Bank{

int getRateOfInterest(){return 9;}

}

//Test class to create objects and call the methods

class Test2{

public static void main(String args[]){

SBI s=new SBI();

ICICI i=new ICICI();

AXIS a=new AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

Output:

SBI Rate of Interest: 8

ICICI Rate of Interest: 7

AXIS Rate of Interest: 9

### **Can we override static method?**

No, a static method cannot be overridden. It can be proved by runtime polymorphism

### **Why can we not override static method?**

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

### **Can we override java main method?**

No, because the main is a static method.

# **ExceptionHandling with MethodOverriding in Java**

There are many rules if we talk about methodoverriding with exception handling. The Rules are as follows:

**· If the superclass method does not declare an exception**

o If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

**· If the superclass method declares an exception**

o If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

### **If the superclass method does not declare an exception**

#### 1) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.

· import java.io.\*;

· class Parent{

· void msg(){System.out.println("parent");}

· }

·

· class TestExceptionChild extends Parent{

· void msg()throws IOException{

· System.out.println("TestExceptionChild");

· }

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

· Parent p=new TestExceptionChild();

· p.msg();

· }

· }

Output :

Output:Compile Time Error

#### **2) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.**

import java.io.\*;

class Parent{

void msg(){System.out.println("parent");}

}

class TestExceptionChild1 extends Parent{

void msg()throws ArithmeticException{

System.out.println("child");

}

public static void main(String args[]){

Parent p=new TestExceptionChild1();

p.msg();

}

}

Output:child

### **If the superclass method declares an exception**

#### **1) Rule: If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.**

### **Example in case subclass overridden method declares parent exception**

import java.io.\*;

class Parent{

void msg()throws ArithmeticException{System.out.println("parent");}

}

class TestExceptionChild2 extends Parent{

void msg()throws Exception{System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild2();

try{

p.msg();

}catch(Exception e){}

}

}

Output:Compile Time Error

### **Example in case subclass overridden method declares same exception**

import java.io.\*;

class Parent{

void msg()throws Exception{System.out.println("parent");}

}

class TestExceptionChild3 extends Parent{

void msg()throws Exception{System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild3();

try{

p.msg();

}catch(Exception e){}

}

}

Output:child

### **Example in case subclass overridden method declares subclass exception**

· import java.io.\*;

· class Parent{

· void msg()throws Exception{System.out.println("parent");}

· }

·

· class TestExceptionChild4 extends Parent{

· void msg()throws ArithmeticException{System.out.println("child");}

·

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

· Parent p=new TestExceptionChild4();

· try{

· p.msg();

· }catch(Exception e){}

· }

· }

Output:child

### **Example in case subclass overridden method declares no exception**

import java.io.\*;

class Parent{

void msg()throws Exception{System.out.println("parent");}

}

class TestExceptionChild5 extends Parent{

void msg(){System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild5();

try{

p.msg();

}catch(Exception e){}

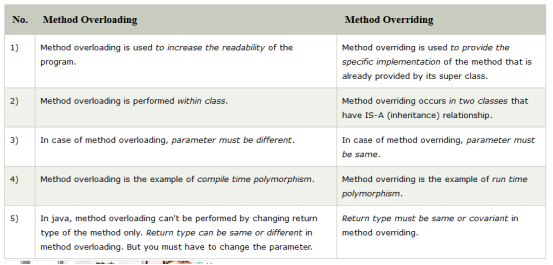
}

}

Output:child

# **Difference between method overloading and method overriding in java**

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:



## **Java Method Overloading example**

· class OverloadingExample{

· static int add(int a,int b){return a+b;}

· static int add(int a,int b,int c){return a+b+c;}

· }

## **Java Method Overriding example**

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void eat(){System.out.println("eating bread...");}

}

What is covariant return type

Covariant return type refers to return type of an overriding method. It allows to narrow down return type of an overridden method without any need to cast the type or check the return type. Covariant return type works only for non-primitive return types.

From Java 5 onwards, we can override a method by changing its return type only by abiding the condition that return type is a subclass of that of overridden method return type.

Following example showcases the same.

## **Example**

class SuperClass {

SuperClass get() {

System.out.println("SuperClass");

return this;

}}public class Tester extends SuperClass {

Tester get() {

System.out.println("SubClass");

return this;

}

public static void main(String[] args) {

SuperClass tester = new Tester();

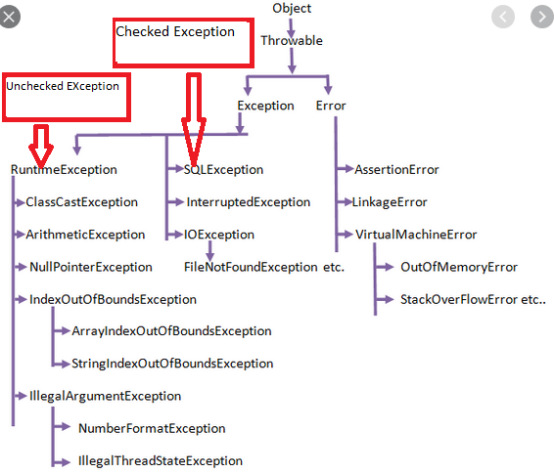
tester.get();

}}

**Output**

Subclass

Below are the following tree diagram of exception.



# **Access Modifiers in Java**

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

**1. Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**2. Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

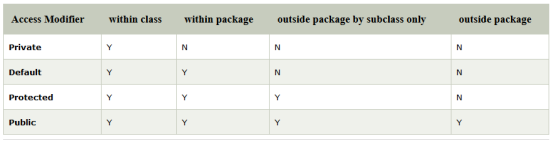
**3. Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

**4. Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

### **Understanding Java Access Modifiers**

Let's understand the access modifiers in Java by a simple table.



**1) Private**

The private access modifier is accessible only within the class.

**Simple example of private access modifier**

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

class A{

private int data=40;

private void msg(){System.out.println("Hello java");}

}

public class Simple{

public static void main(String args[]){

A obj=new A();

System.out.println(obj.data);//Compile Time Error

obj.msg();//Compile Time Error

}

}

### **Role of Private Constructor**

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

class A{

private A(){}//private constructor

void msg(){System.out.println("Hello java");}

}

public class Simple{

public static void main(String args[]){

A obj=new A();//Compile Time Error

}

}

#### **Note: A class cannot be private or protected except nested class.**

### **2) Default**

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

**Example of default access modifier**

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

//save by A.java

package pack;

class A{

void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.\*;

class B{

public static void main(String args[]){

A obj = new A();//Compile Time Error

obj.msg();//Compile Time Error

}

}

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

### **3) Protected**

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

**Example of protected access modifier**

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

//save by A.java

package pack;

public class A{

protected void msg(){System.out.println("Hello");}

}

· //save by B.java

· package mypack;

· import pack.\*;

·

· class B extends A{

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

· B obj = new B();

· obj.msg();

· }

· }

Output:Hello

### **4) Public**

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

**Example of public access modifier**

//save by A.java

package pack;

public class A{

public void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.\*;

class B{

public static void main(String args[]){

A obj = new A();

obj.msg();

}

}

Output:Hello

### **Java Access Modifiers with Method Overriding**

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

class A{

protected void msg(){System.out.println("Hello java");}

}

public class Simple extends A{

void msg(){System.out.println("Hello java");}//C.T.Error

public static void main(String args[]){

Simple obj=new Simple();

obj.msg();

}

}

The default modifier is more restrictive than protected. That is why there is a compile-time error.

System.out.println(s1 == s2); // true

**System.out.println(s1 == s3); // false**

**System.out.println(s1 == s4); // true**

**System.out.println(s1 == s2); // true**

**System.out.println(s1 == s3); // false**

**System.out.println(s1 == s4); // tru**