Difference between JDK, JRE and JVM

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

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

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

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM. It physically exists.

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.



### **Internal Architecture of JVM**

|  |
| --- |
|  |



### **1) Classloader**

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

### **2) Class(Method) Area**

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

### **3) Heap**

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

### **4) Stack**

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

### **5) Program Counter Register**

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

### **6) Native Method Stack**

It contains all the native methods used in the application.

### **7) Execution Engine**

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

# Variable in Java

# Variable is name of reserved area allocated in memory. int data=50;//Here data is variable

# variable in java

### **Types of Variable**

# There are three types of variables in java.

# types of variable

#### Local Variable

|  |
| --- |
| A variable that is declared inside the method is called local variable. |

#### Instance Variable

|  |
| --- |
| A variable that is declared inside the class but outside the method is called instance variable. It is not declared as static. |

#### Static variable

|  |
| --- |
| A variable that is declared as static is called static variable. It cannot be local. |
| 1. **class** A{ 2. **int** data=50;//instance variable 3. **static** **int** m=100;//static variable 4. **void** method(){ 5. **int** n=90;//local variable 6. } 7. }//end of class |
| **Data Types in Java**  |  | | --- | | In java, there are two types of data types   * primitive data types * non-primitive data types |  1. datatype in java |
|  |
| **Why char uses 2 byte in java and what is \u0000 ?** Because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system. |

# Java Switch Statement

# The Java switch statement is executes one statement from multiple conditions. It is like if-else-if ladder statement.

# 

# Output: - 20

# Note: - The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

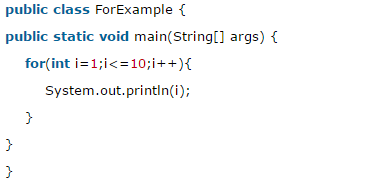
# Java For Loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

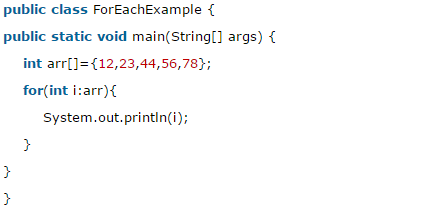
* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

## **Java Simple For Loop**



## **Java For-each Loop**

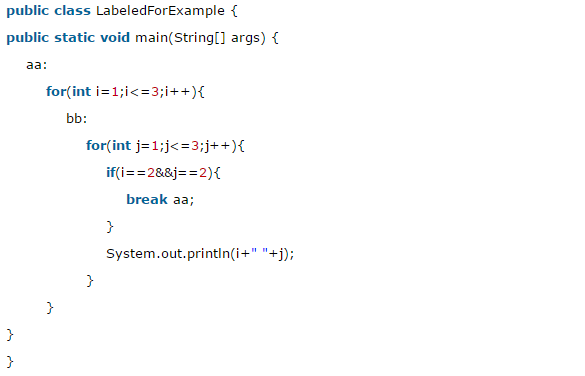
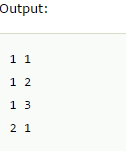
The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value.



## **Java Labeled For Loop**

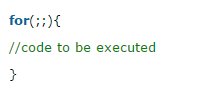
We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

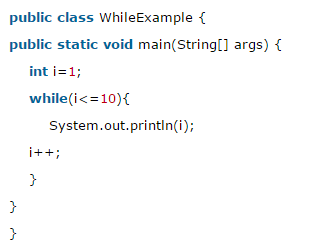
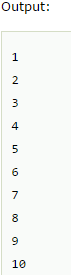
If you use **break bb;**, it will break inner loop only which is the default behavior of any loop.

## **Java Infinitive For Loop**

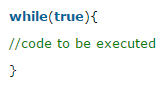
 Now, you need to press ctrl+c to exit from the program.

# Java While Loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

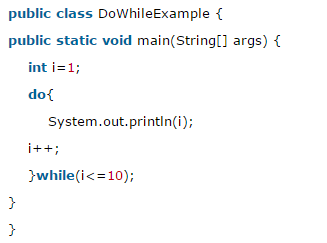
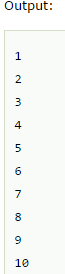
 

## **Java Infinitive While Loop**

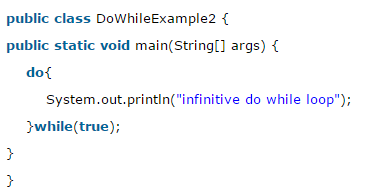
 Now, you need to press ctrl+c to exit from the program.

# Java do-while Loop

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use while loop.

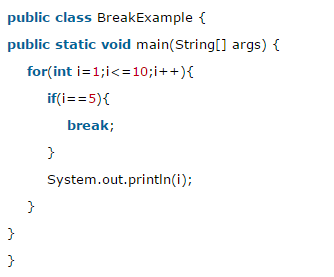
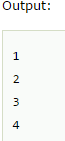
 

## **Java Infinitive do-while Loop**

 Now, you need to press ctrl+c to exit from the program.

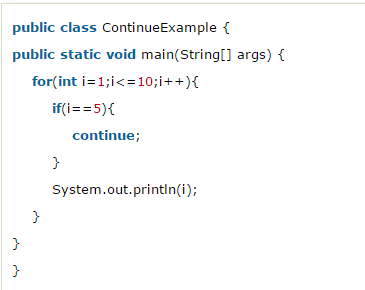
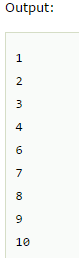
# Java Break Statement

The Java break is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

# Java Continue Statement

The Java continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

# Java Comments

The java comments are statements that are not executed by the compiler and interpreter.

## **Types of Java Comments**

There are 3 types of comments in java.

1. Single Line Comment //This is single line comment
2. Multi Line Comment

/\*

This

is

multi line

comment

\*/

1. Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**

/\*\*

This

is

documentation

comment

\*/

# Java OOPs Concepts

The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

**Smalltalk** is considered as the first truly object-oriented programming language.

**Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

## **Object**

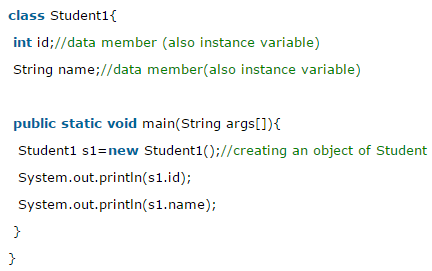
Any entity that has state (value) and behavior (functionality) is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

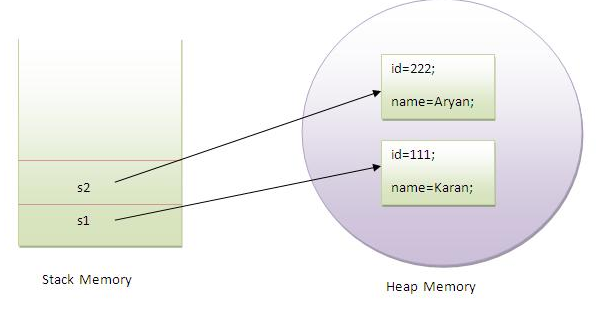
## **Class**

**Collection of objects** is called class. It is a logical entity. It is a template or blueprint from which objects are created.

**Object is an instance of a class.** Class is a template or blueprint from which objects are created. So object is the instance (result) of a class.

 Output:0 null

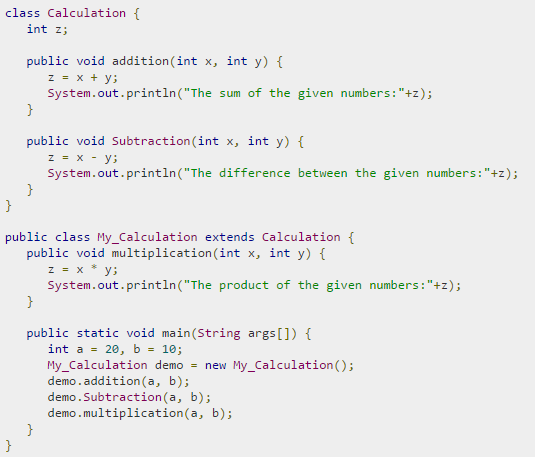
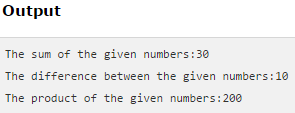
**Note:-** A variable that is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when object (instance) is created. That is why, it is known as instance variable. The **new** keyword is used to allocate memory at runtime.



Object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area.

#### **Inheritance**

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

In the given program, when an object to **My\_Calculation** class is created, a copy of the contents of the superclass is made within it. That is why, using the object of the subclass you can access the members of a superclass.

The Superclass reference variable can hold the subclass object, but using that variable you can access only the members of the superclass, so to access the members of both classes it is recommended to always create reference variable to the subclass. In above example, using the superclass reference variable ( cal in this case) you cannot call the method multiplication(), which belongs to the subclass My\_Calculation.



**Note** − A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

**Super Keyword**

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

* It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
* It is used to **invoke the superclass** constructor from subclass.

**IS-A Relationship**

IS-A is a way of saying: This object is a type of that object.

public class Animal { // Animal is superclass of Mammal, Reptile class.

}

public class Mammal extends Animal { // Mammal is subclass of Animal.

}

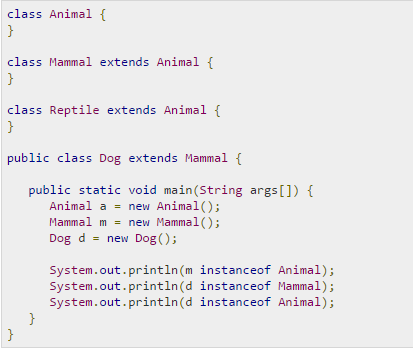
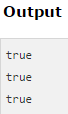
public class Reptile extends Animal { // Reptile is subclass of Animal.

}

public class Dog extends Mammal { // Dog is subclass of both Mammal and Animal.

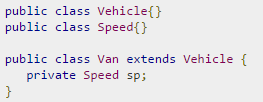
}

**instanceof Keyword**

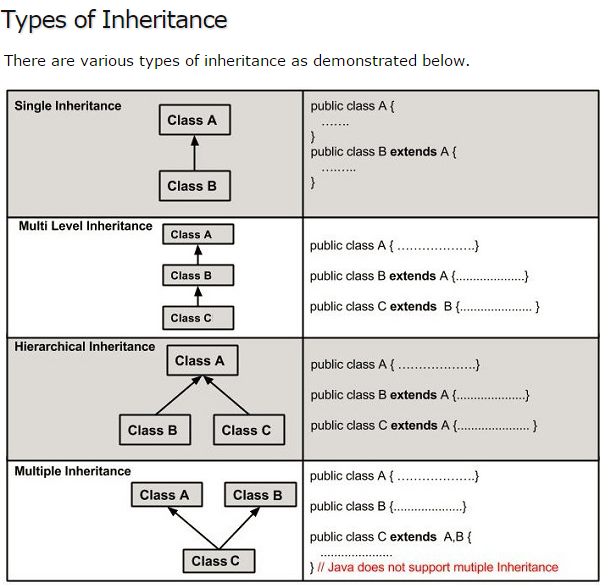
 

**HAS-A Relationship**

This determines whether a certain class **HAS-A** certain thing. This relationship helps to reduce duplication of code as well as bugs.



This shows that class Van HAS-A Speed. By having a separate class for Speed, we do not have to put the entire code that belongs to speed inside the Van class, which makes it possible to reuse the Speed class in multiple applications.



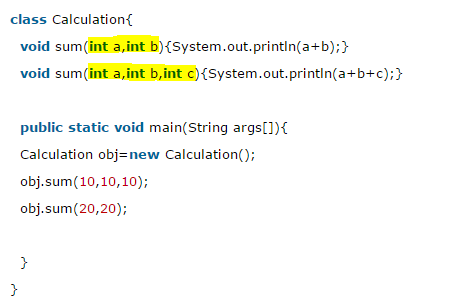
#### **Polymorphism**

When **one task is performed by different ways** i.e. known as polymorphism. In java, we use **method overloading** and **method overriding** to achieve polymorphism.

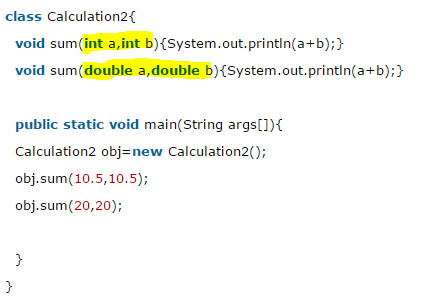
**Method Overloading (Compile time polymorphism)**

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**. There are two ways to overload the method in java

1. By changing number of arguments

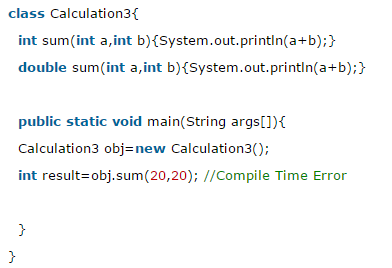
 

1. By changing the data type

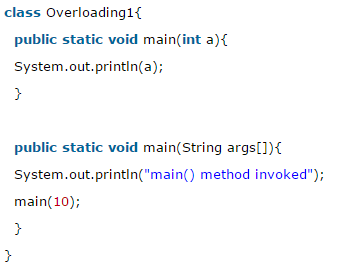
### Q) Why Method Overloading is not possible by changing the return type of method?

A) In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity.



### Q) Can we overload main() method?

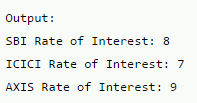
A) Yes, by method overloading. You can have any number of main methods in a class by method overloading.

**Method Overriding (Run time polymorphism)**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

1. Method must have same name as in the parent class
2. Method must have same parameter as in the parent class.
3. Must be IS-A relationship (inheritance).

### Q) Can we override static method?

### A) No, static method cannot be overridden.

### Q) Why we cannot override static method?

A) Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area. **main() is also static method so cannot be overridden.**

#### **Abstraction**

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use **abstract class** and **interface** to achieve abstraction.

1. Abstract class (0 to 100%)
2. Interface (100%)

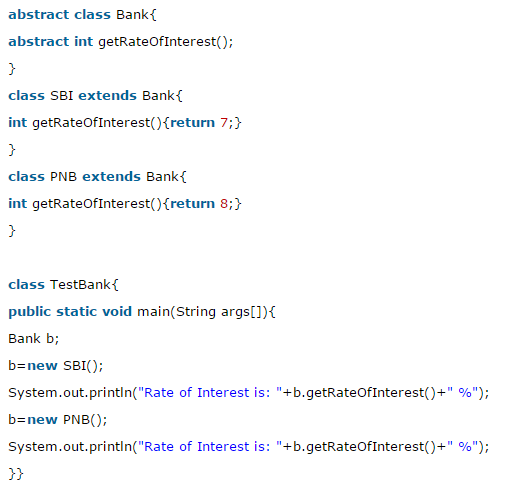
### **Abstract class in Java**

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

A method that is declared as abstract and does not have implementation is known as **abstract method**.

**Rule:** If there is any abstract method in a class, that class must be abstract.

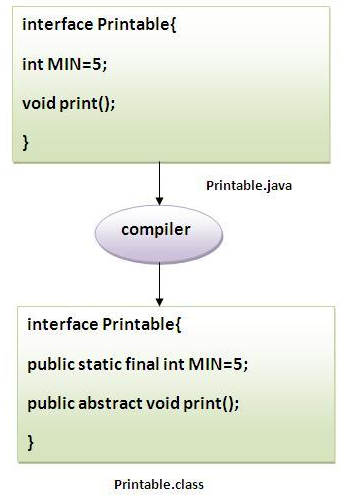
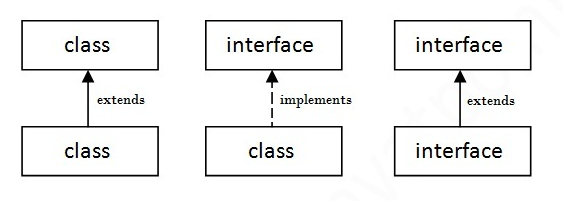
**Rule:** If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

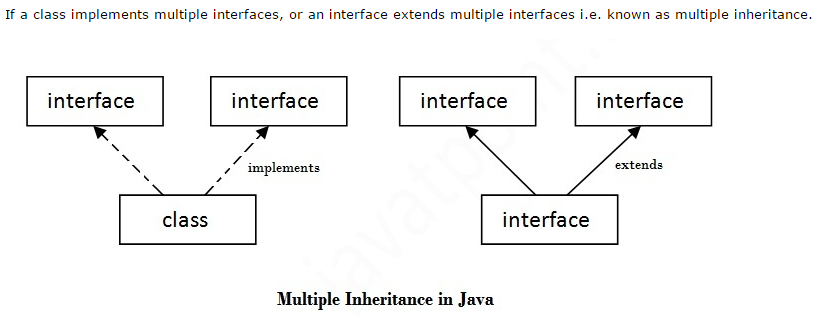
# Interface in Java

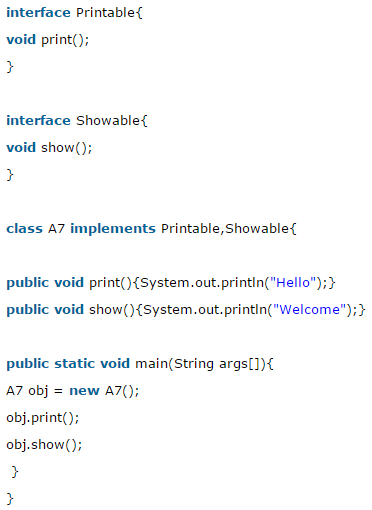
The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java. It cannot be instantiated just like abstract class.

Note:- Interface fields are public, static and final by default, and methods are public and abstract.

## **Multiple inheritance in Java by interface**



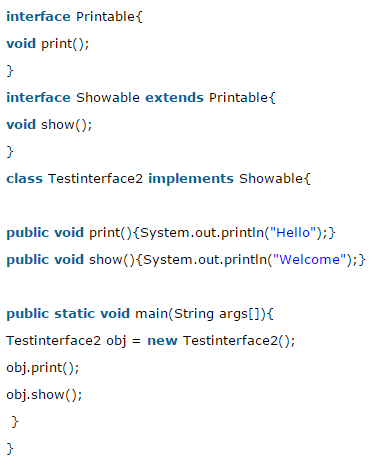
 

### Q) Multiple inheritance is not supported through class in java but it is possible by interface, why?

A) Multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

## **Interface inheritance**

A class implements interface but one interface extends another interface.

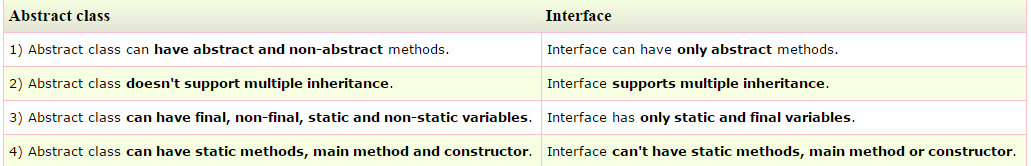
 

## **What is marker or tagged interface?**

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

# Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.



#### **Encapsulation**

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here. By providing only setter or getter method, you can make the class **read-only or write-only**. It provides you the **control over the data**. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

## **Advantage of OOPs over Procedure-oriented programming language**

|  |
| --- |
| 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. |
| 2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere. |

## **What is difference between object-oriented programming language and object-based programming language?**

Object based programming language follows all the features of OOPs except Inheritance. **JavaScript** and **VBScript** are examples of object based programming languages.

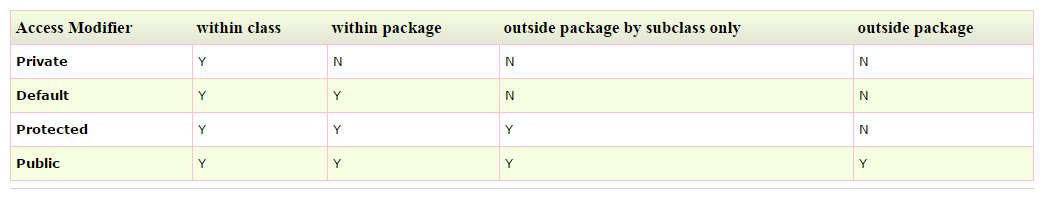
# Access Modifiers in java

# There are two types of modifiers in java: access modifiers and non-access modifiers.

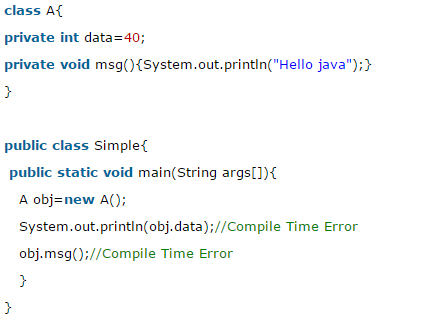
There are 4 types of java access modifiers:

1. **Private -** The private access modifier is accessible only within class.
2. **Default -** If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within 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.
4. **Public -** The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc.



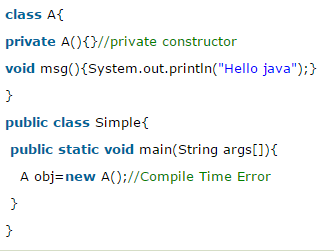
### **1) private access modifier**



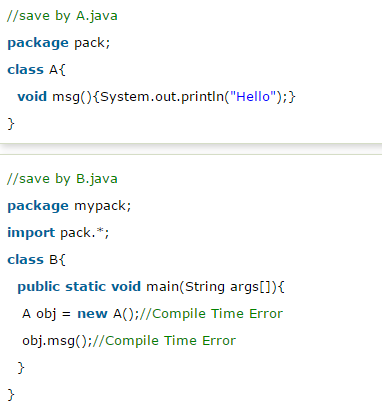
We are accessing these private members from outside the class, so there is compile time error.

### 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:



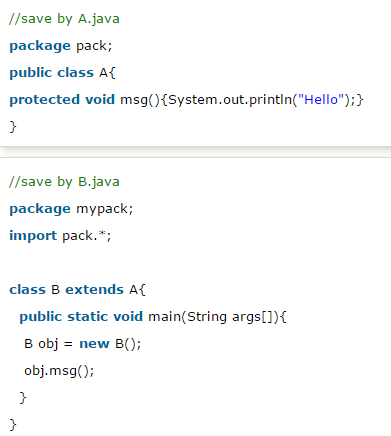
### **2) default access modifier**



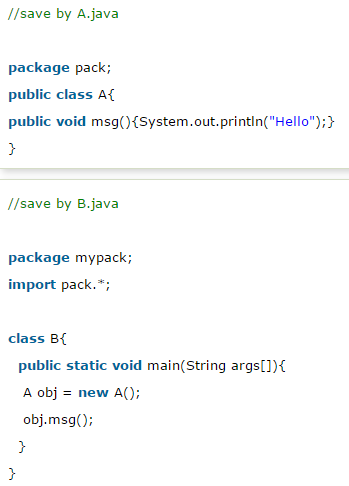
In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

### **3) protected 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.

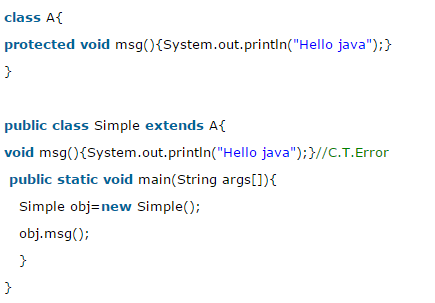
 

### **4) public access modifier**

### **Java access modifiers with method overriding**

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



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

# Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

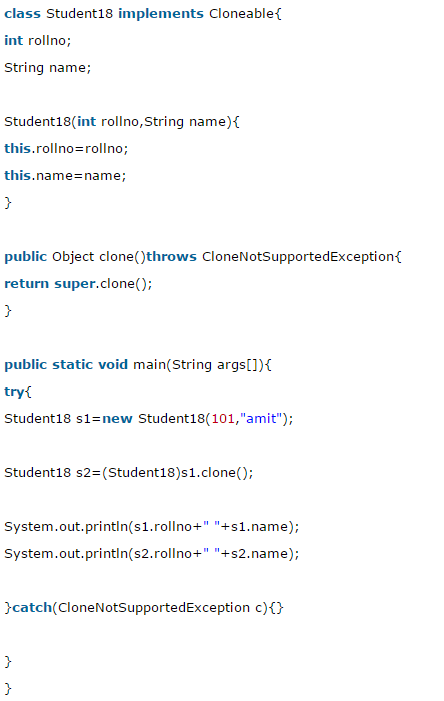
The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

# Object Cloning in Java

The **object cloning** is a way to create exact copy of an object. For this purpose, clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

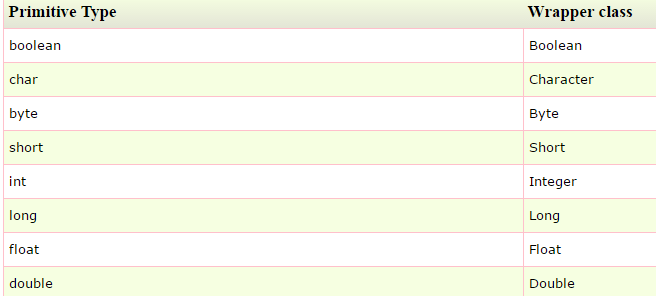
Advantage of using Object cloning to have less processing task.

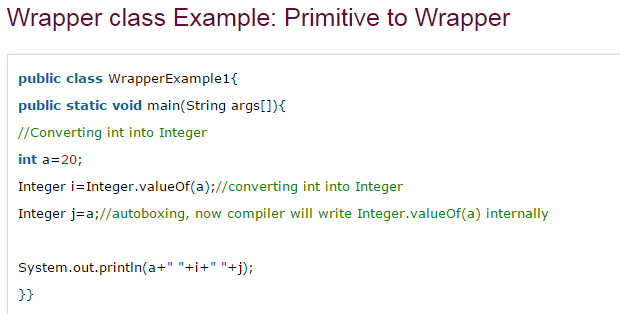
 

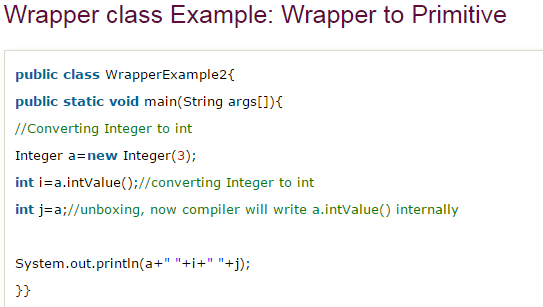
# Wrapper class in Java

**Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.

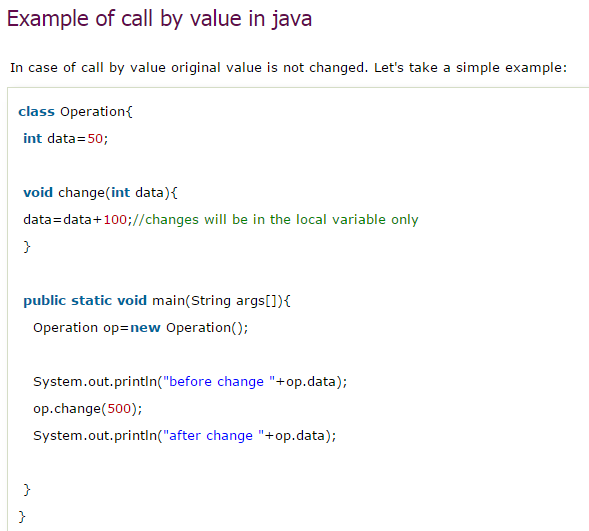


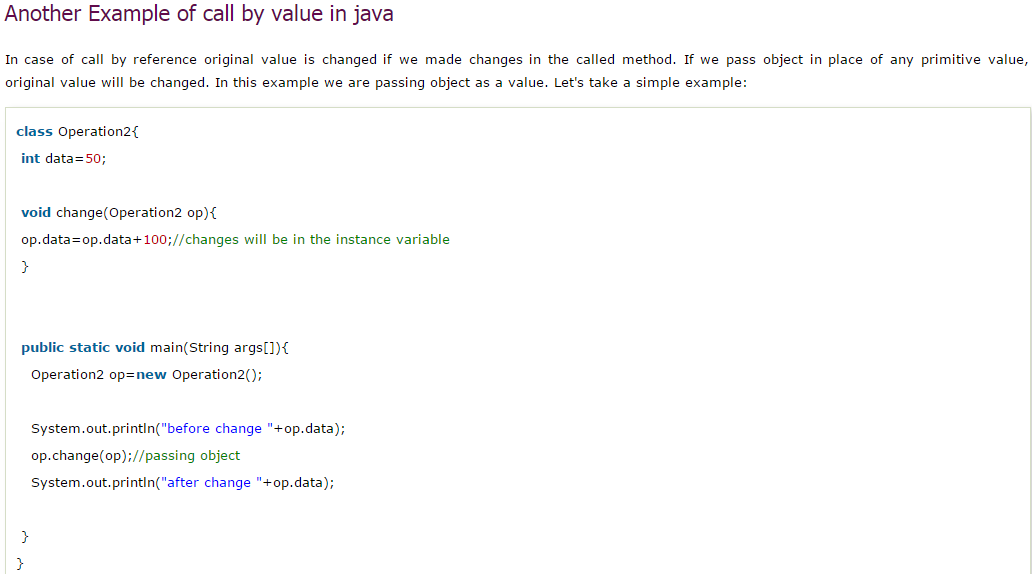
 

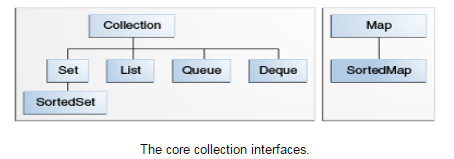
# Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value.

Collection Framework in Java



All the core collection interfaces are generic.

**public interface Collection<E>**

The <E> syntax tells you that the interface is generic. When you declare a Collection instance you can and should specify the type of object contained in the collection. Specifying the type allows the compiler to verify (at compile-time) that the type of object you put into the collection is correct, thus reducing errors at runtime.

**Collection —** the root of the collection hierarchy. A collection represents a group of objects known as its elements.

**Set —** a collection that cannot contain duplicate elements.

**List —** an ordered collection (sometimes called a sequence). Lists can contain duplicate elements.

**Queue —** a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations. Queues typically, but do not necessarily, order elements in a FIFO (first-in, first-out) manner. In a FIFO queue, all new elements are inserted at the tail of the queue and the head of the queue is the element that would be removed.

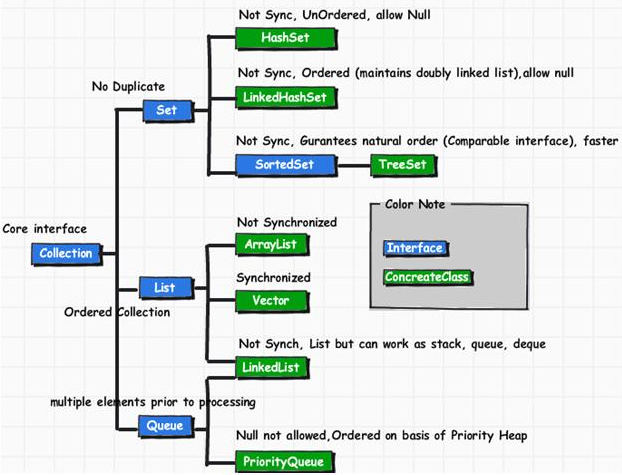
**Deque —** a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Deque provides additional insertion, extraction, and inspection operations. Deques can be used both as FIFO (first-in, first-out) and LIFO (last-in, first-out). In a deque all new elements can be inserted, retrieved and removed at both ends.

**Map —** an object that maps keys to values. A Map cannot contain duplicate keys; each key can map to at most one value.

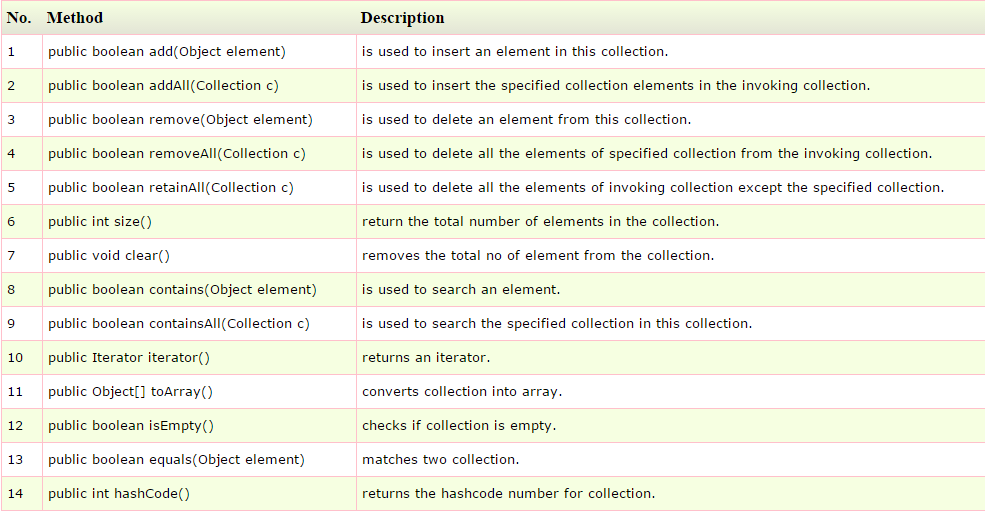
The last two core collection interfaces are merely sorted versions of Set and Map:

**SortedSet —** a Set that maintains its elements in ascending order.

**SortedMap —** a Map that maintains its mappings in ascending key order.



### **Methods of Collection interface**



### **Iterator interface**

Iterator interface provides the facility of iterating the elements in forward direction only. There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator. It is rarely used.

**List interface**

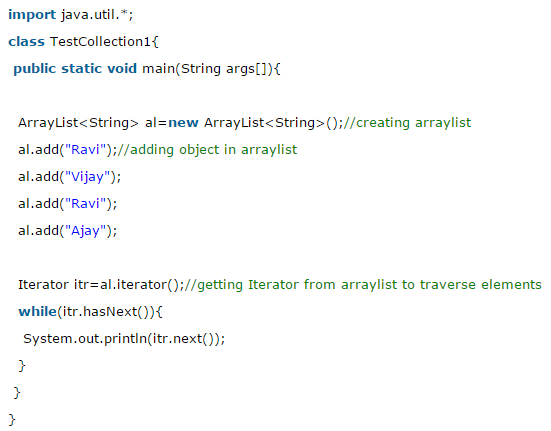
The Java platform contains two general-purpose List implementations. **ArrayList**, which is usually the better-performing implementation, and **LinkedList** which offers better performance under certain circumstances.

### **Commonly used methods of List Interface:**

1. public void add(int index,Object element);
2. public boolean addAll(int index,Collection c);
3. public object get(int Index position);
4. public object set(int index,Object element);
5. public object remove(int index);
6. public ListIterator listIterator();
7. public ListIterator listIterator(int i);

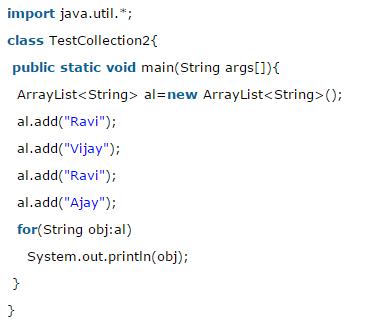
# Java ArrayList class

* Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Output- 

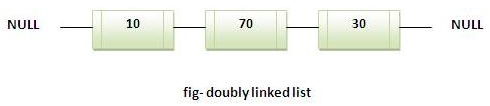
There are two ways to iterate through collections.

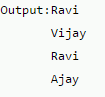
1. By Iterator interface.
2. By for-each loop.

 Output- 

# Java LinkedList class

* Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.
* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
* Java LinkedList class can be used as list, stack or queue.

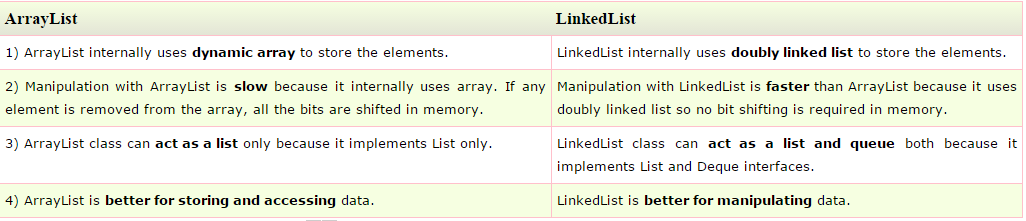


# Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

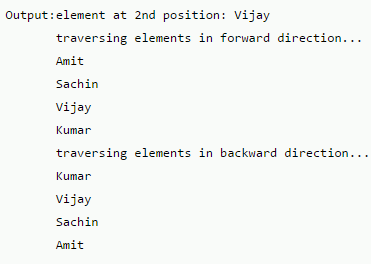
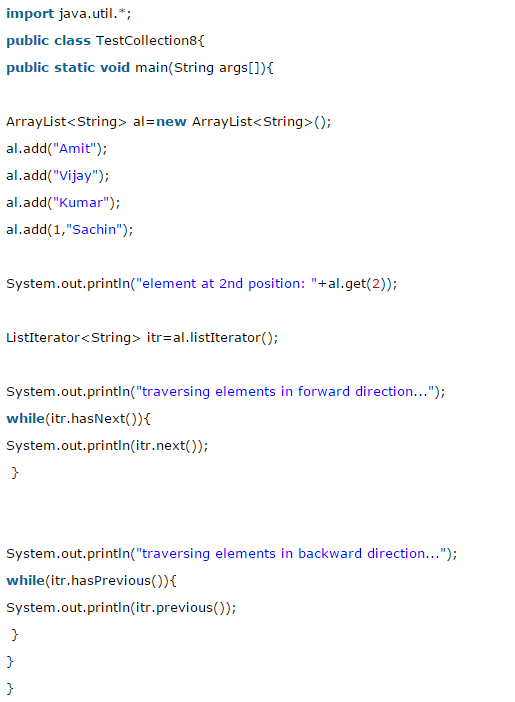


## **Java ListIterator Interface**

ListIterator Interface is used to traverse the element in backward and forward direction.

### **Commonly used methods of ListIterator Interface:**

1. public boolean hasNext();
2. public Object next();
3. public boolean hasPrevious();
4. public Object previous();

**Set interface**

A Set is a collection that cannot contain duplicate elements. The Java platform contains three general-purpose Set implementations:

**HashSet -** which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration.

**TreeSet -** which stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashSet.

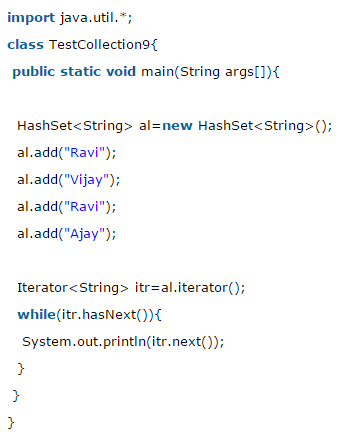
**LinkedHashSet -** which is implemented as a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set (insertion-order).

## **Difference between List and Set:**

List can contain duplicate elements whereas Set contains unique elements only.

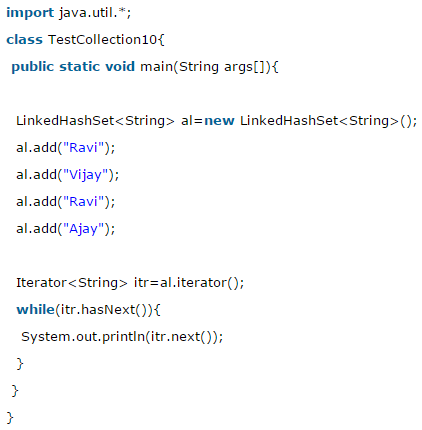
# Java HashSet class

* uses hashtable to store the elements.It extends AbstractSet class and implements Set interface.
* contains unique elements only.

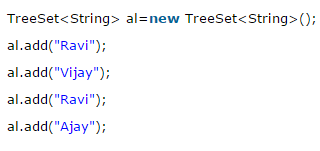
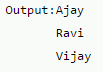
# Java LinkedHashSet class

* contains unique elements only like HashSet. It extends HashSet class and implements Set interface.
* maintains insertion order.

 Output - 

# Java TreeSet class

* contains unique elements only like HashSet. The TreeSet class implements NavigableSet interface that extends the SortedSet interface.
* maintains ascending order.

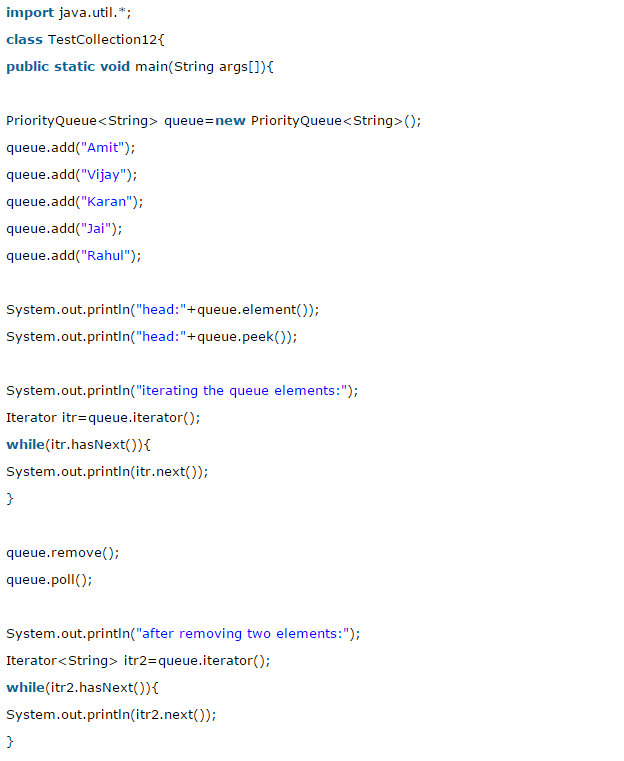
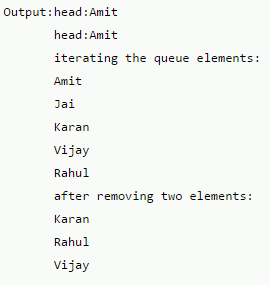
 

# Java Queue Interface

The Queue interface basically orders the element in FIFO(First In First Out)manner.

## **PriorityQueue class**

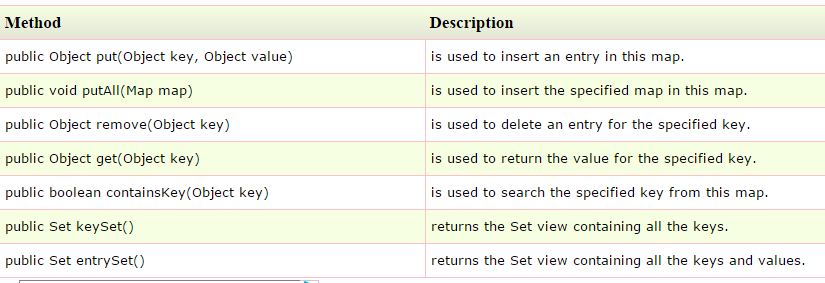
The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner.

# Java Map Interface

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

Map is useful if you have to search, update or delete elements on the basis of key.



## **Entry Interface**

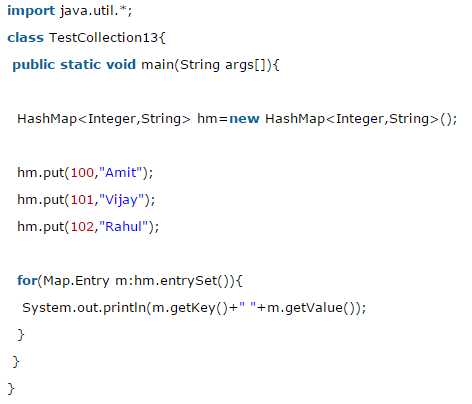
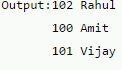
Entry is the subinterface of Map. So we will be accessed it by Map.Entry name. It provides methods to get key and value.

### **Methods of Map.Entry interface**

1. **public Object getKey():** is used to obtain key.
2. **public Object getValue():**is used to obtain value.

# Java HashMap class

* A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It maintains no order.

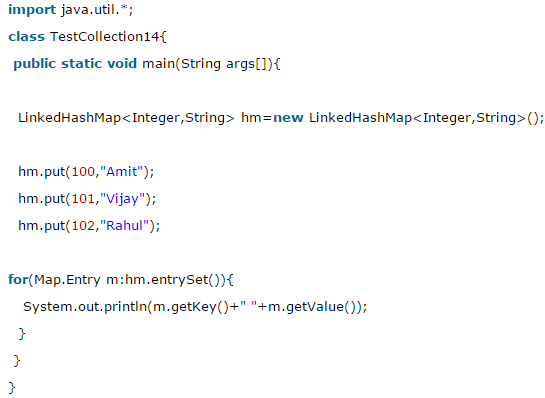
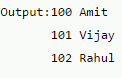
 

### **What is difference between HashSet and HashMap?**

HashSet contains only values whereas HashMap contains entry(key and value).

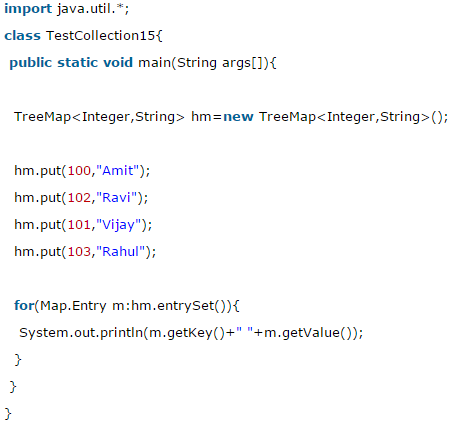
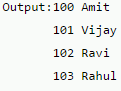
# Java LinkedHashMap class

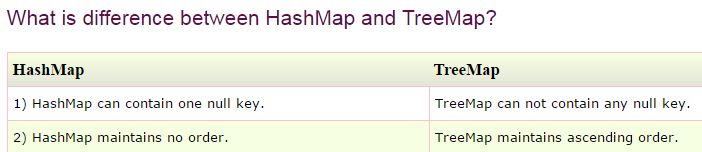
* A LinkedHashMap contains values based on the key. It implements the Map interface and extends HashMap class.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is same as HashMap instead maintains insertion order.

# Java TreeMap class

* A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* It contains only unique elements.
* It cannot have null key but can have multiple null values.
* It is same as HashMap instead maintains ascending order.

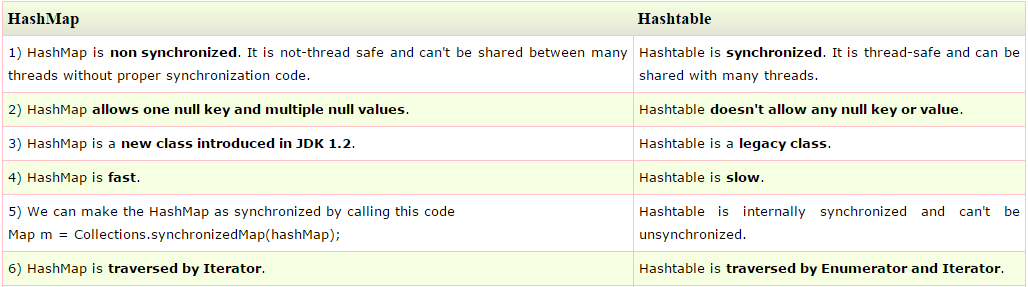
 



# Difference between HashMap and Hashtable

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

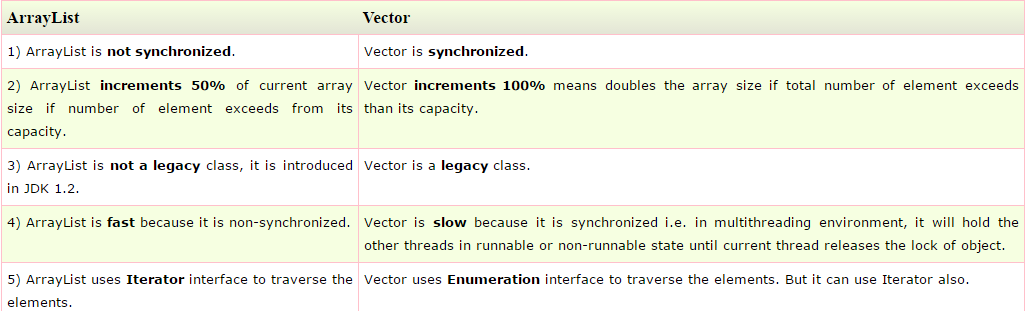
But there are many differences between HashMap and Hashtable classes that are given below.



# Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.



# Sorting in Collection

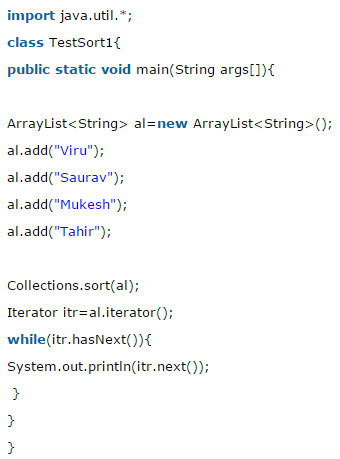
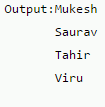
We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

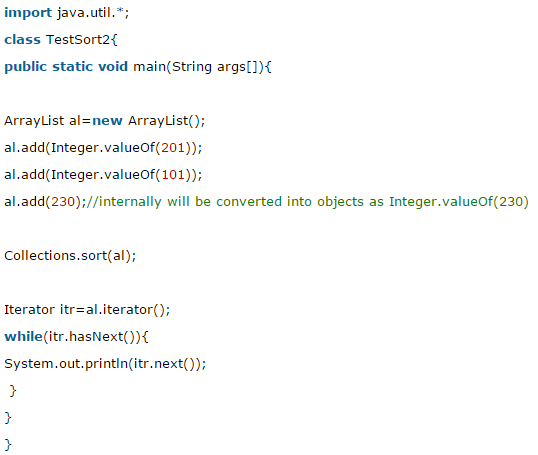
**public void sort(List list):** is used to sort the elements of List. List elements must be of Comparable type.

#### Note: String class and Wrapper classes implements the Comparable interface.So if you store the objects of string or wrapper classes, it will be Comparable.

### **Example of Sorting the elements of List that contains string objects**

### **Example of Sorting the elements of List that contains Wrapper class objects**

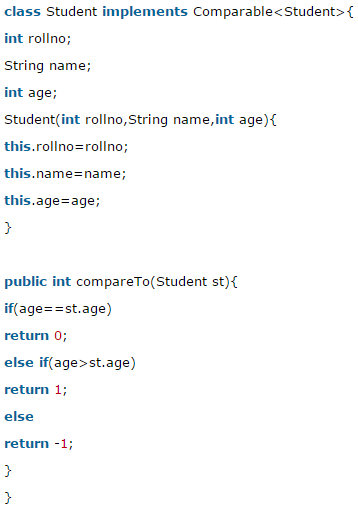
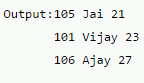
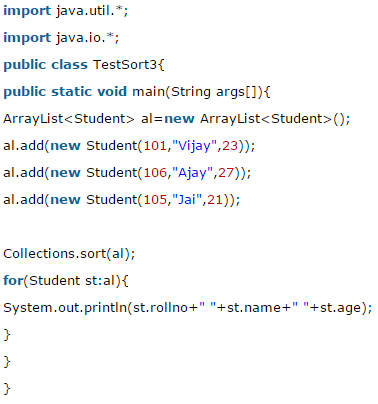
 

# Java Comparable interface

Java Comparable interface is used to order the objects of user-defined class.This interface is found in java.lang package and contains only one method named compareTo(Object). It provide single sorting sequence only i.e. you can sort the elements on based on single data member only. For example it may be rollno, name, age or anything else.

## **Java Comparable Example**

Let's see the example of Comparable interface that sorts the list elements on the basis of age.

# Java Comparator interface

**Java Comparator interface** is used to order the objects of user-defined class.

This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

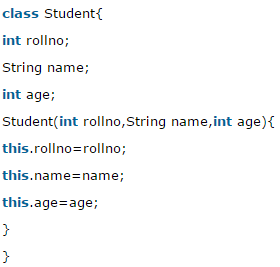
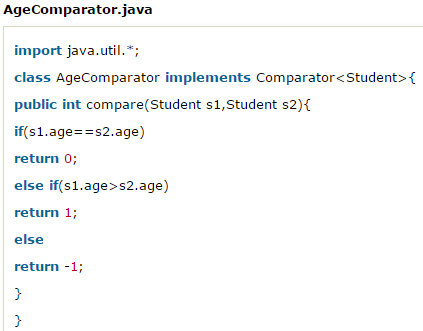
It provides multiple sorting sequence i.e. you can sort the elements on the basis of any data member, for example rollno, name, age or anything else.

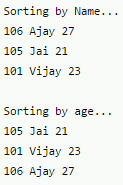
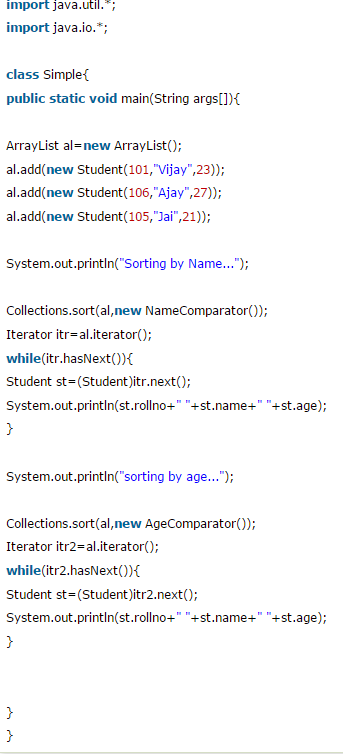
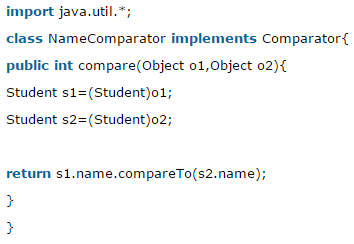
int compare(Object obj1, Object obj2)

obj1 and obj2 are the objects to be compared. This method returns zero if the objects are equal. It returns a positive value if obj1 is greater than obj2. Otherwise, a negative value is returned.

boolean equals(Object obj)

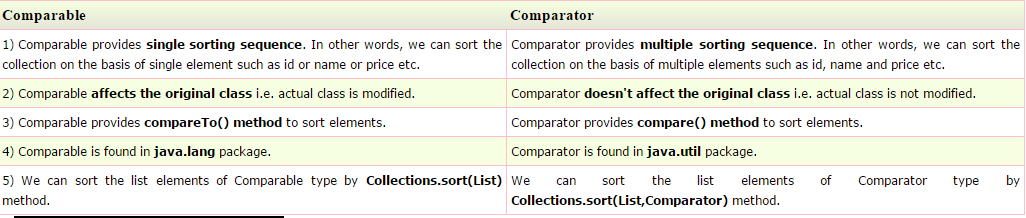
obj is the object to be tested for equality. The method returns true if obj and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.



# Difference between Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.



# Properties class in Java

The **properties** object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.

### **Advantage of properties file**

**Recompilation is not required, if information is changed from properties file:** If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.

