MAP INTERFACE

***INTRODUCTION ::***

* The Map interface maps unique keys to values. A key is an object that you use to retrieve a value at a later date.
* Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key.
* Several methods throw a NoSuchElementException when no items exist in the invoking map.
* A **ClassCastException** is thrown when an object is incompatible with the elements in a map.
* A **NullPointerException** is thrown if an attempt is made to use a null object and null is not allowed in the map.
* An **UnsupportedOperationException** is thrown when an attempt is made to change an unmodifiable map.
* The interface methods can be broken down into three sets of operations: altering, querying and providing alternative views.
* The alteration operation allows you to add and remove key-value pairs from the map.
* Both the key and value can be null. However you should not add a Map to itself as a key or value.  
   ***Object put(Object key, Object value)   
   Object remove(Object key)   
   void putAll(Map t)   
   void clear()***
* The query operations allow you to check on the contents of the map  
   ***Object get(Object key)   
   boolean containsKey(Object key)   
   boolean containsValue(Object value)   
   int size()  
   boolean isEmpty()***
* The set methods allow you to work with the group of keys or values as a collection  
   ***Set keySet()   
   Collection values()   
   Set entrySet()***

import java.util.\*;

public class CollectionsDemo {

public static void main(String[] args) {

Map m1 = new HashMap();

m1.put(8,"Zara");

m1.put(31,"Mahnaz");

m1.put(12,"Ayan");

m1.put(31,"Daisy");

m1.put(16,null);

m1.put(null,null);

m1.put(8,"ram");

m1.put("8","raju");

System.out.println();

System.out.println(" Map Elements");

System.out.print("\t" + m1);

}

}

***TreeMap:***

* TreeMap elements are in sorted ordered in term of keys.
* TreeMap doesn’t allow null keys.
* TreeMap allows only similar types of keys

import java.util.\*;

public class CollectionsDemo

{

public static void main(String[] args)

{

Map m1 = new TreeMap();

m1.put(8,"Zara");

m1.put(31,"Mahnaz");

m1.put(12,"Ayan");

m1.put(31,"Daisy");

m1.put(16,null);

// m1.put(null,null);

m1.put(8,"ram");

// m1.put("8","raju");

System.out.println();

System.out.println(" Map Elements");

System.out.print("\t" + m1);

}

}

import java.util.\*;

class MapDemo

{

public static void main(String args[])

{

String days[]={"Sunday", "Monday", "Tuesday", "Wednesnday","Thursday", "Friday",”Saturday"};

Map map = new HashMap();

try

{

for(int i=0; i<7; i++)

{

map.put(i, days[i]);

}

TreeMap tMap=new TreeMap(map);

//Rerieving all keys

System.out.println("Keys of tree map: " + tMap.keySet());

//Rerieving all values

System.out.println("Values of tree map: " + tMap.values());

//Rerieving the First key and its value

System.out.println("First key: " + tMap.firstKey() +" Value: " + tMap.get(tMap.firstKey()) + "\n");

//Removing the first key and value

System.out.println("Removing first data: " + tMap.remove(tMap.firstKey()));

System.out.println("Now the tree map Keys: " + tMap.keySet());

System.out.println("Now the tree map contain: " + tMap.values() + "\n");

//Rerieving the Last key and value

System.out.println("Last key: " + tMap.lastKey() + " Value: " + tMap.get(tMap.lastKey()) + "\n");

//Removing the last key and value

System.out.println("Removing last data: " + tMap.remove(tMap.lastKey()));

System.out.println("Now the tree map Keys: " + tMap.keySet());

System.out.println("Now the tree map contain: " + tMap.values());

}

catch(Exception e){}

}

}

***Difference between HashMap and Hashtable in Java***

* Both HashMap and Hashtable implements Map interface but there are some significant difference between them which is important to remember before deciding whether to use HashMap or Hashtable in Java.
* Some of them are [thread-safety](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html), [synchronization](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html)and speed. here are those differences :
* The HashMap class is roughly equivalent to Hashtable, except that it is non synchronized and permits nulls. (HashMap allows null values as key and value whereas [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html)doesn't allow nulls).
* One of the major differences between **HashMap and Hashtable** is that HashMap is [non synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap cannot be shared between multiple threads without proper synchronization.
* Another significant difference between [HashMap vs. Hashtable](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html) is that Iterator in the HashMap is  a [fail-fast iterator](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally  by adding or removing any element except Iterator's own remove() method.
* One more notable *difference between Hashtable and HashMap* is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don't need synchronization and HashMap is only used by one thread, it outperforms Hashtable in Java.

***Tree map Vs Hashmap:***

* Both TreeMap & HashMap are not synchronized.
* To make it synchronized we have to explicitly call Collections.synchronizedMap( mapName ) .
* Both supports “fail-fast” iterators. Both of them doesn’t support duplicate keys.

***HashMap***

* HashMap allows null as both keys and values.
* HashMap is useful when we need to access the map without cosidering how they are added to the map (means, unordered lookup of values using their keys).
* HashMap is synchronized while it is being looked up.
* HashMap doesn’t allow duplicated entries.
* The performance of HashMap is based on two optional parameter which we can specify during the creation of the HashMap one is Initial capacity and second one is load factor.
* Initial capacity is the bucket size assigned to a HashMap during its creation.
* Load factor decides when the HashMap needs to be expanded. If the load factor is 0.75, the size will be increased when the current size of the map crosses 75% of its capacity.

***TreeMap***

* The basic difference between HashMap & TreeMap is that, in a TreeMap the elements are stored in a tree .
* TreeMap allows us to retrieve the elements in some sorted order defined by the user.
* So we can say that TreeMap is slower than HashMap.

***Difference between LinkedHashMap and HashMap in Java***

* HashMap and LinkedHashMap are two most common used Map implementation in Java and main difference between HashMap and LinkedHashMap is that LinkedHashMap maintain insertion order of keys, Order in which keys are inserted in to LinkedHashMap.
* On the other hand HashMap doesn't maintain any order or keys or values.

In terms of Performance there is not much difference between HashMap and LinkedHashMap but yes LinkedHashMap has more memory foot print than HashMap to maintain doubly LinkedList which it uses to keep track of insertion order of keys.

**TreeMap :**

**import** java.util.\*;

**public** **class** Treemapdemo

{

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

{

Map m=**new** HashMap();

m.put("AAA",**new** Integer(111));

System.*out*.println("Map object data is:"+m);

TreeMap obj=**new** TreeMap();

obj.put("S",**new** Integer(1));

obj.put("T",**new** Integer(2));

obj.put("U",**new** Integer(3));

obj.put("V",**new** Integer(4));

System.*out*.println(" TreeMap object data is"+obj);

obj.putAll(m);

System.*out*.println(" TreeMap object data after adding an object"+obj);

System.*out*.println(" First obj is "+obj.firstKey());

System.*out*.println(" Last obj is "+obj.lastKey());

System.*out*.println(" postion of V is"+obj.ceilingEntry("V"));

System.*out*.println(" object contains 4 value :"+obj.containsValue(4));

//De assembling

Set s=obj.entrySet();

Iterator itr=s.iterator();

**while**(itr.hasNext())

{

Map.Entry me=(Map.Entry)itr.next();

Object kobj=me.getKey();

Object vobj=me.getValue();

System.*out*.println("Kobj-------->"+kobj);

System.*out*.println("vobj--------->"+vobj);

}

obj.clear();

System.*out*.println(" TreeMap object data after clear an object"+obj);

}

}

**HashTable demo:**

import java.util.Enumeration;

import java.util.Hashtable;

import java.util.Map;

public class Hashtabledemo

{

public static void main(String[] args)

{

Hashtable ht=new Hashtable();

ht.put("Nag","java");

ht.put("xxx", "Testing");

ht.put("yyy","MS.net");

System.out.println("Values are:"+"\n");

System.out.println(ht.values());

//De assembling data

Enumeration en=ht.elements();

while(en.hasMoreElements())

{

System.out.println(en.nextElement());

}

}

}

LEGACY COLLECTION OF CLASSES::

**Creating Hashtable is nothing but creating an object of Hashtable class.(it is a legacy CFW class)**

**Legacy(old) collection Framework:**

When SUN Micro Systems has developed java, collection framework was known as data structures. Data structures in java were unable to meet industry requirements at that time. Hence data structures of java was reengineered and they have added ‘n’ number of classes and interfaces and in later stages the data structures of java is known as new collection framework.

**Interfaces:**

We have only one interface, namely **java.util.Enumeration**. This interface is used for extracting the data from legacy collection framework classes

**Classes:**

As a part of legacy collection framework we have the following essential classes: **Vector, Stack, Dictionary, Hashtable and properties**. Here, Vector and Stack belongs to one dimensional classes whereas Dictionary, Hashtable and Properties belongs to two dimensional classes.

* **What is the difference between normal collection framework and legacy collection framework?**

**Answer:** All the classes in the normal collection framework are by default belongs to non synchronized classes whereas all classes in legacy collection framework are by default belongs to synchronized classes

**Vector:**

Its functionality is exactly similar to ArrayList but Vector class belongs to synchronized,whereas ArrayList belongs to non-synchronized class.

Creating a Vector is nothing but creating an object of java.util.Vector class.

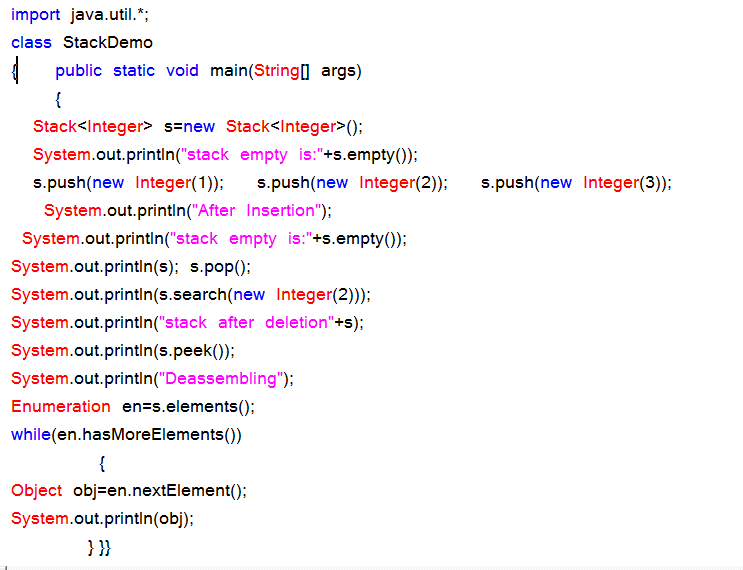
Null insertion is possible here

**Stack:** Stack is the sub-class of Vector class. The basic working principal of Stack is Last In First Out.

**Stack API:**

**Constructors:**

1.Stack (); 2.Stack (int size);

****

**Instance methods:**

public boolean empty ();

public void push (Object);

public Object pop ();

public Object peek ();

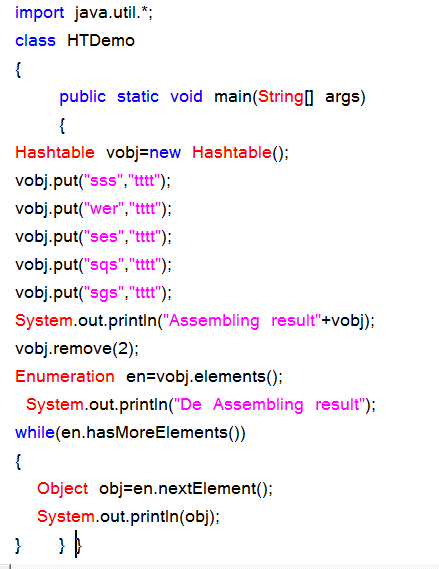
public int search(Object);

**Dictionary:**

Dictionary is an abstract class, whose object allows to retrieve to store the data in the form of (key, value) pair. An object of Dictionary never allows duplicate values as key objects and null values.

**Hashtable:**

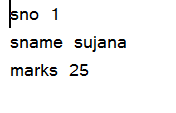
Hashtable is the concrete sub-class of Dictionary and where object allows us to store in the form of (key, value) pair. Hashtable object organizes its data by following hashing mechanism. We cannot determine in which order the Hashtable displays its data.



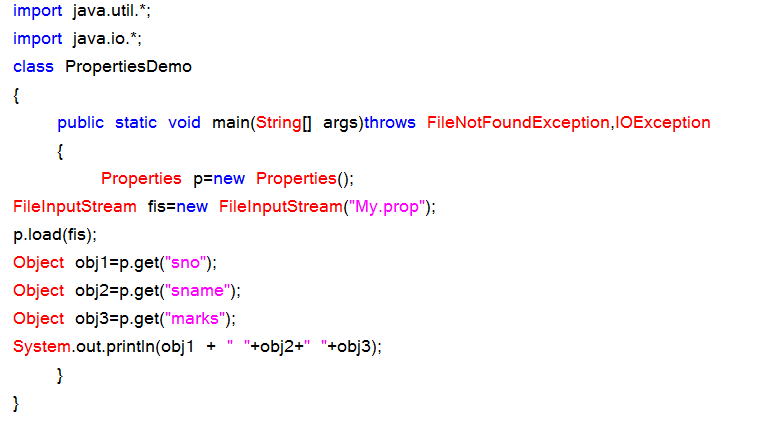
**Properties class:**

Properties is the sub-class of Hashtable class. Properties class object is used for reading of maintaining system environmental variables and reading the data from resource data file or properties file.

**Create a Properties file with My.prop or My.rbf:**



**Properties program:**



**Properties API:**

**Constructor:**

Properties ();

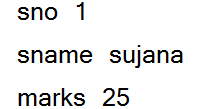
**Instance Methods:**

public void setProperty (Object kobj, Object vobj);

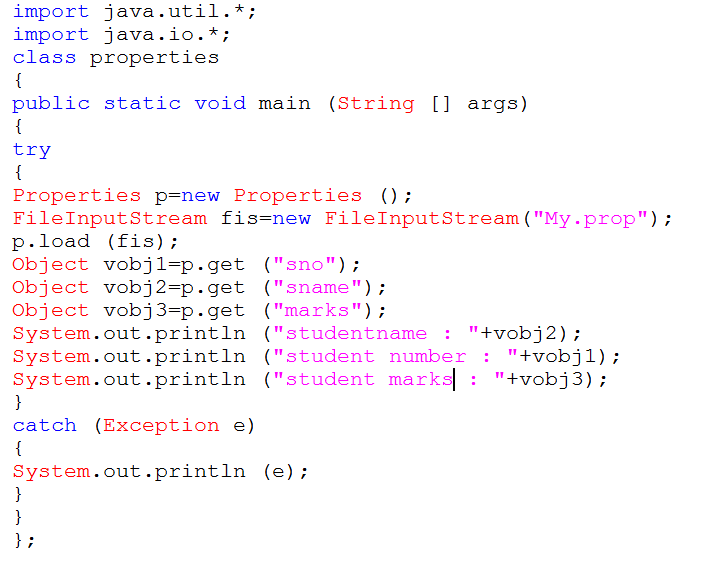
public Object getProperty (Object kobj);

public void load (InputStream);

**Preparation of ResourceBundleFile / Properties file:**



Program on properties file:



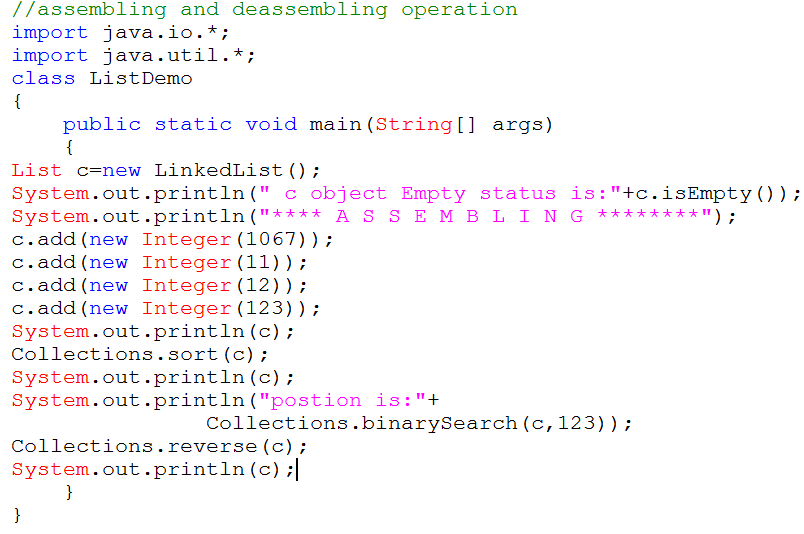
**Collections Class:**

Collections class is a utility class having static methods for doing operations on objects of classes which implement the Collection interface. For example, Collections has methods for finding the max element in a Collection.

Collections is merely an utility method class for doing certain operations, for example adding thread safety to your ArrayList instance by doing this:

List list = Collections.synchronizedList(new Arraylist());

**Source code:**



**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 only single sorting sequence i.e. you can sort the elements on based on single datamember only.For instance it may be either rollno,name,age or anything else.

**Syntax:**

public int compareTo(Object obj): is used to compare the current object with the specified object.

We can sort the elements of:

* String objects
* Wrapper class objects
* User-defined class objects

This interface present in java.lang package & contains only one method that is "compareTo()".

**Method:** public int compareTo(Object obj)

obj1.compareTo(Obj2 ) 🡪 1

1) if it returns -ve value obj1 has to come before obj2

2)if it returns +ve value obj1 has to come after obj2

3)if it returns 0 value obj1 , obj2 are equal

**Example:**

import java.util.\*;

class Test

{

public static void main(String args[])

{

System.out.println("A".compareTo("Z"));

System.out.println("Z".compareTo("K"));

System.out.println("A".compareTo("A"));

}

}

**Ex:**

import java.util.\*;

class Test1

{

public static void main(String args[])

{

TreeSet t=new TreeSet();

t.add("Z");

t.add("K");

t.add("D");

t.add("M");

t.add("D");

System.out.println(t); //**[ D , K , M , Z ]**

}

}

**Program:**

class Student implements Comparable{

int rollno;

String name;

int age;

Student(int rollno,String name,int age){

this.rollno=rollno;

this.name=name;

this.age=age;

}

public int compareTo(Object obj){

Student st=(Student)obj;

if(age==st.age)

return 0;

else if(age>st.age)

return 1;

else

return -1;

}

}

**Simple.java:-**

import java.util.\*;

import java.io.\*;

class Simple

{

public static void main(String args[])

{

ArrayList al=new ArrayList();

al.add(new Student(101,"Vijay",23));

al.add(new Student(106,"Ajay",27));

al.add(new Student(105,"sunil",21));

Collections.sort(al);

Iterator itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+""+st.name+""+st.age);

}

}

}

**Public Comparator comparator():-**

It returns Comparator object describes underlying Sorting technique.if we used default natural sorting order then we will get null.

**Note:**

1) The default natural sorting order for the numbers is ascending order

2) The default natural sorting order for characters & strings is

alphabetical order(Dictionary based order)

**Note:** If we are not satisfied with default natural sorting order.Then we can define

our own customized sorting by using comparator.

* Comparable meant for default Natural sorting order
* Comparator meant for customized sorting order

Collections class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use TreeSet. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

**Method of Collections class for sorting List elements**

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 user-defined class objects on age basis

**public void sort(List list,Comparator c):** is used to sort the elements of List by the given comparator.

Example of sorting the elements of List that contains user-defined class objects on the basis of age and name.In this example, we have created 4 java classes:

AgeComparator.java,NameComparator.java,Simple.java,Student.java

This class contains three fields rollno, name and age and a parameterized constructor.

**Student.java**

class Student{

int rollno;

String name;

int age;

Student(int rollno,String name,int age){

this.rollno=rollno;

this.name=name;

this.age=age;

}}

**AgeComparator.java:**

This class defines comparison logic based on the age. If age of first object is greater than the second, we are returning positive value, it can be any one such as 1, 2 , 10 etc. If age of first object is less than the second object, we are returning negative value, it can be any negative value and if age of both objects are equal, we are returning 0.

**Program:**

import java.util.\*;

class AgeComparator implements Comparator{

public int Compare(Object o1,Object o2){

Student s1=(Student)o1;

Student s2=(Student)o2;

if(s1.age==s2.age)

return 0;

else if(s1.age>s2.age)

return 1;

else

return -1;

}

}

**NameComparator.java**

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

**Program:**

import java.util.\*;

class NameComparator implements Comparator{

public int Compare(Object o1,Object o2){

Student s1=(Student)o1;

Student s2=(Student)o2;

return s1.name.compareTo(s2.name);

}

}

**Simple.java:**In this class, we are printing the objects values by sorting on the basis of name and age.

import java.util.\*;

import java.io.\*;

class Simple{

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add(new Student(101,"Vijay",23));

al.add(new Student(106,"Ajay",27));

al.add(new Student(105,"Jai",21));

System.out.println("Sorting by Name...");

Collections.sort(al,new NameComparator());

Iterator itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

System.out.println("sorting by age...");

Collections.sort(al,new AgeComparator());

Iterator itr2=al.iterator();

while(itr2.hasNext()){

Student st=(Student)itr2.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}

**Output**: Sorting by Name...

106 Ajay 27

105 Jai 21

101 Vijay 23

Sorting by age...

105 Jai 21

101 Vijay 23

106 Ajay 27