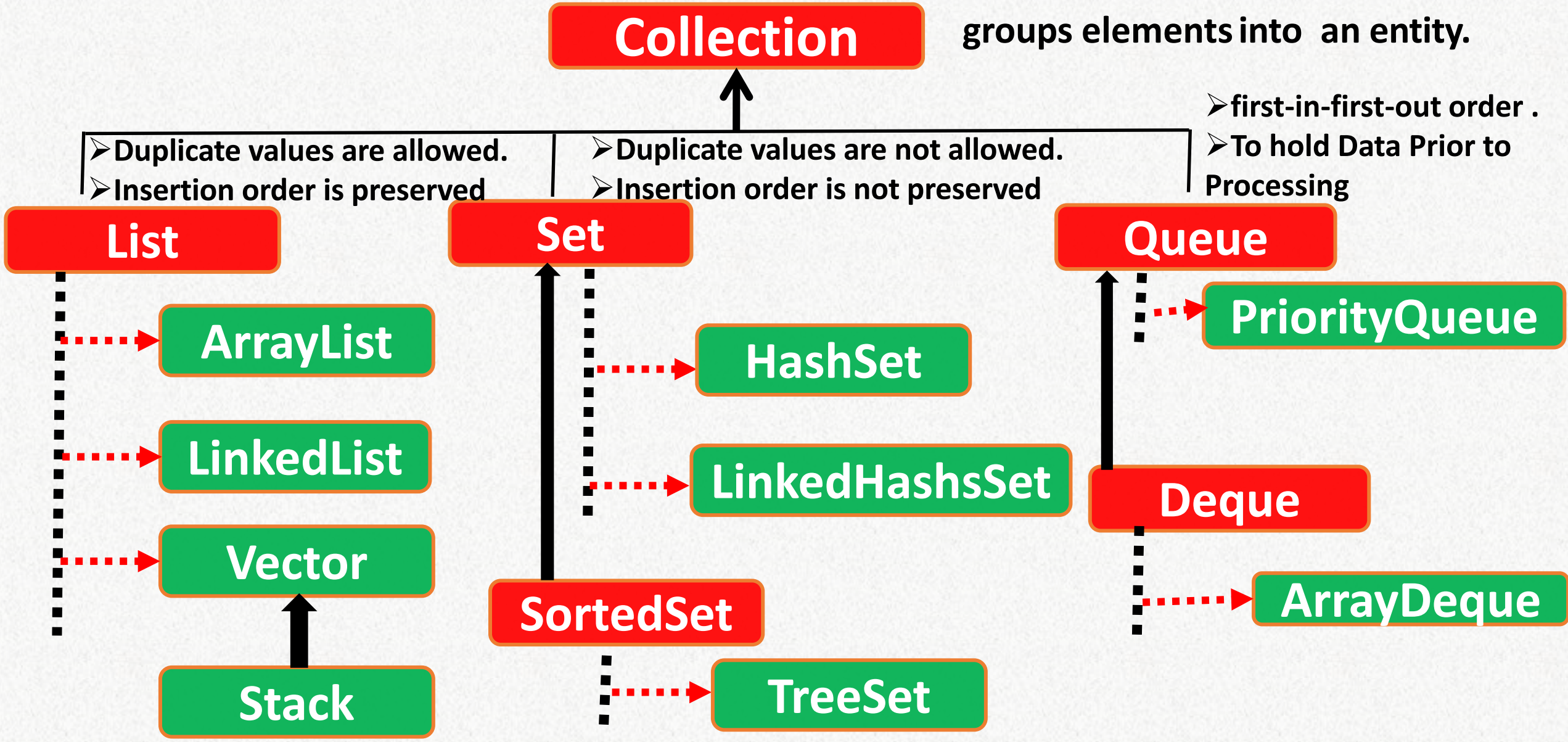
Collections Framework

- **Collection:** It is used to that groups elements into an entity.

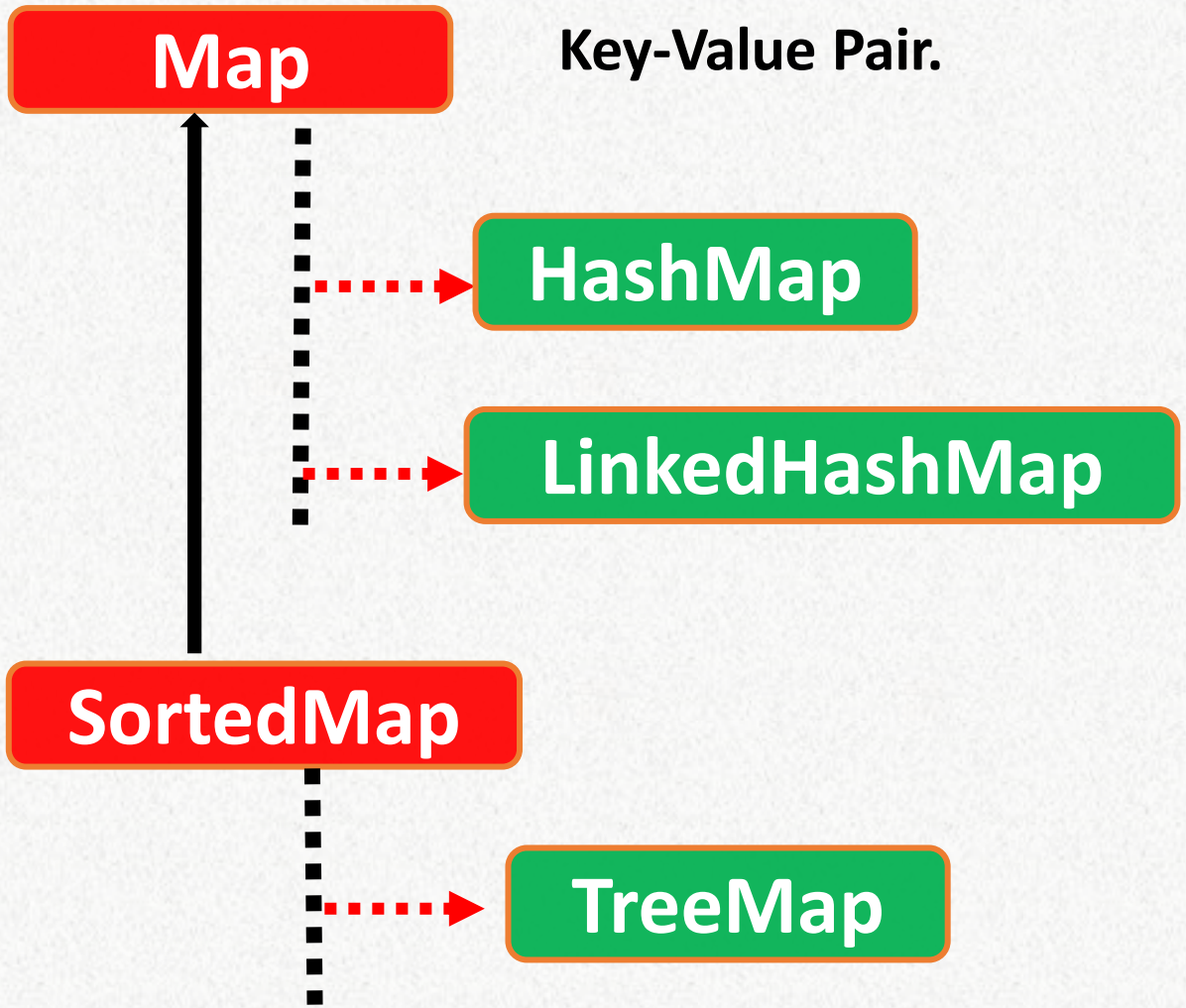
- Examples: list of bank accounts,
  set of students,
  group of telephone numbers.

- **Collections framework :** It provides a well-designed set of interfaces and classes for storing and manipulating groups of data as a single unit, a collection.”

Collections Framework



Collections Framework



Collections Framework

Collection: Represents group of individual objects.

Methods:

boolean add(object o)

boolean addAll(Collection c)

boolean remove(object o)

boolean removeAll(Collection c)

int size()

boolean isEmpty()

void clear()

boolean retainAll(Collection c)

Iterator iterator()

boolean contains(Object o)

boolean containsAll(Object o)

Object[] toArray()

Collections Framework

List: It is the child interface of Collection interface. It inhibits a list type data structure in which insertion ordered objects is preserved and duplicate values are allowed.

Methods:

boolean add(int index, object o)

boolean addAll(int index,Collection c)

object remove(int index,object o)

object get(int index)

object set(int index,Object o)

int indexOf(object o)

int LastIndexOf(object o)

Collections Framework

ArrayList : The ArrayList class implements the List interface.

- It uses a dynamic array to store the duplicate element of different data types.
- It preserves insertion order and is non-synchronized.
- The ArrayList class elements can be randomly accessed.

Constructors

ArrayList arr = new ArrayList();

ArrayList arr = new ArrayList(c);

ArrayList arr = new ArrayList(N);

Collections Framework

- **Iterator:** Iterator interface provides the facility of iterating the elements in a forward direction only.
- **public boolean hasNext():**It returns true if the iterator has more elements otherwise it returns false.
- **public Object next():**It returns the element and moves the cursor pointer to the next element.
- **public void remove():**It removes the last elements returned by the iterator. It is less used.

Collections Framework

```
import java.util.*;
class ArrayListIterator{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();//Creating arraylist
list.add(“Durga");//Adding object in arraylist
list.add(“Madhu");
list.add(“Naveen");
//Traversing list through Iterator
Iterator itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Collections Framework

LinkedList : The LinkedList class implements the List interface.

- It uses doubly linked list to storer elements
- It uses a dynamic array to store the duplicate element of different data types.
- It preserves insertion order and is non-synchronized.
- The LinkedList class elements can be randomly accessed.

Constructors

LinkedList ll = new LinkedList();

LinkedList ll = new LinkedList(c);

Methods

addFirst(Object o)

addLast(Object o)

removeFirst()

removeLast()

getFirst()

getLast()

Collections Framework

Vector : *It creates dynamic array* which can grow or shrink its size.

- It stores the duplicate element and preserves insertion order
- It is Thread safe as every method is synchronized.

Constructors

Vector v = new Vector();

Vector v = new Vector(intialcapacity);

Vector v = new Vector(initialcapacity,increament);

Vector v = new Vector(Collection c);

Methods

addElement(Object o)

removeElement(Object o)

removeElement At(intdex)

firstElement()

lastElement ()

Collections Framework

Stack : *It is child class of **Vector** class which is used to implement stack data structure. It is based on Last-In-First-Out (LIFO).*

- It stores the duplicate element and preserves insertion order

Constructors

Stack st = new Stack();

Methods

push(Object o)

pop(Object o)

peek(intdex)

isEmpty()

search ()

Collections Framework

Set: It is the child interface of Collection interface. It inhibits a list type data structure in which insertion ordered objects is not preserved and duplicate values are not allowed.

- It doesn’t contain any methods we have to use Collection Interface methods only.

Collections Framework

HashSet : HashSet extends AbstractSet and implements the Set interface.

- It creates a collection that uses a hash table for storage.
- Duplicates not allowed and insertion order not preservred.
- Null insertion is possible only once.
- A hash table stores information by using a mechanism called **hashing**.
- In hashing, the informational content of a key is used to determine a unique value, called its hash code.
- HashSet implements serializable and clonable Interfaces.

Constructors

HashSet v = new HashSet ();

HashSet v = new HashSet (intialcapacity);

HashSet v = new HashSet (initialcapacity,fillratio);

HashSet v = new HashSet (Collection c);

Collections Framework

SortedSet: It is the child interface of Set interface. It inhibits a list type data structure in which objects are stored in some sorted order and duplicate values are not allowed.

Methods

Object first()

Object last()

Comparator comparator()

SortedSet headSet(obj)

SortedSet tailSet(obj)

SortedSet subSet(obj,obj)

Collections Framework

TreeSet : TreeSet implements the SortedSet interface.

- It creates a collection that uses a Balanced tree for storage.
- Duplicates not allowed and insertion order not preservred.
- Null insertion is possible only once.
- Hetrogeneous Objects are not allowed.
- TreeSet implements serializable and clonable Interfaces.

Constructors

TreeSet v = new TreeSet ();

TreeSet v = new TreeSet (Comparator c);

TreeSet v = new TreeSet (SortedSet s);

TreeSet v = new TreeSet (Collection c);

Collections Framework

Queue: It is the child interface of Collection interface.

It inhibits a list type data structure in which objects are stored in **FIFO.**

Methods

Object add(): Adds the element and returns true upon success.

Object Offer(Object): Adds the element to the queue.

Object remove(): It removes the head retrieve element.

Object pool(): It retrieves and removes the head of queue, or returns null if queue is empty.

Object peek(): retrieves the head element.

Collections Framework

PriorityQueue : The PriorityQueue class provides the facility of using queue.

- Duplicates not allowed and insertion order not preservred.
- Null insertion is not possible.
- PriorityQueue implements serializable and clonable Interfaces.

Constructors

PriorityQueue v = new PriorityQueue ();

PriorityQueue v = new PriorityQueue (intial capacity);

PriorityQueue v = new PriorityQueue (intial capacity,Comparator c);

PriorityQueue v = new PriorityQueue (SortedSet s);

PriorityQueue v = new PriorityQueue (Collection c);

Collections Framework

```
import java.util.*;
class PriorityQueueDemo{
public static void main(String args[])
{
PriorityQueue<String> qu=new PriorityQueue<String>();
qu.add("Durga");
qu.add("Madhu");
qu.add("Naveen");
qu.add("RAmesh");
System.out.println("head:"+qu.element());
System.out.println("head:"+qu.peek());
System.out.println("the queueelements:");
Iterator itr=queue.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
queue.remove();
queue.poll();
System.out.println("afterremovingtwoelements:");
Iterator<String>itr2=queue.iterator();
while(itr2.hasNext()){
System.out.println(itr2.next());
}
}
}
```

Collections Framework

ArrayDeque : The ArrayDeque class provides the facility of using deque and resizable-array. It inherits AbstractCollection class and implements the Deque interface.

- Duplicates not allowed and insertion order not preservred.
- Null insertion is not possible.
- ArrayDeque implements serializable and clonable Interfaces.

Constructors

ArrayDeque ad = new ArrayDeque ();

ArrayDeque ad = new ArrayDeque (intial capacity);

ArrayDeque ad = new ArrayDeque (Collection c);

Methods

Object OfferFirst()

Object offerlast()

Object poolFirst()

Object poollast()

Collections Framework

```
import java.util.*;
public class ArrayDequeDemo{
public static void main(String[]args){
ArrayDeque<String> deque=new ArrayDeque<String>();
deque.offer("Durga");
deque.offer("Madhu");
deque.add("Naveen");
deque.offerFirst("Ramesh");
System.out.println("AfterofferFirstTraversal...");
for(String s:deque){
System.out.println(s);
}
deque.pollLast();
System.out.println("AfterpollLast()Traversal...");
for(String s:deque){
System.out.println(s);
}
}
}
```

Collections Framework

Map: It stores group of objects as key and value pair.

- It doesn't allows duplicate keys, but allows duplicate values.
- Each key can map to at most one value only.
- Each key-value pair is called Entry.

Entry: Each key-value pair is called Entry.

- Entry Interface defined in Map Interface.

Methods

Object put(Object key, Object value)

void putAll(Map map)

V get(Object key):remove()

V remove(Object key)

boolean remove(Object key, Object value)

Object containsKey(Object key)

Object containsValue(Object value)

Set valueSet()

Set keySet()

Set entrySet()

Collections Framework

HashMap : The HashMap class provides the facility of HashTable datastructure .

- Duplicate key not allowed and Values can be repeated.
- Null keys and values not allowed.
- ArrayDeque implements serializable and clonable Interfaces.

Constructors

HashMap hm=new HashMap();

HashMap hm=new HashMap(int capacity)

HashMap hm=new HashMap(int capacity, float fillratio):

HashMap hm=new HashMap(Map m)

Collections Framework

```
import java.util.*;
public class HashMapDemo {

        public static void main(String[] args) {
                Scanner read = new Scanner(System.in);
                HashMap<Integer,String> studinfo = new HashMap<Integer,String>();
                studinfo.put(501, "Durga");
                studinfo.put(510, "Madhu");
                studinfo.put(507, "Naveen");
                studinfo.put(512, "Ramesh");
                System.out.println("Student Information\n" + studinfo);
                System.out.println("Employee IDs : " + studinfo.keySet());
                 for(Map.Entry m:studinfo.entrySet())
                {
                 System.out.println(m.getKey()+" "+m.getValue());
                }
}       }
```

```
Student Information
{512=Ramesh, 501=Durga, 507=Naveen, 510=Madhu}
Employee IDs : [512, 501, 507, 510]
512 Ramesh
501 Durga
507 Naveen
510 Madhu
```

Collections Framework

LinkedHashMap : The HashMap class provides the facility of HashTable datastructure .

- Duplicate key not allowed and Values can be repeated.
- Null key is allowed only once and values null are allowed.
- ArrayDeque implements serializable and clonable Interfaces.
- Insertion order preserved

Constructors

LinkedHashMap():

Constructs an empty insertion-ordered LinkedHashMap instance with the default initial capacity (16) and load factor (0.75).

Collections Framework

```
import java.util.*;
public class LinkedHashMapDemo
{
	public static void main(String[] args)
	{
		Scanner read = new Scanner(System.in);
		LinkedHashMap studinfo = new LinkedHashMap();
		studinfo.put(501, "Durga");
		studinfo.put(510, "Madhu");
		studinfo.put(507, "Naveen");
		studinfo.put(512, "Ramesh");
		System.out.println("Student Information\n" + studinfo);
		System.out.print("Enter Rollnumber: ");
		int id = read.nextInt();
		if(studinfo.containsKey(id))
		{
		System.out.println("Student Name - " + studinfo.get(id));
		}
		else
		{
		System.out.println("No Record Found");
		}
	}
}
```

```
C:\Users\Exam\Desktop>java LinkedHashMapDemo
Student Information
{501=Durga, 510=Madhu, 507=Naveen, 512=Ramesh}
Enter Rollnumber: 510
Student Name - Madhu
```

Collections Framework

TreeMap : The TreeMap class is a child class of AbstractMap, and it implements the NavigableMap interface which is a child interface of SortedMap.

- It is used to store the data in the form of key, value pair using a red-black tree concepts.
- Duplicate key not allowed and Values can be repeated.
- Null keys and values not allowed.
- follows the ascending oreder based on keys.

Constructors

```
TreeMap tm=new TreeMap();

TreeMap tm=new TreeMap(int capacity)

TreeMap tm=new TreeMap(int capacity, float fillratio):

TreeMap tm=new TreeMap(Map m)
```

Collections Framework

```
import java.util.*;
class TreeMapDemo
{
public static void main(String args[])
{
TreeMap studinfo =new TreeMap();//
studinfo.put(501, "Durga");
studinfo.put(510, "Madhu");
studinfo.put(507, "Naveen");
studinfo.put(512, "Ramesh");
System.out.println(studinfo);
Set s =studinfo.keySet();
System.out.println(s);
Collection s1 =studinfo.values();
System.out.println(s1);
Set s2=studinfo.entrySet();
Iterator it=s2.iterator();
while(it.hasNext())
{
Map.Entry me=(Map.Entry)it.next();
if(me.getKey().equals("512"))
{
me.setValue("RameshTendulkar");
}
System.out.println(me.getKey()+"--->"+me.getValue());
}}}
```

```
C:\Users\Exam\Desktop>java TreeMapDemo
{501=Durga, 507=Naveen, 510=Madhu, 512=Ramesh}
[501, 507, 510, 512]
[Durga, Naveen, Madhu, Ramesh]
501--->Durga
507--->Naveen
510--->Madhu
512--->RameshTendulkar
```

Collections Framework

Dictionary( ): which works like a Map. The Dictionary is an abstract class used to store and manage elements in the form of a pair of key and value.

Dictionary d=new Dictionary( )

Object put(Object key, Object value): Inserts a key and its value into the dictionary. Returns null on success; returns the previous value associated with the key if the key is already exist.
Object remove(Object key): **it** returns the value associated with given key and removes the same; Returns null if the key does not exist.
Object get(Object key): It returns the value associated with given key; Returns null if the key does not exist.
Enumeration keys( ) Returns an enumeration of the keys contained in the dictionary.
Enumeration elements( ) Returns an enumeration of the values contained in the dictionary.
boolean isEmpty( ) It returns true if dictionary has no elements; otherwise returns false.
int size( ) It returns the total number of elements in the dictionary.

Collections Framework

```
import java.util.*;
public class DictionaryDemo {
        public static void main(String args[]) {
                Dictionary dict = new Hashtable();
                dict.put(501, "Durga");
                dict.put(510, "Madhu");
                dict.put(507, "Naveen");
                dict.put(512, "Ramesh");
                System.out.println("Dictionary\n=> " + dict);
        System.out.println("\n\nValue associated with key 512 => " + dict.get(512));
        System.out.println("\nDictionary has " + dict.size() + " elements");
        System.out.println("\nIs Dictionary empty?  " + dict.isEmpty());
        System.out.print("\nKeys in Dictionary\n=> ");
        Enumeration i = dict.keys();
        while(i.hasMoreElements()) {
        System.out.print(" " + i.nextElement());
        }
        System.out.print("\n\nValues in Dictionary\n=> ");
        Enumeration i1 = dict.elements();
        while(i1.hasMoreElements()) {
    System.out.print(" " + i1.nextElement());
  }}}
```

```
C:\Users\Admin\Desktop>java DictionaryDemo
Dictionary
=> {512=Ramesh, 501=Durga, 510=Madhu, 507=Naveen}


Value associated with key 512 => Ramesh

Dictionary has 4 elements

Is Dictionary empty?  false

Keys in Dictionary
=>  512 501 510 507

Values in Dictionary
=>  Ramesh Durga Madhu Naveen
```

Collections Framework

Hashtable which works like a HashMap but it is synchronized.
The Hashtable is a concrete class of Dictionary.
It is used to store and manage elements in the form of a pair of key and value.

Constructors:

Hashtable( ) It creates an empty hashtable with the default initial capacity 11.

Hashtable(int capacity) It creates an empty hashtable with the specified initial capacity.

Hashtable(int capacity, float loadFactor) It creates an empty hashtable with the specified initial capacity and loading factor.

Hashtable(Map m) It creates a hashtable containing elements of Map m.

Collections Framework

```
import java.util.*;

class HashtableDemo {
        public static void main(String args[]) {
                Hashtable ht = new Hashtable();
                ht.put(501, "Durga");
                ht.put(510, "Madhu");
                ht.put(507, "Naveen");
                ht.put(512, "Ramesh");
                System.out.println("HashTable\n=> " + ht);
        System.out.println("\n\nValue associated with key 512 => " + ht.get(512));
        System.out.println("\nHashtable has " + ht.size() + " elements");
        System.out.println("\nIs hashtable empty?  " + ht.isEmpty());

        System.out.print("\nKeys in hashtable\n=> ");
        System.out.print("\n\nValues in hashtable\n=> ");
        System.out.println("\nKeys => " + ht.keySet());
        System.out.println("\nKeys => " + ht.values());

}}
```

```
C:\Users\Admin\Desktop>java HashtableDemo
htionary
=> {512=Ramesh, 501=Durga, 510=Madhu, 507=Naveen}


Value associated with key 512 => Ramesh

Hashtable has 4 elements

Is hashtable empty?  false

Keys in hashtable
=>

Values in hashtable
=>
Keys => [512, 501, 510, 507]

Keys => [Ramesh, Durga, Madhu, Naveen]
```

Collections Framework

Properties which is a child class of Hashtable class.

- It implements interfaces like Map,
- Implements Cloneable, and Serializable.
- Properties class, we can load key, value pairs into a Properties object from a stream.
- Properties class, we can save the Properties object to a stream.

Constructors:

Properties( ) It creates an empty property list with no default values.

Properties(Properties defaults) It creates an empty property list with the specified defaults.

Methods

String getProperty(String key) It returns value associated with the specified key.

void setProperty(String key, String value) It calls the put method of Hashtable.

void store(OutputStream os, String comment) It writes the properties in the OutputStream object.

Collections Framework

```
import java.io.*;
import java.util.*;
public class PropertiesDemo {
        public static void main(String[] args) {
                FileOutputStream fos = null;
                File confFile = null;
                try {
                        confFile = new File("studentfile.properties");
                        fos = new FileOutputStream(confFile);
                        Properties cp = new Properties();
                        cp.setProperty("Name1", "Durga");
                        cp.setProperty("Name2", "Madhu");
                        cp.store(fos, "Student Names");
                        fos.close();
                        System.out.println("Student data saved!!!");
                }
                catch(Exception e) {
                        System.out.println("Something went wrong while opening file");
                }
        }
}
```

```
C:\Users\Admin\Desktop>java PropertiesDemo
Student data saved!!!
```

Collections Framework

The **StringTokenizer** class in java used to break a string into tokens

Constructors:

StringTokenizer(String str) It creates StringTokenizer object for the specified string str with default delimeter.

StringTokenizer(String str, String delimeter) It creates StringTokenizer object for the specified string str with specified delimeter.

StringTokenizer(String str, String delimeter, boolean returnValue) It creates StringTokenizer object with specified string, delimeter and returnValue.

Methods

boolean hasMoreTokens():checks if there is more tokens available.

String nextToken() It returns the next token from the StringTokenizer object.

String nextToken(String delimeter) It returns the next token based on the delimeter.

boolean hasMoreElements() It returns true if there are more tokens object. otherwise returns false.

Object nextElement() It returns the next token from the StringTokenizer object.

int countTokens(): returns the total number of tokens.

Collections Framework

```
import java.util.StringTokenizer;
public class StringTokenizerDemo{
 public static void main(String args[]){
   StringTokenizer st = new StringTokenizer("The StringTokenizer used to break a string into
tokens");
  StringTokenizer st2 = new StringTokenizer("The StringTokenizer,used to,break a, string, into
tokens",",");
  System.out.println("default Splitting of String1");
while (st.hasMoreTokens()) {
        System.out.println(st.nextToken());
    }
System.out.println("Comma Splitting of String2");
  while (st2.hasMoreElements()) {
        System.out.println(st2.nextElement());
    }
   System.out.println(st2.nextElement());
   }
}
```

```
C:\Users\Admin\Desktop>java StringTokenizerDemo
default Splitting of String1
The
StringTokenizer
used
to
break
a
string
into
tokens
Comma Splitting of String2
The StringTokenizer
used to
break a
 string
 into tokens
```

Collections Framework

The **Date** is a built-in class in java used to work with date and time in java.
The Date class is available inside the **java.util** package.
The Date class represents the date and time with millisecond precision.

Constructors:

Date( ) It creates a Date object that represents current date and time.
Date(long milliseconds) It creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.

Methods

long getTime() It returns the time represented by this date object.
void setTime(long time) It changes the current date and time to given time.
boolean after(Date date) It returns true, if the invoking date is after the argumented date.
int hashCode() It returns the hash code value of the invoking date object.
int compareTo(Date date) It compares current date with given date.
String toString() It converts this date into Instant object.

Collections Framework

```
import java.util.Date;
public class DateDemo {
        public static void main(String[] args) {
                Date time = new Date();
                System.out.println("Current date : " + time);
                System.out.println("Date : " + time.getTime() + " milliseconds");
                System.out.println("hashCode : " + time.hashCode());
                System.out.println("Date to String : " + time.toString());
        }
}
```

```
C:\Users\Admin\Desktop>java DateDemo
Current date : Sat Jul 17 08:11:56 IST 2021
Date : 1626489716488 milliseconds
hashCode : -1302888846
Date to String : Sat Jul 17 08:11:56 IST 2021
```

Collections Framework

The Comparator is an interface available in the **java.util** package.
The java **Comparator** is used to order the objects of user-defined classes.
The java Comparator can compare two objects from two different classes.
Using the java Comparator, we can sort the elements based on data members of a class.
For example, we can sort based on roolNo, age, salary, marks, etc.

Methods

int **compare(Object obj1, Object obj2):** It is used to compares the obj1 with o bj2 .

boolean equals(Object obj): It is used to check the equity between current object and argumented object.

Collections Framework

```
import java.util.*;

class Student{

        String name;
        double percentage;

        Student(String name, double percentage){
                this.name = name;
                this.percentage = percentage;
        }

}

class PercentageComparator implements Comparator<Student>{
        public int compare(Student stud1, Student stud2) {
                if(stud1.percentage < stud2.percentage)
                        return 1;
                return -1;
        }
}
```

Collections Framework

```
public class ComparatorStudent{

	public static void main(String args[]) {

		ArrayList<Student> studList = new ArrayList<Student>();

		studList.add(new Student("Madhu", 85.2));
		studList.add(new Student("Durga", 45.7));
		studList.add(new Student("Ramesh", 90.8));
		studList.add(new Student("NAveen", 76.5));

		Comparator<Student> com = new PercentageComparator();

		Collections.sort(studList, com);

		System.out.println("Avg % --> Name");
		System.out.println("---------------------");
		for(Student stud:studList) {
			System.out.println(stud.percentage + " --> " + stud.name);
		}
	}
}
```