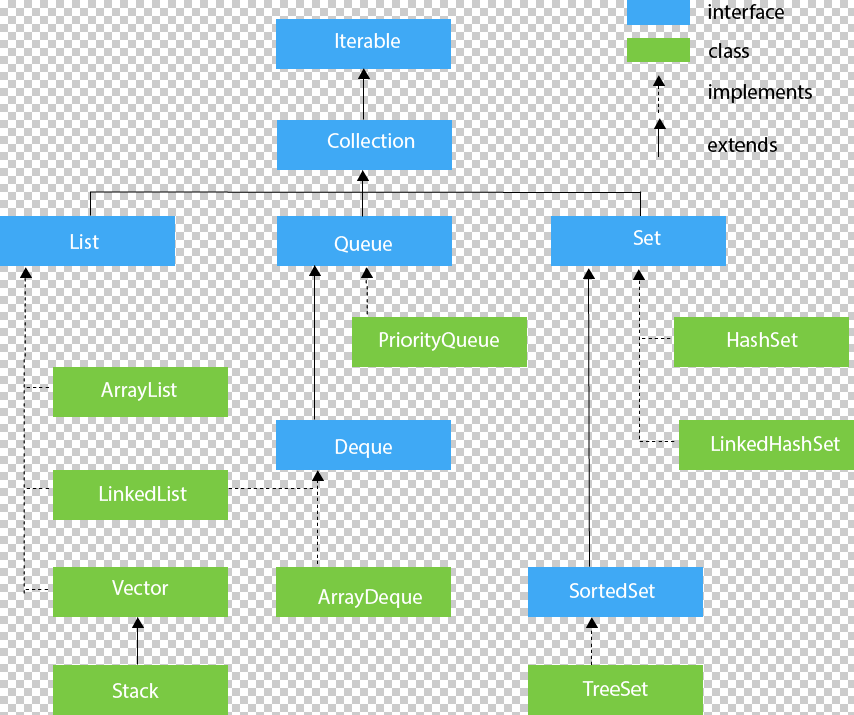
**Collections in Java**

* The **Collection in Java** is a framework that provides architecture to store and manipulate the group of objects.
* All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion, etc. can be achieved by Java Collections.
* Java Collection means a single unit of objects.
* Java Collection framework provides many interfaces (Set, List, Queue, Deque, etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet, etc.).

**Hierarchy of Collection Framework**



### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection c) | is used to insert the specified collection elements  in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection  from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection  except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no. of elements from the collection. |
| 8 | public boolean contains(Object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collections. |
| 14 | public int hashCode() | returns the hash code number of the collection. |

**COLLECTION CLASSES**

## ArrayList

* The ArrayList class implements the List interface.
* It uses a dynamic array to store the duplicate element of different data types.
* The ArrayList class maintains the insertion order and is non-synchronized.
* The elements stored in ArrayList class can be randomly accessed. **import** java.util.\*;

**class** TestJavaCollection1{

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

ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist

list.add("GOPAL");//Adding object in arraylist

list.add("AMMU");

list.add("CHINNI");

list.add("AMMU");

//Traversing list through Iterator

Iterator itr=list.iterator();

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

System.out.println(itr.next());  }  }  }

## LinkedList

* LinkedList implements the Collection interface.
* It uses a doubly linked list internally to store the elements.
* It can store the duplicate elements.
* It maintains the insertion order and is not synchronized.

**import** java.util.\*;

**public** **class** TestJavaCollection2{

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

LinkedList<String> al=**new** LinkedList<String>();

al.add("MGIT");

al.add("CSE");

al.addFirst("IT");

al.add("CSE");

Iterator<String> itr=al.iterator();

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

System.out.println(itr.next());  }  }  }

## HashSet

* HashSet class implements Set Interface.
* It represents the collection that uses a hash table for storage.
* Hashing is used to store the elements in the HashSet.
* It contains the unique items.

**import** java.util.\*;

**public** **class** TestJavaCollection7{

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

//Creating HashSet and adding elements

HashSet<String> set=**new** HashSet<String>();

set.add("AMMU");

set.add("SREE");

set.add("AMMU");

set.add("SREEKSHITA");

//Traversing elements

Iterator<String> itr=set.iterator();

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

System.out.println(itr.next());  }  }  }

## TreeSet

* Java TreeSet class implements the Set interface that uses a tree for storage.
* Like HashSet, TreeSet also contains the unique elements
* However, the access and retrieval time of TreeSet is quite fast.
* The elements in TreeSet stored in ascending order.

Consider the following example:

**import** java.util.\*;

**public** **class** TestJavaCollection9{

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

//Creating and adding elements

TreeSet<String> set=**new** TreeSet<String>();

set.add("GOPAL");

set.add("CHINNI");

set.add("GOPAL");

set.add("AMMU");

//traversing elements

Iterator<String> itr=set.iterator();

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

System.out.println(itr.next());  }  }  }

### Methods of Java Queue Interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(object) | It is used to insert the specified element into this queue and return trueupon success. |
| boolean offer(object) | It is used to insert the specified element into this queue. |
| Object remove() | It is used to retrieves and removes the head of this queue. |
| Object poll() | It is used to retrieves and removes the head of this queue,  or returns null if this queue is empty. |
| Object element() | It is used to retrieves, but does not remove, the head of this queue. |
| Object peek() | It is used to retrieves, but does not remove, the head of this queue,  or returns null if this queue is empty. |

## PriorityQueue class

The PriorityQueue class provides the facility of using queue.

### Java PriorityQueue Example

**import** java.util.\*;

**class** TestCollection12{

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

PriorityQueue<String> queue=**new** PriorityQueue<String>();

queue.add("Amit");

queue.add("Vijay");

queue.add("Karan");

queue.add("Jai");

queue.add("Rahul");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

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

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

**while**(itr2.hasNext()){

System.out.println(itr2.next());  }  }  }

## ArrayDeque class

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

The important points about ArrayDeque class are:

* Unlike Queue, we can add or remove elements from both sides.
* Null elements are not allowed in the ArrayDeque.
* ArrayDeque is not thread safe, in the absence of external synchronization.
* ArrayDeque has no capacity restrictions.
* ArrayDeque is faster than LinkedList and Stack.

## Java ArrayDeque Example

**import** java.util.\*;

**public** **class** ArrayDequeExample {

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

   //Creating Deque and adding elements

   Deque<String> deque = **new** ArrayDeque<String>();

   deque.add("Ravi");

   deque.add("Vijay");

   deque.add("Ajay");

   //Traversing elements

**for** (String str : deque) {

   System.out.println(str);     }     }  }

**ITERATOR**

* to cycle through the elements in a collection.
* For example, to display each element the easiest way is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.

## The Methods Declared by Iterator

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **boolean hasNext( )**  Returns true if there are more elements. Otherwise, returns false. |
| 2 | **Object next( )**  Returns the next element. Throws NoSuchElementException if there is not a next element. |
| 3 | **void remove( )**  Removes the current element. Throws IllegalStateException if an attempt is made to call remove( ) that is not preceded by a call to next( ). |

//EXAMPLES ABOVE -- SHOW ITERATOR

# For-each loop (Advanced or Enhanced For loop):

* The for-each loop introduced in Java5.
* It is mainly used to traverse array or collection elements.

## Advantage of for-each loop:

* It makes the code more readable.
* It elimnates the possibility of programming errors.

## Syntax of for-each loop:

**for**(data\_type variable : array | collection){}

### Simple Example of for-each loop for traversing the array elements:

**class** ForEachExample1{

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

**int** arr[]={12,13,14,44};

**for**(**int** i:arr){

     System.out.println(i);     }   }   }

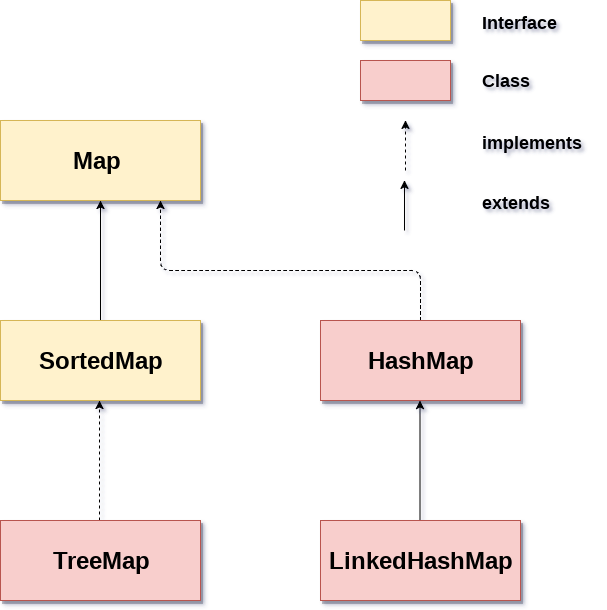
**Java Map Interface**

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.

## Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap and TreeMap. The hierarchy of Java Map is given below:



**Map doesn't allow duplicate keys, but you can have duplicate values.** HashMap and LinkedHashMap allows null keys and values but TreeMap doesn't allow any null key or value.

|  |  |
| --- | --- |
| **Class** | **Description** |
| [HashMap](https://www.javatpoint.com/java-hashmap) | HashMap is the implementation of Map but it  doesn't maintain any order. |
| [LinkedHashMap](https://www.javatpoint.com/java-linkedhashmap) | LinkedHashMap is the implementation of Map,  it inherits HashMap class.  It maintains insertion order. |
| [TreeMap](https://www.javatpoint.com/java-treemap) | TreeMap is the implementation of Map and SortedMap,  it maintains ascending order. |
| Method | Description |
| Object put(Object key, Object value) | It is used to insert an entry in this map. |
| void putAll(Map map) | It is used to insert the specified map in this map. |
| Object remove(Object key) | It is used to delete an entry for the specified key. |
| Object get(Object key) | It is used to return the value for the specified key. |
| boolean containsKey(Object key) | It is used to search the specified key from this map. |
| Set keySet() | It is used to return the Set view containing all the keys. |
| Set entrySet() | It is used to return the Set view containing all the keys  and values. |

## Map.Entry Interface

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

### Methods of Map.Entry interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| Object getKey() | It is used to obtain key. |
| Object getValue() | It is used to obtain value. |

**import** java.util.\*;

**class** MapInterfaceExample{

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

  Map<Integer,String> map=**new** HashMap<Integer,String>();

  map.put(100,"Amit");

  map.put(101,"Vijay");

  map.put(102,"Rahul");

**for**(Map.Entry m:map.entrySet()){

   System.out.println(m.getKey()+" "+m.getValue());   }   }  }

**Java HashMap class**

The important points about Java HashMap class are:

* A HashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It maintains no order.

### Java HashMap Example

# import java.util.\*;

# class TestCollection13{

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

# HashMap<Integer,String> hm=new HashMap<Integer,String>();

# hm.put(678,"Amit");

# hm.put(55,"Vijay");

# hm.put(103,"Rahul");

# hm.put(103,"naina");

# hm.put(104,"");

# hm.put(105,"");

# hm.put(null,"shanthi");

# hm.put(null,"sree");

# for(Map.Entry m:hm.entrySet()){

# System.out.println(m.getKey()+" "+m.getValue());

# }

# }

# }

# Java LinkedHashMap class

The important points about Java LinkedHashMap class are:

* A LinkedHashMap contains values based on the key.
* 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 LinkedHashMap Example

# import java.util.\*;

# class TestCollection14{

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

# LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integer,String>();

# 

# hm.put(104,"Amit");

# hm.put(102,"Vijay");

# hm.put(100,"Rahul");

# hm.put(103,"sree");

# hm.put(104,"");

# hm.put(105,"");

# hm.put(null,"shanthi");

# hm.put(null,"sree");

# for(Map.Entry m:hm.entrySet()){

# System.out.println(m.getKey()+" "+m.getValue()); } } }

# Java TreeMap class

Java TreeMap class implements the Map interface by using a tree. It provides an efficient means of storing key/value pairs in sorted order.

The important points about Java TreeMap class are:

* A TreeMap contains values based on the key.
* 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.

### Java TreeMap Example:

# import java.util.\*;

# class TestCollection15{

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

# TreeMap<Integer,String> hm=new TreeMap<Integer,String>();

# hm.put(100,"Amit");

# hm.put(102,"Ravi");

# hm.put(101,"Vijay");

# hm.put(103,"Rahul");

# hm.put(103,"payal");

# hm.put(104,"");

# hm.put(105,"");

# //hm.put(null,"shanthi");

# //hm.put(null,"sree");

# for(Map.Entry m:hm.entrySet()){

# System.out.println(m.getKey()+" "+m.getValue()); } } }

# Java Comparator interface

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

A comparator object is capable of comparing two objects of two different classes.

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.

#### compare() method

**public int compare(Object obj1,Object obj2):** compares the first object with second object.

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

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

## Java Comparator Example (Generic)

**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**

**import** java.util.\*;

**class** AgeComparator **implements** Comparator<Student>{

**public** **int** compare(Student s1,Student s2){

**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.

**import** java.util.\*;

**class** NameComparator **implements** Comparator<Student>{

**public** **int** compare(Student s1,Student s2){

**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<Student> al=**new** ArrayList<Student>();

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());

**for**(Student st: al){

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

}

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

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

**for**(Student st: al){

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

}  }  }

# Java - The Collection Algorithms

Some Methods

**static void copy(List list1, List list2)**

Copies the elements of list2 to list1.

**static Object max(Collection c)**

Returns the maximum element in **c** as determined by natural ordering. The collection need not be sorted.

**static Object min(Collection c)**

Returns the minimum element in **c** as determined by natural ordering.

**static Comparator reverseOrder( )**

Returns a reverse comparator.

**static void shuffle(List list)**

Shuffles (i.e., randomizes) the elements in list.

**static void sort(List list, Comparator comp)**

Sorts the elements of list as determined by comp.

**static void swap(List list, int idx1, int idx2)**

Exchanges the elements in the list at the indices specified by idx1 and idx2.

import java.util.\*;

public class AlgorithmsDemo {

public static void main(String args[]) {

// Create and initialize linked list

LinkedList ll = new LinkedList();

ll.add(new Integer(-8));

ll.add(new Integer(20));

ll.add(new Integer(-20));

ll.add(new Integer(8));

// Create a reverse order comparator

Comparator r = Collections.reverseOrder();

// Sort list by using the comparator

Collections.sort(ll, r);

// Get iterator

Iterator li = ll.iterator();

System.out.print("List sorted in reverse: ");

while(li.hasNext()) {

System.out.print(li.next() + " ");

}

System.out.println();

Collections.shuffle(ll);

// display randomized list

li = ll.iterator();

System.out.print("List shuffled: ");

while(li.hasNext()) {

System.out.print(li.next() + " ");

}

System.out.println();

System.out.println("Minimum: " + Collections.min(ll));

System.out.println("Maximum: " + Collections.max(ll));

}}

**LEGACY CLASSES AND INTERFACES**

**the following are the legacy classes defined by java.util package**

1. **Dictionary**
2. **HashTable**
3. **Properties**
4. **Stack**
5. **Vector**

**There is only one legacy interface called Enumeration**

**NOTE: All the legacy classes are synchronized**

**DICTIONARY**

Dictionary is an abstract class that represents a key/value storage repository and operates much like Map.

Given a key and value, you can store the value in a Dictionary object. Once the value is stored, you can retrieve it by using its key.

The abstract methods defined by Dictionary are listed below −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **Enumeration elements( )**  Returns an enumeration of the values contained in the dictionary. |
| 2 | **Object get(Object key)**  Returns the object that contains the value associated with the key. If the key is not in the dictionary, a null object is returned. |
| 3 | **boolean isEmpty( )**  Returns true if the dictionary is empty, and returns false if it contains at least one key. |
| 4 | **Enumeration keys( )**  Returns an enumeration of the keys contained in the dictionary. |
| 5 | **Object put(Object key, Object value)**  Inserts a key and its value into the dictionary. Returns null if the key is not already in the dictionary; returns the previous value associated with the key if the key is already in the dictionary. |
| 6 | **Object remove(Object key)**  Removes the key and its value. Returns the value associated with the key. If the key is not in the dictionary, a null is returned. |
| 7 | **int size( )**  Returns the number of entries in the dictionary. |

# Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

The important points about Java Hashtable class are:

* A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* It contains only unique elements.
* It may have not have any null key or value.
* It is synchronized.

**import** java.util.\*;

**class** TestCollection16{

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

  Hashtable<Integer,String> hm=**new** Hashtable<Integer,String>();

  hm.put(100,"Amit");

  hm.put(102,"Ravi");

  hm.put(101,"Vijay");

  hm.put(103,"Rahul");

**for**(Map.Entry m:hm.entrySet()){

   System.out.println(m.getKey()+" "+m.getValue());

  }

 }

}

Output:

103 Rahul

102 Ravi

101 Vijay

100 Amit

# 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.

### Recompilation is not required, if information is changed from properties file Example of Properties class to get information from properties file

To get information from the properties file, create the properties file first.

**db.properties**

user=system

password=oracle

Now, lets create the java class to read the data from the properties file.

**Test.java**

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

**public** **class** Test {

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

    FileReader reader=**new** FileReader("db.properties");

          Properties p=**new** Properties();

    p.load(reader);

          System.out.println(p.getProperty("user"));

    System.out.println(p.getProperty("password"));  }  }

Output:

system

Oracle

# Java Vector Class

|  |  |  |
| --- | --- | --- |
| 1) | [add()](https://www.javatpoint.com/java-vector-add-method) | It is used to append the specified element in the given vector. |
| 2) | [addAll()](https://www.javatpoint.com/java-vector-addall-method) | It is used to append all of the elements in the specified collection to the end of this Vector. |
| 3) | [addElement()](https://www.javatpoint.com/java-vector-addelement-method) | It is used to append the specified component to the end of this vector. It increases the vector size by one. |
| 4) | [capacity()](https://www.javatpoint.com/java-vector-capacity-method) | It is used to get the current capacity of this vector. |
| 5) | [clear()](https://www.javatpoint.com/java-vector-clear-method) | It is used to delete all of the elements from this vector. |
| 6) | [insertElementAt()](https://www.javatpoint.com/java-vector-insertelementat-method) | It is used to insert the specified object as a component in the given vector at the specified index. |
| 7) | [isEmpty()](https://www.javatpoint.com/java-vector-isempty-method) | It is used to check if this vector has no components. |
| 8) | [remove()](https://www.javatpoint.com/java-vector-remove-method) | It is used to remove the specified element from the vector. If the vector does not contain the element, it is unchanged. |
| 9) | [removeAll()](https://www.javatpoint.com/java-vector-removeall-method) | It is used to delete all the elements from the vector that are present in the specified collec |

Java Vector class comes under the java.util package. The vector class implements a growable array of objects.

SOME METHODS

**import** java.util.\*;

**public** **class** VectorExample1 {

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

          //Create an empty vector with initial capacity 4

          Vector<String> vec = **new** Vector<String>(4);

          //Adding elements to a vector

          vec.add("Tiger");

          vec.add("Lion");

          vec.add("Dog");

          vec.add("Elephant");

          //Check size and capacity

          System.out.println("Size is: "+vec.size());

          System.out.println("Default capacity is: "+vec.capacity());

          //Display Vector elements

          System.out.println("Vector element is: "+vec);

          vec.addElement("Rat");

          vec.addElement("Cat");

          vec.addElement("Deer");

          //Again check size and capacity after two insertions

          System.out.println("Size after addition: "+vec.size());

          System.out.println("Capacity after addition is: "+vec.capacity());

          //Display Vector elements again

          System.out.println("Elements are: "+vec);

          //Checking if Tiger is present or not in this vector

**if**(vec.contains("Tiger"))

            {

               System.out.println("Tiger is present at the index " +vec.indexOf("Tiger"));

            }

**else**

            {

               System.out.println("Tiger is not present in the list.");

            }

            //Get the first element

          System.out.println("The first animal of the vector is = "+vec.firstElement());

          //Get the last element

          System.out.println("The last animal of the vector is = "+vec.lastElement());

       }

}

**Output:**

Size is: 4

Default capacity is: 4

Vector element is: [Tiger, Lion, Dog, Elephant]

Size after addition: 7

Capacity after addition is: 8

Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]

Tiger is present at the index 0

The first animal of the vector is = Tiger

The last animal of the vector is = Deer

# Java - The Stack Class

Stack is a subclass of Vector that implements a standard last-in, first-out stack.

Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **boolean empty()**  Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements. |
| 2 | **Object peek( )**  Returns the element on the top of the stack, but does not remove it. |
| 3 | **Object pop( )**  Returns the element on the top of the stack, removing it in the process. |
| 4 | **Object push(Object element)**  Pushes the element onto the stack. Element is also returned. |
| 5 | **int search(Object element)**  Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, .1 is returned. |

import java.util.\*;

public class StackDemo {

static void showpush(Stack st, int a) {

st.push(new Integer(a));

System.out.println("push(" + a + ")");

System.out.println("stack: " + st);

}

static void showpop(Stack st) {

System.out.print("pop -> ");

Integer a = (Integer) st.pop();

System.out.println(a);

System.out.println("stack: " + st);

}

public static void main(String args[]) {

Stack st = new Stack();

System.out.println("stack: " + st);

showpush(st, 42);

showpush(st, 66);

showpush(st, 99);

showpop(st);

showpop(st);

showpop(st);

try {

showpop(st);

} catch (EmptyStackException e) {

System.out.println("empty stack");

}

}

}

# StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

Constructors of StringTokenizer class

There are 3 constructors defined in the StringTokenizer class.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringTokenizer(String str) | creates StringTokenizer with specified string. |
| StringTokenizer(String str, String delim) | creates StringTokenizer with specified string and delimeter. |
| StringTokenizer(String str, String delim, boolean returnValue) | creates StringTokenizer with specified string, delimeter  and returnValue. If return value is true,  delimiter characters are considered to be tokens. If it is false,  delimiter characters serve to separate tokens. |

Methods of StringTokenizer class

The 6 useful methods of StringTokenizer class are as follows:

|  |  |
| --- | --- |
| **Public method** | **Description** |
| boolean hasMoreTokens() | checks if there is more tokens available. |
| String nextToken() | returns the next token from the StringTokenizer object. |
| String nextToken(String delim) | returns the next token based on the delimeter. |
| boolean hasMoreElements() | same as hasMoreTokens() method. |
| Object nextElement() | same as nextToken() but its return type is Object. |
| int countTokens() | returns the total number of tokens. |

**import** java.util.StringTokenizer;

**public** **class** Simple{

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

   StringTokenizer st = **new** StringTokenizer("we are students of cse");

**while** (st.hasMoreTokens()) {

         System.out.println(st.nextToken());

     }

   }

}

# Java - The BitSet Class

The BitSet class creates a special type of array that holds bit values. The BitSet array can increase in size as needed. This makes it similar to a vector of bits.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **BitSet( )**  This constructor creates a default object. |
| 2 | **BitSet(int size)**  This constructor allows you to specify its initial size, i.e., the number of bits that it can hold. All bits are initialized to zero. |

import java.util.BitSet;

public class BitSetDemo {

public static void main(String args[]) {

BitSet bits1 = new BitSet(16);

BitSet bits2 = new BitSet(16);

// set some bits

for(int i = 0; i < 16; i++) {

if((i % 2) == 0) bits1.set(i);

if((i % 5) != 0) bits2.set(i);

}

System.out.println("Initial pattern in bits1: ");

System.out.println(bits1);

System.out.println("\nInitial pattern in bits2: ");

System.out.println(bits2);

// AND bits

bits2.and(bits1);

System.out.println("\nbits2 AND bits1: ");

System.out.println(bits2);

// OR bits

bits2.or(bits1);

System.out.println("\nbits2 OR bits1: ");

System.out.println(bits2);

// XOR bits

bits2.xor(bits1);

System.out.println("\nbits2 XOR bits1: ");

System.out.println(bits2);

}

}

## Output

Initial pattern in bits1:

{0, 2, 4, 6, 8, 10, 12, 14}

Initial pattern in bits2:

{1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14}

bits2 AND bits1:

{2, 4, 6, 8, 12, 14}

bits2 OR bits1:

{0, 2, 4, 6, 8, 10, 12, 14}

bits2 XOR bits1:

{}

# java.util.Date

The java.util.Date class represents date and time in java.

|  |  |  |
| --- | --- | --- |
| **No.** | **Constructor** | **Description** |
| 1) | Date() | Creates a date object representing current date and time. |
| 2) | Date(long milliseconds) | Creates a date object for the given milliseconds since January 1,  1970, 00:00:00 GMT. |

## java.util.Date Methods

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | [boolean after(Date date)](https://www.javatpoint.com/java-date-after-method) | tests if current date is after the given date. |
| 2) | [boolean before(Date date)](https://www.javatpoint.com/java-date-before-method) | tests if current date is before the given date. |
| 3) | [Object clone()](https://www.javatpoint.com/java-date-clone-method) | returns the clone object of current date. |
| 4) | [int compareTo(Date date)](https://www.javatpoint.com/java-date-compareto-method) | compares current date with given date. |
| 5) | [boolean equals(Date date)](https://www.javatpoint.com/java-date-equals-method) | compares current date with given date for equality. |
| 6) | [static Date from(Instant instant)](https://www.javatpoint.com/java-date-from-method) | returns an instance of Date object from Instant date. |
| 7) | [long getTime()](https://www.javatpoint.com/java-date-gettime-method) | returns the time represented by this date object. |
| 8) | [int hashCode()](https://www.javatpoint.com/java-date-hashcode-method) | returns the hash code value for this date object. |
| 9) | [void setTime(long time)](https://www.javatpoint.com/java-date-settime-method) | changes the current date and time to given time. |
| 10) | [Instant toInstant()](https://www.javatpoint.com/java-date-toinstant-method) | converts current date into Instant object. |
| 11) | [String toString()](https://www.javatpoint.com/java-date-tostring-method) | converts this date into Instant object. |

# Java Calendar Class

Java Calendar class is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc.

There are around 48 methods

Some of them are

|  |  |  |
| --- | --- | --- |
| 1. | [public void add(int field, int amount)](https://www.javatpoint.com/post/java-calendar-add-method) | Adds the specified (signed) amount of time to the  given calendar field. |
| 2. | [public boolean after (Object when)](https://www.javatpoint.com/post/java-calendar-after-method) | The method Returns true if the time represented  by this Calendar is after the time represented by  when Object. |
| 3. | [public boolean before(Object when)](https://www.javatpoint.com/post/java-calendar-before-method) | The method Returns true if the time represented by this  Calendar is before the time represented by when Object. |
| 4. | [public final void clear(int field)](https://www.javatpoint.com/post/java-calendar-clear-method) | Set the given calendar field value and the time value of  this Calendar undefined. |
| 6. | [public int compareTo(Calendar anotherCalendar)](https://www.javatpoint.com/post/java-calendar-compareto-method) | The compareTo() method of Calendar class compares the  time values (millisecond offsets) between two calendar  object. |

**import** java.util.Calendar;

**public** **class** CalendarExample1 {

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

   Calendar calendar = Calendar.getInstance();

   System.out.println("The current date is : " + calendar.getTime());

   calendar.add(Calendar.DATE, -15);

   System.out.println("15 days ago: " + calendar.getTime());

   calendar.add(Calendar.MONTH, 4);

   System.out.println("4 months later: " + calendar.getTime());

   calendar.add(Calendar.YEAR, 2);

   System.out.println("2 years later: " + calendar.getTime());

   }

}

**RANDOM CLASS IN UTIL PACKAGE**

The **java.util.Random** class instance is used to generate a stream of pseudorandom numbers.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **Random()**  This creates a new random number generator. |
| 2 | **Random(long seed)**  This creates a new random number generator using a single long seed. |
| Sr.No. | **Method & Description** |
| 1 | [**protected int next(int bits)**](https://www.tutorialspoint.com/java/util/random_next.htm)  **This method generates the next pseudorandom number.** |
| 2 | [**boolean nextBoolean()**](https://www.tutorialspoint.com/java/util/random_nextboolean.htm)  **This method returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.** |
| 4 | [**double nextDouble()**](https://www.tutorialspoint.com/java/util/random_nextdouble.htm)  **This method returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.** |
| 5 | [**float nextFloat()**](https://www.tutorialspoint.com/java/util/random_nextfloat.htm)  **This method returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.** |
| 6 | [**int nextInt()**](https://www.tutorialspoint.com/java/util/random_nextint.htm)  This method returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence. | |
| 8 | [**int nextInt(int n)**](https://www.tutorialspoint.com/java/util/random_nextint_inc_exc.htm)  This method returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. |
| 9 | [**long nextLong()**](https://www.tutorialspoint.com/java/util/random_nextlong.htm)  This method returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence. |

# Java.util.Formatter Class

The **java.util.Formatter** class provides support for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **Formatter()**  This constructor constructs a new formatter. |
| 4 | **Formatter(File file)**  **This constructor constructs a new formatter with the specified file** |
| Sr.No. | **Method & Description** |
| 1 | [**void close()**](https://www.tutorialspoint.com/java/util/formatter_close.htm)  **This method closes this formatter.** |
| 2 | [**void flush()**](https://www.tutorialspoint.com/java/util/formatter_flush.htm)  **This method flushes this formatter.** |

import java.util.\*;

class FormatterDemo {

public static void main(String args[])

{

// create Formatter class object

Formatter f = new Formatter();

// + sign specifier

f.format("%+d", 111);

System.out.println(f);

f = new Formatter();

f.format("%(d",-111);

System.out.println(f);

f = new Formatter();

f.format("unit4%nrevision");

System.out.println(f);

}}

# Java Scanner Class

Java Scanner class comes under the java.util package. Java has various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default.

|  |  |  |
| --- | --- | --- |
| **SN** | **Constructor** | **Description** |
| 1) | Scanner(File source) | It constructs a new Scanner that produces values scanned  from the specified file. |
| 2) | Scanner(InputStream source) | It constructs a new Scanner that produces values scanned  from the specified input stream. |

## Java Scanner Class Methods

The following are the list of Scanner methods:

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Modifier & Type** | **Method** | **Description** |
| 1) | void | [close()](https://www.javatpoint.com/post/java-scanner-close-method) | It is used to close this scanner. |
| 2) | boolean | [hasNext()](https://www.javatpoint.com/post/java-scanner-hasnext-method) | It returns true if this scanner has another token  in its input. |
| 3) | boolean | [hasNextBoolean()](https://www.javatpoint.com/post/java-scanner-hasnextboolean-method) | It is used to check if the next token in this scanner's  input can be interpreted as a Boolean using the  nextBoolean() method or not. |
| 4) | boolean | [hasNextDouble()](https://www.javatpoint.com/post/java-scanner-hasnextdouble-method) | It is used to check if the next token in this scanner's  input can be interpreted as a BigDecimal using the  nextByte() method or not. |
| 5) | boolean | [hasNextFloat()](https://www.javatpoint.com/post/java-scanner-hasnextfloat-method) | It is used to check if the next token in this scanner's  input can be interpreted as a Float using the nextFloat() method or not. |
| 6) | boolean | [hasNextInt()](https://www.javatpoint.com/post/java-scanner-hasnextint-method) | It is used to check if the next token in this scanner's  input can be interpreted as an int using the nextInt()  method or not. |
| 7) | boolean | [hasNextLong()](https://www.javatpoint.com/post/java-scanner-hasnextlong-method) | It is used to check if the next token in this scanner's  input can be interpreted as a Long using the nextLong()  method or not. |
| 8) | boolean | [hasNextShort()](https://www.javatpoint.com/post/java-scanner-hasnextshort-method) | It is used to check if the next token in this scanner's  input can be interpreted as a Short using the nextShort() method or not. |

EXAMPLE

import java.util.Scanner;

class ScannerExample

{

public static void main(String args[])

{

Scanner s=new Scanner(System.in);

int a= s.nextInt();

System.out.println(a);

}}