**UNIT – IV**

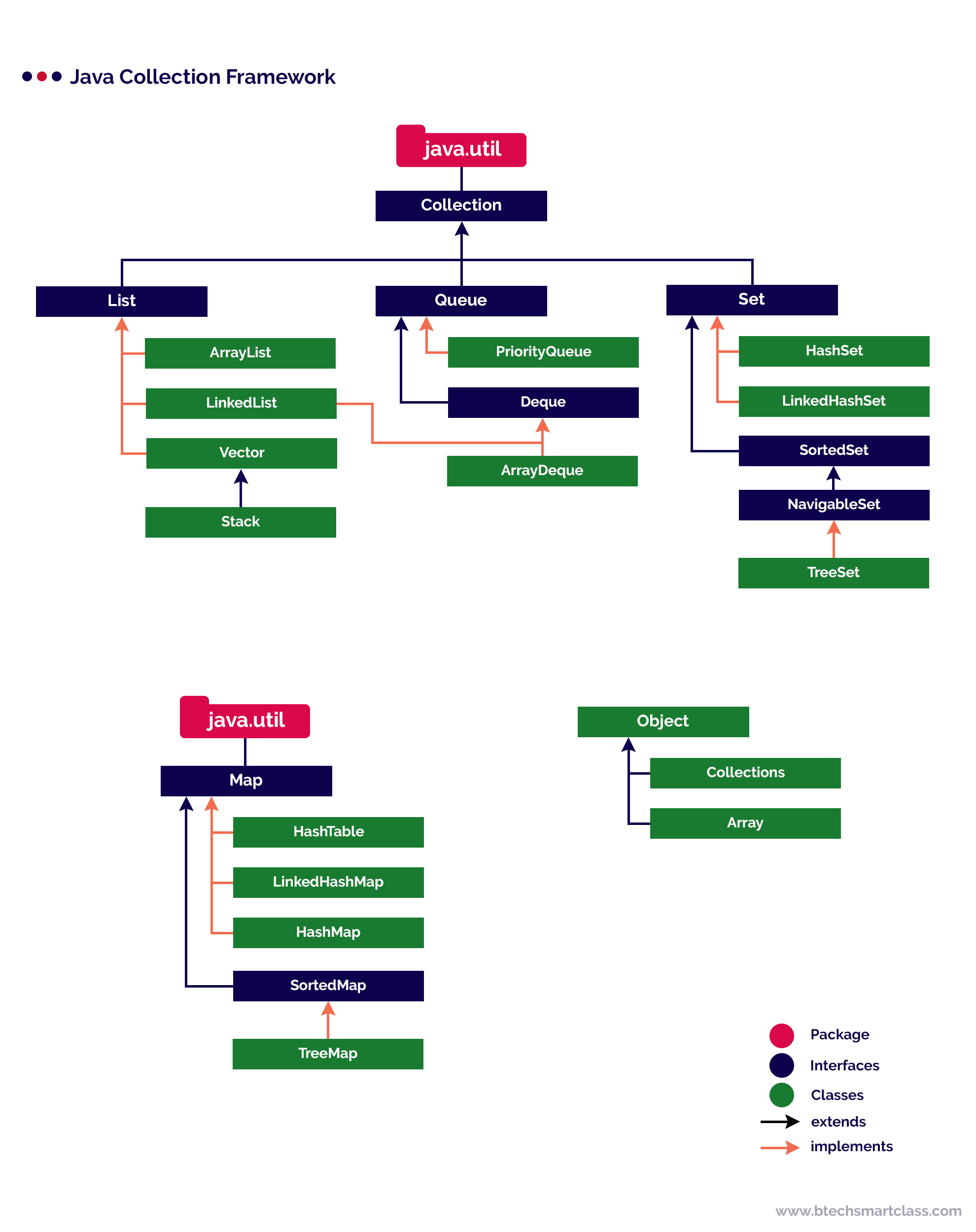
**THE COLLECTIONS FRAMEWORK**

**4.1 Collections overview:**

Java collection framework is a collection of interfaces and classes used to storing and processing a group of individual objects as a single unit. The java collection framework holds several classes that provide a large number of methods to store and process a group of objects. These classes make the programmer task super easy and fast.

Java collection framework was introduced in java 1.2 version.

Java collection framework has the following heirarchy.



Before the collection framework in java (before java 1.2 version), there was a set of classes like **Array**, **Vector**, **Stack**, **HashTable**. These classes are known as **legacy classes**.

The java collection framework contains List, Queue, Set, and Map as top-level interfaces. The List, Queue, and Set stores single value as its element, whereas Map stores a pair of a key and value as its element.

**Collection Classes:**

**ArrayList class**

The ArrayList class is a part of java collection framework. It is available inside the java.util package. The elements of ArrayList are organized as an array internally. The default size of an ArrayList is 10.

The ArrayList class has the following declaration.

ArrayList class constructors

The ArrayList class has the following constructors.

ArrayList( ) - Creates an empty ArrayList.

ArrayList(Collection c) - Creates an ArrayList with given collection of elements.

ArrayList(int size) - Creates an empty ArrayList with given size (capacity).

Operations on ArrayList

The ArrayList class allow us to perform several operations like adding, accesing, deleting, updating, looping, etc. Let's look at each operation with examples.

**Adding Items**

The ArrayList class has the following methods to add items.

boolean add(E element) - Appends given element to the ArrayList.

boolean addAll(Collection c) - Appends given collection of elements to the ArrayList.

void add(int index, E element) - Inserts the given element at specified index.

boolean addAll(int index, Collection c) - Inserts the given collection of elements at specified index.

Let's consider an example program to illustrate adding items to the ArrayList.

Example

import java.util.\*;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list\_1 = new ArrayList<String>();

ArrayList list\_2 = new ArrayList();

//Appending

list\_1.add("BTech");

System.out.println("list\_1: " + list\_1);

list\_1.add("Class");

System.out.println("list\_1: " + list\_1);

//Inserting at specified index

list\_1.add(1, "Smart");

System.out.println("list\_1: " + list\_1);

//Appending a collection of elements

list\_2.addAll(list\_1);

System.out.println("list\_2: " + list\_2);

//Inserting collection of elements at specified index

list\_2.addAll(2, list\_1);

System.out.println("list\_2: " + list\_2);

}

}

**Accessing Items**

The ArrayList class has the following methods to access items.

E get(int index) - Returns element at specified index from the ArrayList.

ArrayList subList(int startIndex, int lastIndex) - Returns an ArrayList that contails elements from specified startIndex to lastIndex-1 from the invoking ArrayList.

int indexOf(E element) - Returns the index value of given element first occurence in the ArrayList.

int lastIndexOf(E element) - Returns the index value of given element last occurence in the ArrayList.

Let's consider an example program to illustrate accessing items from the ArrayList.

Example

import java.util.\*;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list\_1 = new ArrayList<String>();

list\_1.add("BTech");

list\_1.add("Smart");

list\_1.add("Class");

list\_1.add("-");

list\_1.add("Java");

list\_1.add("Tutorial");

list\_1.add("Class");

System.out.println("Element at index 4 is " + list\_1.get(4));

System.out.println("Sublist from index 1 to 4: " + list\_1.subList(1, 5));

System.out.println("Index of element \"Class\" is " + list\_1.indexOf("Class"));

System.out.println("Last index of element \"Class\" is " + list\_1.lastIndexOf("Class"));

}

}

**Updating Items**

The ArrayList class has the following methods to update or change items.

E set(int index, E newElement) - Replace the element at specified index with newElement in the invoking ArrayList.

ArrayList replaceAll(UnaryOperator e) - Replaces each element of invoking ArrayList with the result of applying the operator to that element.

Let's consider an example program to illustrate updating items in the ArrayList.

Example

import java.util.\*;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list\_1 = new ArrayList<String>();

list\_1.add("BTech");

list\_1.add("Smart");

list\_1.add("Class");

list\_1.add("-");

list\_1.add("Java");

list\_1.add("Tutorial");

list\_1.add("Class");

System.out.println("\nList before update: " + list\_1);

list\_1.set(3, ":");

System.out.println("\nList after update: " + list\_1);

list\_1.replaceAll(e -> e.toUpperCase());

System.out.println("\nList after update: " + list\_1);

}

}

**Removing Items**

The ArrayList class has the following methods to remove items.

E remove(int index) - Removes the element at specified index in the invoking ArrayList.

boolean remove(Object element) - Removes the first occurence of the given element from the invoking ArrayList.

boolean removeAll(Collection c) - Removes the given collection of elements from the invoking ArrayList.

void retainAll(Collection c) - Removes all the elements except the given collection of elements from the invoking ArrayList.

boolean removeIf(Predicate filter) - Removes all the elements from the ArrayList that satisfies the given predicate.

void clear( ) - Removes all the elements from the ArrayList.

Let's consider an example program to illustrate removing items from the ArrayList.

Example

import java.util.\*;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list\_1 = new ArrayList<String>();

ArrayList list\_2 = new ArrayList();

ArrayList list\_3 = new ArrayList();

list\_1.add("BTech");

list\_1.add("Smart");

list\_1.add("Class");

list\_1.add("-");

list\_1.add("Java");

list\_1.add("Tutorial");

list\_1.add("Classes");

list\_1.add("on");

list\_1.add("Collection");

list\_1.add("framwork");

list\_1.add("-");

list\_1.add("ArrayList");

list\_2.add("Tutorial");

list\_2.add("Java");

list\_3.add("BTech");

list\_3.add("Smart");

list\_3.add("Class");

System.out.println("\nList\_1 before remove:\n" + list\_1);

System.out.println("\nList\_2 before remove:\n" + list\_2);

System.out.println("\nList\_3 before remove:\n" + list\_3);

list\_1.remove(3);

System.out.println("\nList after removing element from index 3:\n" + list\_1);

list\_1.remove("Tutorial");

System.out.println("\nList after removing \'Tutorial\' element:\n" + list\_1);

list\_1.removeAll(list\_2);

System.out.println("\nList after removing all elements of list\_2 from list\_1:\n" + list\_1);

list\_1.removeIf(n -> n.equals("Classes"));

System.out.println("\nList after removing all elements that are equal to \'Classes\':\n" + list\_1);

list\_1.retainAll(list\_3);

System.out.println("\nList after removing all elements from list\_1 except elements of list\_3:\n" + list\_1);

list\_1.clear();

System.out.println("\nList after removing all elements from list\_1:\n" + list\_1);

}

}

**LinkedList**

The LinkedList class is a part of java collection framework. It is available inside the java.util package.

The LinkedList class has the following constructors.

LinkedList( ) - Creates an empty List.

LinkedList(Collection c) - Creates a List with given collection of elements.

Operations on LinkedList

The LinkedList class allow us to perform several operations like adding, accesing, deleting, updating, looping, etc. Let's look at each operation with examples.

**Adding Items**

The LinkedList class has the following methods to add items.

boolean add(E element) - Appends given element to the List.

boolean addAll(Collection c) - Appends given collection of elements to the List.

void add(int position, E element) - Inserts the given element at specified position.

boolean addAll(int position, Collection c) - Inserts the given collection of elements at specified position.

void addFirst(E element) - Inserts the given element at beggining of the list.

void addLast(E element) - Inserts the given element at end of the list.

boolean offer(E element) - Inserts the given element at end of the list.

boolean offerFirst(E element) - Inserts the given element at beggining of the list.

boolean offerLast(E element) - Inserts the given element at end of the list.

void push(E element) - Inserts the given element at beggining of the list.

Let's consider an example program to illustrate adding items to the LinkedList.

Example

import java.util.\*;

public class LinkedListExample {

public static void main(String[] args) {

LinkedList<String> list\_1 = new LinkedList<String>();

LinkedList list\_2 = new LinkedList();

list\_2.add(10);

list\_2.add(20);

list\_2.addFirst(5);

list\_2.addLast(25);

list\_2.offer(2);

list\_2.offerFirst(1);

list\_2.offerLast(10);

list\_2.push(40);

list\_1.addAll(list\_2);

System.out.println("List\_1: " + list\_1);

System.out.println("List\_2: " + list\_2);

}

}

**Accessing Items**

The LinkedList class has the following methods to access items.

E get(int position) - Returns element at specified position from the LinkedList.

E element( ) - Returns the first element from the invoking LinkedList.

E getFirst( ) - Returns the first element from the invoking LinkedList.

E getLast( ) - Returns the last element from the invoking LinkedList.

E peek( ) - Returns the first element from the invoking LinkedList.

E peekFirst( ) - Returns the first element from the invoking LinkedList, and returns null if list is empty.

E peekLast( ) - Returns the last element from the invoking LinkedList, and returns null if list is empty.

int indexOf(E element) - Returns the index value of given element first occurence in the LinkedList.

int lastIndexOf(E element) - Returns the index value of given element last occurence in the LinkedList.

E pop( ) - Returns the first element from the invoking LinkedList.

Let's consider an example program to illustrate accessing items from the LinkedList.

Example

import java.util.\*;

public class LinkedListExample {

public static void main(String[] args) {

LinkedList list\_1 = new LinkedList();

for(int i = 1; i <= 10; i++)

list\_1.add(i);

System.out.println("List is " + list\_1 + "\n");

System.out.println("get(position) - " + list\_1.get(3));

System.out.println("getFirst() - " + list\_1.getFirst());

System.out.println("getLast() - " + list\_1.getLast());

System.out.println("element() - " + list\_1.element());

System.out.println("peek() - " + list\_1.peek());

System.out.println("peekFirst() - " + list\_1.peekFirst());

System.out.println("peekLast() - " + list\_1.peekLast());

System.out.println("pop() - " + list\_1.pop());

System.out.println("indexOf(element) - " + list\_1.indexOf(5));

System.out.println("lastIndexOf(element) - " + list\_1.lastIndexOf(5));

}

}

**Updating Items**

The LinkedList class has the following methods to update or change items.

E set(int index, E newElement) - Replace the element at specified index with newElement in the invoking LinkedList.

Let's consider an example program to illustrate updating items in the LinkedList.

Example

import java.util.\*;

public class LinkedListExample {

public static void main(String[] args) {

LinkedList list\_1 = new LinkedList();

for(int i = 1; i <= 10; i++)

list\_1.add(i);

System.out.println("List is " + list\_1 + "\n");

list\_1.set(3, 50);

System.out.println("List after update at index 3 is\n" + list\_1 + "\n");

}

}

**Removing Items**

The LinkedList class has the following methods to remove items.

E remove( ) - Removes the first element from the invoking LinkedList.

E remove(int index) - Removes the element at specified index in the invoking LinkedList.

boolean remove(Object element) - Removes the first occurrence of the given element from the invoking LinkedList.

E removeFirst( ) - Removes the first element from the invoking LinkedList.

E removeLast( ) - Removes the last element from the invoking LinkedList.

boolean removeFirstOccurrence(Object element) - Removes from the first occurrence of the given element from the invoking LinkedList.

boolean removeLastOccurrence(Object element) - Removes from the last occurrence of the given element from the invoking LinkedList.

E poll( ) - Removes the first element from the LinkedList, and returns null if the list is empty.

E pollFirst( ) - Removes the first element from the LinkedList, and returns null if the list is empty.

E pollLast( ) - Removes the last element from the LinkedList, and returns null if the list is empty.

E pop( ) - Removes the first element from the LinkedList.

void clear( ) - Removes all the elements from the LinkedList.

Let's consider an example program to illustrate removing items from the LinkedList.

Example

import java.util.\*;

public class LinkedListExample {

public static void main(String[] args) {

LinkedList list\_1 = new LinkedList();

for(int i = 1; i <= 10; i++)

list\_1.add(i);

System.out.println("List initially is " + list\_1);

list\_1.remove();

System.out.println("\nList after remove()\n" + list\_1);

list\_1.remove(3);

System.out.println("\nList after remove(index)\n" + list\_1);

list\_1.removeFirst();

System.out.println("\nList after removeFirst()\n" + list\_1);

list\_1.removeLast();

System.out.println("\nList after removeLast()\n" + list\_1);

list\_1.removeFirstOccurrence(4);

System.out.println("\nList after removeFirstOccurrence()\n" + list\_1);

list\_1.removeLastOccurrence(7);

System.out.println("\nList after removeLastOccurrence()\n" + list\_1);

list\_1.pop();

System.out.println("\nList after pop()\n" + list\_1);

list\_1.clear();

System.out.println("\nList after clear()\n" + list\_1);

}

}

**HashSet**

The HashSet class is a part of java collection framework. It is available inside the java.util package. The HashSet class extends AbstractSet class and implements Set interface.The elements of HashSet are organized using a mechanism called hashing. The HashSet is used to create hash table for storing set of elements.

The HashSet class is used to create a collection that uses a hash table for storing set of elements.

* The HashSet is a child class of AbstractSet
* The HashSet implements interfaces like Set, Cloneable, and Serializable.
* The HashSet does not allows to store duplicate data values, but null values are allowed.
* The HashSet does not maintains the order of insertion.
* The HashSet initial capacity is 16 elements.
* The HashSet is best suitable for search operations.

HashSet class declaration

The HashSet class has the following declaration.

Example

public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable

HashSet class constructors

The HashSet class has the following constructors.

HashSet( ) - Creates an empty HashSet with the default initial capacity (16).

HashSet(Collection c) - Creates a HashSet with given collection of elements.

HashSet(int initialCapacity) - Creates an empty HashSet with the specified initial capacity.

HashSet(int initialCapacity, float loadFactor) - Creates an empty HashSet with the specified initial capacity and loadFactor.

**Operations on HashSet**

The HashSet class allow us to perform several operations like adding, accesing, deleting, updating, looping, etc. Let's look at each operation with examples.

**Adding Items**

The HashSet class has the following methods to add items.

boolean add(E element) - Inserts given element to the HashSet.

boolean addAll(Collection c) - Inserts given collection of elements to the HashSet.

Let's consider an example program to illustrate adding items to the HashSet.

Example

import java.util.\*;

public class HashSetExample {

public static void main(String[] args) {

HashSet set = new HashSet();

HashSet anotherSet = new HashSet();

set.add(10);

set.add(20);

set.add(30);

set.add(40);

set.add(50);

System.out.println("\nHashSet is\n" + set);

anotherSet.addAll(set);

System.out.println("\nanotherSet is\n" + anotherSet);

}

}

**Removing Items**

The HashSet class has the following methods to remove items.

boolean remove(Object o) - Removes the specified element from the invoking HashSet.

boolean removeAll(Collection c) - Removes all the elements of specified collection from the invoking HashSet.

boolean removeIf(Predicate p) - Removes all of the elements of HashSet collection that satisfy the given predicate.

boolean retainAll(Collection c) - Removes all of the elements of HashSet collection except specified collection of elements.

void clear( ) - Removes all the elements from the HashSet.

Let's consider an example program to illustrate removing items from the HashSet.

Example

import java.util.\*;

public class HashSetExample {

public static void main(String[] args) {

HashSet set = new HashSet();

HashSet anotherSet = new HashSet();

set.add(10);

set.add(20);

set.add(30);

set.add(40);

set.add(50);

System.out.println("\nHashSet is\n" + set);

anotherSet.addAll(set);

System.out.println("\nanotherSet is\n" + anotherSet);

set.remove(20);

System.out.println("\nHashSet after remove(20) is\n" + set);

anotherSet.removeAll(set);

System.out.println("\nanotherSet after removeAll(set) is\n" + anotherSet);

set.retainAll(anotherSet);

System.out.println("\nset after retainAll(anotherSet) is\n" + set);

anotherSet.clear();

System.out.println("\nanotherSet after clear() is\n" + anotherSet);

}

}

**TreeSet**

The TreeSet class is a part of java collection framework. It is available inside the java.util package. The TreeSet class extends AbstractSet class and implements NavigableSet, Cloneable, and Serializable interfaces.The elements of TreeSet are organized using a mechanism called tree. The TreeSet class internally uses a TreeMap to store elements. The elements in a TreeSet are sorted according to their natural ordering.

* The TreeSet is a child class of AbstractSet
* The TreeSet implements interfaces like NavigableSet, Cloneable, and Serializable.
* The TreeSet does not allows to store duplicate data values, but null values are allowed.
* The elements in a TreeSet are sorted according to their natural ordering.
* The TreeSet initial capacity is 16 elements.
* The TreeSet is best suitable for search operations.

TreeSet class declaration

The TreeSet class has the following declaration.

Examplesorting order

public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable

TreeSet class constructors

The TreeSet class has the following constructors.

TreeSet( ) - Creates an empty TreeSet in which elements will get stored in default natural sorting order.

TreeSet(Collection c) - Creates a TreeSet with given collection of elements.

TreeSet(Comparator c) - Creates an empty TreeSet with the specified sorting order.

TreeSet(SortedSet s) - This constructor is used to convert SortedSet to TreeSet.

Operations on TreeSet

The TreeSet class allow us to perform several operations like adding, accesing, deleting, updating, looping, etc. Let's look at each operation with examples.

**Adding Items**

The TreeSet class has the following methods to add items.

boolean add(E element) - Inserts given element to the TreeSet if it does not exist.

boolean addAll(Collection c) - Inserts given collection of elements to the TreeSet.

Let's consider an example program to illustrate adding items to the TreeSet.

Example

import java.util.\*;

public class TreeSetExample {

public static void main(String[] args) {

TreeSet set = new TreeSet();

TreeSet anotherSet = new TreeSet();

set.add(10);

set.add(20);

set.add(15);

set.add(5);

System.out.println("\nset is\n" + set);

anotherSet.addAll(set);

System.out.println("\nanotherSet is\n" + anotherSet);

}

}

**Accessing Items**

The TreeSet class has provides the following methods to access items.

E First( ) - Returns the first (smallest) element from the invoking TreeSet.

E last( ) - Returns the last (largest) element from the invoking TreeSet.

E higher(E obj) - Returns the largest element e such that e>obj. If it does not found returns null.

E lower(E obj) - Returns the largest element e such that e<obj. If it does not found returns null.

E ceiling(E obj) - Returns the smallest element e such that e>=obj. If it does not found returns null.

E floor(E obj) - Returns the largest element e such that e<=obj. If it does not found returns null.

SortedSet subSet(E fromElement, E toElement) - Returns a set of elements that lie between the given range which includes fromElement and excludes toElement.

NavigableSet subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) - Returns a set of elements that lie between the given range from the invoking TreeSet.

SortedSet tailSet(E fromElement) - Returns a set of elements that are greater than or equal to the specified fromElement from the invoking TreeSet.

NavigableSet tailSet(E fromElement, boolean inclusive) - Returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element from the invoking TreeSet.

SortedSet headSet(E fromElement) - Returns a set of elements that are smaller than or equal to the specified fromElement from the invoking TreeSet.

NavigableSet headSet(E fromElement, boolean inclusive) - Returns a set of elements that are smaller than or equal to (if, inclusive is true) the specified element from the invoking TreeSet.

Let's consider an example program to illustrate accessing items from a TreeSet.

Example

import java.util.\*;

public class TreeSetExample {

public static void main(String[] args) {

TreeSet set = new TreeSet();

Random num = new Random();

for(int i = 0; i < 10; i++)

set.add(num.nextInt(100));

System.out.println("\nset is\n" + set);

System.out.println("\nfirst() - " + set.first());

System.out.println("\nlast() - " + set.last());

System.out.println("\nhigher(20) - " + set.higher(20));

System.out.println("\nlower(20) - " + set.lower(20));

System.out.println("\nceiling(30) - " + set.ceiling(30));

System.out.println("\nfloor(30) - " + set.floor(30));

System.out.println("\nsubSet(10, 50)\n" + set.subSet(10, 50));

System.out.println("\nsubSet(10, false, 50, true)\n" + set.subSet(10, false, 50, true));

System.out.println("\nheadSet(20)\n" + set.headSet(20));

System.out.println("\nheadSet(20, false)\n" + set.headSet(20, false));

System.out.println("\ntailSet(20)\n" + set.tailSet(20));

System.out.println("\ntailSet(20, false)\n" + set.tailSet(20, false));

}

}

**Updating Items**

The TreeSet class has no methods to update or change items.

**Removing Items**

The TreeSet class has the following methods to remove items.

boolean remove(Object o) - Removes the specified element from the invoking TreeSet.

boolean removeAll(Collection c) - Removes all the elements those are in the specified collection from the invoking TreeSet.

boolean removeIf(Predicate p) - Removes all of the elements of the TreeSet collection that satisfy the given predicate.

boolean retainAll(Collection c) - Removes all the elements except those are in the specified collection from the invoking TreeSet.

E pollFirst( ) - Removes the first (smallest) element from the invoking TreeSet, and returns the same.

E pollLast( ) - Removes the last (largest) element from the invoking TreeSet, and returns the same.

void clear( ) - Removes all the elements from the TreeSet.

Let's consider an example program to illustrate removing items from the TreeSet.

Example

import java.util.\*;

public class TreeSetExample {

public static void main(String[] args) {

TreeSet set = new TreeSet();

TreeSet anotherSet = new TreeSet();

Random num = new Random();

for(int i = 0; i < 10; i++)

set.add(num.nextInt(100));

anotherSet.add(10);

anotherSet.add(20);

anotherSet.add(30);

System.out.println("\nset is\n" + set);

System.out.println("\nanotherSet is\n" + anotherSet);

set.remove(50);

System.out.println("\nset after remove(50) is\n" + set);

set.removeIf(n->n.equals(60));

System.out.println("\nset after removeIf(n->n.equals(60)) is\n" + set);

set.pollFirst();

System.out.println("\nset after pollFirst( ) is\n" + set);

set.pollLast();

System.out.println("\nset after pollLast( ) is\n" + set);

set.removeAll(anotherSet);

System.out.println("\nset after removeAll(anotherSet) is\n" + set);

set.retainAll(anotherSet);

System.out.println("\nset after retainAll(anotherSet) is\n" + set);

}

}

**Map Interfaces and Classes:**

**HashMap Class**

The HashMap class is a child class of AbstractMap, and it implements the Map interface. The HashMap is used to store the data in the form of key, value pair using hash table concept.

Key Properties of HashMap

HashMap is a child class of AbstractMap class.

HashMap implements the interfeaces Map, Cloneable, and Serializable.

HashMap stores data as a pair of key and value.

HashMap uses Hash table concept to store the data.

HashMap does not allow duplicate keys, but values may be repeated.

HashMap allows only one null key and multiple null values.

HashMap does not follow any oreder.

HashMap has the default capacity 16 entries.

Let's consider an example program to illustrate HashMap.

Example

import java.util.\*;

public class HashMapExample {

public static void main(String[] args) {

Scanner read = new Scanner(System.in);

HashMap employeeInfo = new HashMap();

HashMap contactInfo = new HashMap();

employeeInfo.put(1, "Raja");

employeeInfo.put(2, "Gouthami");

employeeInfo.put(3, "Heyansh");

employeeInfo.put(4, "Yamini");

employeeInfo.put(5, "ManuTej");

System.out.println("Employee Information\n" + employeeInfo);

System.out.println("\nPlease enter the ID and Contact number");

System.out.println("Employee IDs : " + employeeInfo.keySet());

System.out.print("Enter ID: ");

int id = read.nextInt();

System.out.print("Enter Contact Number: ");

long contactNo = read.nextLong();

if(employeeInfo.containsKey(id)) {

contactInfo.put(id, contactNo);

}

System.out.println("\n\nEmployee Contact Information\n");

System.out.println("ID - " + id);

System.out.println("Name - " + employeeInfo.get(id));

System.out.println("Contact Number - " + contactInfo.get(id));

}}

**TreeMap Class**

The TreeMap class is a child class of AbstractMap, and it implements the NavigableMap interface which is a child interface of SortedMap. The TreeMap is used to store the data in the form of key, value pair using a red-black tree concepts.

Key Properties of TreeMap

* TreeMap is a child class of AbstractMap class.
* TreeMap implements the NavigableMap interface which is a child interface of SortedMap interface.
* TreeMap stores data as a pair of key and value.
* TreeMap uses red-black tree concept to store the data.
* TreeMap does not allow duplicate keys, but values may be repeated.
* TreeMap does not allow null key, but allows null values.
* TreeMap follows the ascending oreder based on keys.
* Let's consider an example program to illustrate TreeMap.

Example

import java.util.\*;

public class HashMapExample {

public static void main(String[] args) {

Scanner read = new Scanner(System.in);

TreeMap employeeInfo = new TreeMap();

TreeMap contactInfo = new TreeMap();

employeeInfo.put(1, "Raja");

employeeInfo.put(4, "Gouthami");

employeeInfo.put(5, "Heyansh");

employeeInfo.put(3, "Yamini");

employeeInfo.put(2, "ManuTej");

System.out.println("Employee Information\n" + employeeInfo);

System.out.println("\nPlease enter the ID and Contact number");

System.out.println("Employee IDs : " + employeeInfo.keySet());

System.out.print("Enter ID: ");

int id = read.nextInt();

System.out.print("Enter Contact Number: ");

long contactNo = read.nextLong();

if(employeeInfo.containsKey(id)) {

contactInfo.put(id, contactNo);

}

System.out.println("\n\nEmployee Contact Information\n");

System.out.println("ID - " + id);

System.out.println("Name - " + employeeInfo.get(id));

System.out.println("Contact Number - " + contactInfo.get(id));

}

}

**The Legacy Classes and Interfaces:**

**Vector**

In java, the package java.util contains a class called Vector which implements the List interface.The Vector is similar to an ArrayList. Like ArrayList Vector also maintains the insertion order. But Vector is synchronized, due to this reason, it is rarly used in the non-thread application. It also lead to poor performance.

* The Vector is a class in the java.util package.
* The Vector implements List interface.
* The Vector is a legacy class.
* The Vector is synchronized.

The Vector class in java has the following constructor.

|  |  |
| --- | --- |
| S. No. | Constructor with Description |
| 1 | Vector( ) |
| It creates an empty Vector with default initail capacity of 10. |
| 2 | Vector(int initialSize) |
| It creates an empty Vector with specified initail capacity. |
| 3 | Vector(int initialSize, int incr) |
| It creates a vector whose initial capacity is specified by size and whose increment is specified by incr. |
| 4 | Vector(Collection c) |
| It creates a vector that contains the elements of collection c. |

Let's consider an example program to illustrate methods of Vector class.

Example

import java.util.\*;

public class VectorClassExample

{

public static void main(String[] args)

{

Vector list = new Vector();

list.add(10);

list.add(30);

list.add(0, 100);

list.addElement(50);

System.out.println("Vector => " + list);

System.out.println("get(2) => " + list.get(2));

System.out.println("firstElement() => " + list.firstElement());

System.out.println("indexOf(50) => " + list.indexOf(50));

System.out.println("contains(30) => " + list.contains(30));

System.out.println("capacity() => " + list.capacity());

System.out.println("size() => " + list.size());

System.out.println("isEmpty() => " + list.isEmpty());

}

}

**Stack**

In java, the package java.util contains a class called Stack which is a child class of Vector class. It implements the standard principle Last-In-First-Out of stack data structure.

The Stack has push method for inesrtion and pop method for deletion. It also has other utility methods.

In Stack, the elements are added to the top of the stack and removed from the top of the stack.

The Stack class in java has the following constructor.

|  |  |
| --- | --- |
| S. No. | Constructor with Description |
| 1 | Stack( ) |
| It creates an empty Stack. |

The Stack class in java has the following methods.

|  |  |
| --- | --- |
| S.No. | Methods with Description |
| 1 | Object push(Object element) |
| It pushes the element onto the stack and returns the same. |
| 2 | Object pop( ) |
| It returns the element on the top of the stack and removes the same. |
| 3 | int search(Object element) |
| If element found, it returns offset from the top. Otherwise, -1 is returned. |
| 4 | Object peek( ) |
| It returns the element on the top of the stack. |
| 5 | boolean empty() |
| It returns true if the stack is empty, otherwise returns false. |

Let's consider an example program to illustrate methods of Stack class.

Example

import java.util.\*;

public class StackClassExample {

public static void main(String[] args) {

Stack stack = new Stack();

Random num = new Random();

for(int i = 0; i < 5; i++)

stack.push(num.nextInt(100));

System.out.println("Stack elements => " + stack);

System.out.println("Top element is " + stack.peek());

System.out.println("Removed element is " + stack.pop());

System.out.println("Element 50 availability => " + stack.search(50));

System.out.println("Stack is empty? - " + stack.isEmpty());

}

}

**Hashtable**

In java, the package java.util contains a class called 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.

The Hashtable stores data as a pair of key and value. In the Hashtable, each key associates with a value. Any non-null object can be used as a key or as a value. We can use the key to retrieve the value back when needed.

The Hashtable class is no longer in use, it is obsolete. The alternate class is HashMap.

* The Hashtable class is a concrete class of Dictionary.
* The Hashtable class is synchronized.
* The Hashtable does no allow null key or value.
* The Hashtable has the initial default capacity 11.

The Hashtable class in java has the following constructors.

|  |  |
| --- | --- |
| S.No. | Constructor with Description |
| 1 | Hashtable( ) |
| It creates an empty hashtable with the default initial capacity 11. |
| 2 | Hashtable(int capacity) |
| It creates an empty hashtable with the specified initial capacity. |
| 3 | Hashtable(int capacity, float loadFactor) |
| It creates an empty hashtable with the specified initial capacity and loading factor. |
| 4 | Hashtable(Map m) |
| It creates a hashtable containing elements of Map m. |

The Hashtable class in java has the following methods.

|  |  |
| --- | --- |
| S. No. | Methods with Description |
| 1 | V put(K key, V value) |
| It inserts the specified key and value into the hash table. |
| 2 | void putAll(Map m)) |
| It inserts all the elements of Map m into the invoking Hashtable. |
| 3 | V putIfAbsent(K key, V value) |
| If the specified key is not already associated with a value associates it with the given value and returns null, else returns the current value. |
| 4 | V getOrDefault(Object key, V defaultValue) |
| It returns the value associated with given key; or defaultValue if the hashtable contains no mapping for the key. |
| 5 | V get(Object key) |
| It returns the value associated with the given key. |
| 6 | Enumeration keys() |
| Returns an enumeration of the keys of the hashtable. |
| 7 | Set keySet() |
| Returns a set view of the keys of the hashtable. |
| 8 | Collection values() |
| It returns a collection view of the values contained in the Hashtable. |
| 9 | Enumeration elements() |
| Returns an enumeration of the values of the hashtable. |
| 10 | Set entrySet() |
| It returns a set view of the mappings contained in the hashtable. |
| 11 | int hashCode() |
| It returns the hash code of the hashtable. |
| 12 | Object clone() |
| It returns a shallow copy of the Hashtable. |
| 13 | V remove(Object key) |
| It returns the value associated with given key and removes the same. |
| 14 | boolean remove(Object key, Object value) |
| It removes the specified values with the associated specified keys from the hashtable. |
| 15 | boolean contains(Object value) |
| It returns true if the specified value found within the hash table, else return false. |
| 16 | boolean containsValue(Object value) |
| It returns true if the specified value found within the hash table, else return false. |
| 17 | boolean containsKey(Object key) |
| It returns true if the specified key found within the hash table, else return false. |
| 18 | V replace(K key, V value) |
| It replaces the specified value for a specified key. |
| 19 | boolean replace(K key, V oldValue, V newValue) |
| It replaces the old value with the new value for a specified key. |
| 20 | void replaceAll(BiFunction function) |
| It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception. |
| 21 | void rehash() |
| It is used to increase the size of the hash table and rehashes all of its keys. |
| 22 | String toString() |
| It returns a string representation of the Hashtable object. |
| 23 | V merge(K key, V value, BiFunction remappingFunction) |
| If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value. |
| 24 | void forEach(BiConsumer action) |
| It performs the given action for each entry in the map until all entries have been processed or the action throws an exception. |
| 25 | boolean isEmpty( ) |
| It returns true if Hashtable has no elements; otherwise returns false. |
| 26 | int size( ) |
| It returns the total number of elements in the Hashtable. |
| 27 | void clear() |
| It is used to remove all the lements of a Hashtable. |
| 28 | boolean equals(Object o) |
| It is used to compare the specified Object with the Hashtable. |

Let's consider an example program to illustrate methods of Hashtable class.

Example

import java.util.\*;

public class HashtableExample {

public static void main(String[] args) {

Random num = new Random();

Hashtable table = new Hashtable();

//put(key, value)

for(int i = 1; i <= 5; i++)

table.put(i, num.nextInt(100));

System.out.println("Hashtable => " + table);

//get(key)

System.out.println("\nValue associated with key 3 => " + table.get(3));

System.out.println("Value associated with key 30 => " + table.get(30)); //keySet()

System.out.println("\nKeys => " + table.keySet());

//values()

System.out.println("\nValues => " + table.values());

//entrySet()

System.out.println("\nKey, Value pairs as a set => " + table.entrySet());

//hashCode()

System.out.println("\nHash code => " + table.hashCode());

//hashCode()

System.out.println("\nTotal number of elements => " + table.size());

//isEmpty()

System.out.println("\nEmpty status of Hashtable => " + table.isEmpty());

}}

**Other Utilities**

**Scanner**

The Scanner is a built-in class in java used for read the input from the user in java programming. The Scanner class is defined inside the java.util package.The Scanner class implements Iterator interface.

The Scanner class provides the easiest way to read input in a Java program.

The Scanner object breaks its input into tokens using a delimiter pattern, the default delimiter is whitespace.

The Scanner class in java has the following constructors.

|  |  |
| --- | --- |
| S.No. | Constructor with Description |
| 1 | Scanner(InputStream source) |
| It creates a new Scanner that produces values read from the specified input stream. |
| 2 | Scanner(InputStream source, String charsetName) |
| It creates a new Scanner that produces values read from the specified input stream. |
| 3 | Scanner(File source) |
| It creates a new Scanner that produces values scanned from the specified file. |
| 4 | Scanner(File source, String charsetName) |
| It creates a new Scanner that produces values scanned from the specified file. |
| 5 | Scanner(String source) |
| It creates a new Scanner that produces values scanned from the specified string. |
| 6 | Scanner(Readable source) |
| It creates a new Scanner that produces values scanned from the specified source. |
| 7 | Scanner(ReadableByteChannel source) |
| It creates a new Scanner that produces values scanned from the specified channel. |
| 8 | Scanner(ReadableByteChannel source, String charsetName) |
| It creates a new Scanner that produces values scanned from the specified channel. |

The Scanner class in java has the following methods.

|  |  |
| --- | --- |
| S.No. | Methods with Description |
| 1 | String next() |
| It reads the next complete token from the invoking scanner. |
| 2 | String next(Pattern pattern) |
| It reads the next token if it matches the specified pattern. |
| 3 | String next(String pattern) |
| It reads the next token if it matches the pattern constructed from the specified string. |
| 4 | boolean nextBoolean() |
| It reads a boolean value from the user. |
| 5 | byte nextByte() |
| It reads a byte value from the user. |
| 6 | double nextDouble() |
| It reads a double value from the user. |
| 7 | float nextFloat() |
| It reads a floating-point value from the user. |
| 8 | int nextInt() |
| It reads an integer value from the user. |
| 9 | long nextLong() |
| It reads a long value from the user. |
| 10 | short nextShort() |
| It reads a short value from the user. |
| 11 | String nextLine() |
| It reads a string value from the user. |
| 12 | boolean hasNext() |
| It returns true if the invoking scanner has another token in its input. |
| 13 | void remove() |
| It is used when remove operation is not supported by this implementation of Iterator. |
| 14 | void close() |
| It closes the invoking scanner. |

Let's consider an example program to illustrate methods of Scanner class.

Example

import java.util.Scanner;

public class ScannerClassExample {

public static void main(String[] args) {

Scanner read = new Scanner(System.in); // Input stream is used

System.out.print("Enter any name: ");

String name = read.next();

System.out.print("Enter your age in years: ");

int age = read.nextInt();

System.out.print("Enter your salary: ");

double salary = read.nextDouble();

System.out.print("Enter any message: ");

read = new Scanner(System.in);

String msg = read.nextLine();

System.out.println("\n------------------------------------------");

System.out.println("Hello, " + name);

System.out.println("You are " + age + " years old.");

System.out.println("You are earning Rs." + salary + " per month.");

System.out.println("Words from " + name + " - " + msg);}}

**StringTokenizer**

The StringTokenizer is a built-in class in java used to break a string into tokens. The StringTokenizer class is available inside the java.util package.The StringTokenizer class object internally maintains a current position within the string to be tokenized.

A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

The StringTokenizer class in java has the following constructor.

|  |  |
| --- | --- |
| S. No. | Constructor with Description |
| 1 | StringTokenizer(String str) |
| It creates StringTokenizer object for the specified string str with default delimeter. |
| 2 | StringTokenizer(String str, String delimeter) |
| It creates StringTokenizer object for the specified string str with specified delimeter. |
| 3 | StringTokenizer(String str, String delimeter, boolean returnValue) |
| It creates StringTokenizer object with specified string, delimeter and returnValue. |

The StringTokenizer class in java has the following methods.

|  |  |
| --- | --- |
| S.No. | Methods with Description |
| 1 | String nextToken() |
| It returns the next token from the StringTokenizer object. |
| 2 | String nextToken(String delimeter) |
| It returns the next token from the StringTokenizer object based on the delimeter. |
| 3 | Object nextElement() |
| It returns the next token from the StringTokenizer object. |
| 4 | boolean hasMoreTokens() |
| It returns true if there are more tokens in the StringTokenizer object. otherwise returns false. |
| 5 | boolean hasMoreElements() |
| It returns true if there are more tokens in the StringTokenizer object. otherwise returns false. |
| 6 | int countTokens() |
| It returns total number of tokens in the StringTokenizer object. |

Let's consider an example program to illustrate methods of StringTokenizer class.

Example

import java.util.StringTokenizer;

public class StringTokenizerExample {

public static void main(String[] args) {

String url = "www.btechsmartclass.com";

String title = "BTech Smart Class";

StringTokenizer tokens = new StringTokenizer(title);

StringTokenizer anotherTokens = new StringTokenizer(url, ".");

System.out.println("\nTotal tokens in title is " + tokens.countTokens());

System.out.print("Tokens in the title => ");

while(tokens.hasMoreTokens()) {

System.out.print(tokens.nextToken() + ", ");

}

System.out.println("\n\nTotal tokens in url is " + anotherTokens.countTokens());

System.out.println("Tokens in the url with delimeter (.) => ");

while(anotherTokens.hasMoreElements()) {

System.out.print(anotherTokens.nextElement() + ", ");

}

}

}

**Date**

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.The Date class implements Serializable, Cloneable and Comparable interface.Most of the constructors and methods of Date class has been deprecated after Calendar class introduced.

The Date class in java has the following constructor.

|  |  |
| --- | --- |
| S. No. | Constructor with Description |
| 1 | Date( ) |
| It creates a Date object that represents current date and time. |
| 2 | Date(long milliseconds) |
| It creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT. |
| 3 | Date(int year, int month, int date) – Depricated |
| It creates a date object with the specified year, month, and date. |
| 4 | Date(int year, int month, int date, int hrs, int min) – Depricated |
| It creates a date object with the specified year, month, date, hours, and minuts. |
| 5 | Date(int year, int month, int date, int hrs, int min, int sec) – Depricated |
| It creates a date object with the specified year, month, date, hours, minuts and seconds. |
| 5 | Date(String s) – Depricated |
| It creates a Date object and initializes it so that it represents the date and time indicated by the string s, which is interpreted as if by the parse(java.lang.String) method. |

The Date class in java has the following methods.

|  |  |
| --- | --- |
| S.No. | Methods with Description |
| 1 | long getTime() |
| It returns the time represented by this date object. |
| 2 | boolean after(Date date) |
| It returns true, if the invoking date is after the argumented date. |
| 3 | boolean before(Date date) |
| It returns true, if the invoking date is before the argumented date. |
| 4 | Date from(Instant instant) |
| It returns an instance of Date object from Instant date. |
| 5 | void setTime(long time) |
| It changes the current date and time to given time. |
| 6 | Object clone( ) |
| It duplicates the invoking Date object. |
| 7 | int compareTo(Date date) |
| It compares current date with given date. |
| 8 | boolean equals(Date date) |
| It compares current date with given date for equality. |
| 9 | int hashCode() |
| It returns the hash code value of the invoking date object. |
| 10 | Instant toInstant() |
| It converts current date into Instant object. |
| 11 | String toString() |
| It converts this date into Instant object. |

Let's consider an example program to illustrate methods of Date class.

Example

import java.time.Instant;

import java.util.Date;

public class DateClassExample {

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("after() => " + time.after(time) + " milliseconds");

System.out.println("before() => " + time.before(time) + " milliseconds");

System.out.println("hashCode() => " + time.hashCode());

}

}

**Random**

The Random is a built-in class in java used to generate a stream of pseudo-random numbers in java programming. The Random class is available inside the java.util package.The Random class implements Serializable, Cloneable and Comparable interface.

* The Random class is a part of java.util package.
* The Random class provides several methods to generate random numbers of type integer, double, long, float etc.
* The Random class is thread-safe.
* Random number generation algorithm works on the seed value. If not provided, seed value is created from system nano time.

The Random class in java has the following constructors.

|  |  |
| --- | --- |
| S.No. | Constructor with Description |
| 1 | Random() |
| It creates a new random number generator. |
| 2 | Random(long seedValue) |
| It creates a new random number generator using a single long seedValue. |

The Random class in java has the following methods.

|  |  |
| --- | --- |
| S.No. | Methods with Description |
| 1 | int next(int bits) |
| It generates the next pseudo-random number. |
| 2 | Boolean nextBoolean() |
| It generates the next uniformly distributed pseudo-random boolean value. |
| 3 | double nextDouble() |
| It generates the next pseudo-random double number between 0.0 and 1.0. |
| 4 | void nextBytes(byte[] bytes) |
| It places the generated random bytes into an user-supplied byte array. |
| 5 | float nextFloat() |
| It generates the next pseudo-random float number between 0.0 and 1.0.. |
| 6 | int nextInt() |
| It generates the next pseudo-random int number. |
| 7 | int nextInt(int n) |
| It generates the next pseudo-random integer value between zero and n. |
| 8 | long nextLong() |
| It generates the next pseudo-random, uniformly distributed long value. |
| 9 | double nextGaussian() |
| It generates the next pseudo-random Gaussian distributed double number with mean 0.0 and standard deviation 1.0. |
| 10 | void setSeed(long seedValue) |
| It sets the seed of the random number generator using a single long seedValue. |
| 11 | DoubleStream doubles() |
| It returns a stream of pseudo-random double values, each conforming between 0.0 and 1.0. |
| 12 | DoubleStream doubles(double start, double end) |
| It retruns an unlimited stream of pseudo-random double values, each conforming to the given start and end. |
| 13 | DoubleStream doubles(long streamSize) |
| It returns a stream producing the pseudo-random double values for the given streamSize number, each between 0.0 and 1.0. |
| 14 | DoubleStream doubles(long streamSize, double start, double end) |
| It returns a stream producing the given streamSizenumber of pseudo-random double values, each conforming to the given start and end. |
| 15 | IntStream ints() |
| It returns a stream of pseudo-random integer values. |
| 16 | IntStream ints(int start, int end) |
| It retruns an unlimited stream of pseudo-random integer values, each conforming to the given start and end. |
| 17 | IntStream ints(long streamSize) |
| It returns a stream producing the pseudo-random integer values for the given streamSize number. |
| 18 | IntStream ints(long streamSize, int start, int end) |
| It returns a stream producing the given streamSizenumber of pseudo-random integer values, each conforming to the given start and end. |
| 19 | LongStream longs() |
| It returns a stream of pseudo-random long values. |
| 20 | LongStream longs(long start, long end) |
| It retruns an unlimited stream of pseudo-random long values, each conforming to the given start and end. |
| 21 | LongStream longs(long streamSize) |
| It returns a stream producing the pseudo-random long values for the given streamSize number. |
| 22 | LongStream longs(long streamSize, long start, long end) |
| It returns a stream producing the given streamSizenumber of pseudo-random long values, each conforming to the given start and end. |

Let's consider an example program to illustrate methods of Random class.

Example

import java.util.Random;

public class RandomClassExample {

public static void main(String[] args) {

Random rand = new Random();

System.out.println("Integer random number - " + rand.nextInt());

System.out.println("Integer random number from 0 to 100 - " + rand.nextInt(100));

System.out.println("Boolean random value - " + rand.nextBoolean());

System.out.println("Double random number - " + rand.nextDouble());

System.out.println("Float random number - " + rand.nextFloat());

System.out.println("Long random number - " + rand.nextLong());

System.out.println("Gaussian random number - " + rand.nextGaussian());

}

}

**FILES**

**Stream classes:**

In java, the IO operations are performed using the concept of streams. Generally, a stream means a continuous flow of data. In java, a stream is a logical container of data that allows us to read from and write to it. A stream can be linked to a data source, or data destination, like a console, file or network connection by java IO system. The stream-based IO operations are faster than normal IO operations.

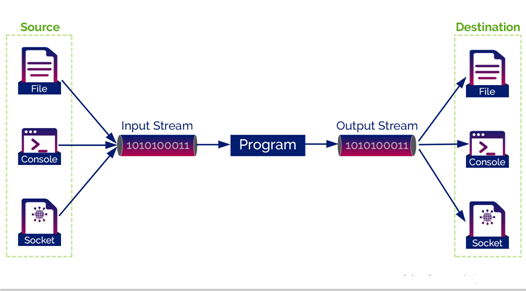
The Stream is defined in the java.io package.

To understand the functionality of java streams, look at the following picture.

In java, the stream-based IO operations are performed using two separate streams input stream and output stream. The input stream is used for input operations, and the output stream is used for output operations. The java stream is composed of bytes.

In Java, every program creates 3 streams automatically, and these streams are attached to the console.

* **System.out**: standard output stream for console output operations.
* **System.in**: standard input stream for console input operations.
* **System.err**: standard error stream for console error output operations.

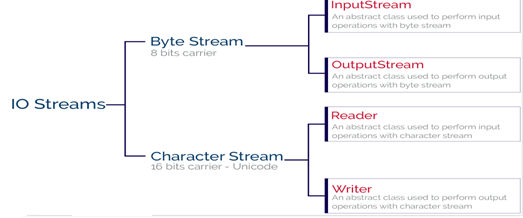


The Java streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects.

Java provides two types of streams, and they are as follows.

* **Byte Stream**
* **Character Stream**

The following picture shows how streams are categorized, and various built-in classes used by the java IO system.



Both character and byte streams essentially provides a convenient and efficient way to handle data streams in Java.

**Byte Stream Classes:**

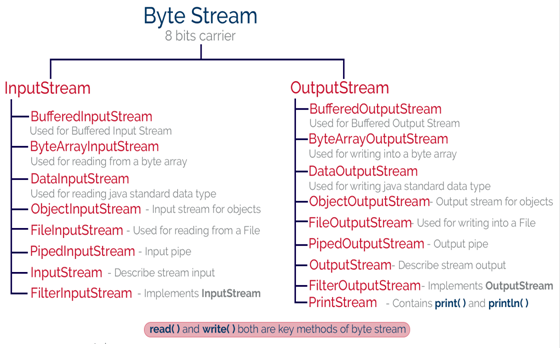
In java, the byte stream is an 8 bits carrier. The byte stream in java allows us to transmit 8 bits of data.

In Java 1.0 version all IO operations were byte oriented, there was no other stream (character stream).

The java byte stream is defined by two abstract classes, InputStream and OutputStream. The InputStream class used for byte stream based input operations, and the OutputStream class used for byte stream based output operations.

The InputStream and OutputStream classes have several concreate classes to perform various IO operations based on the byte stream.

The following picture shows the classes used for byte stream operations.



**InputStream class**

The InputStream class has defined as an abstract class, and it has the following methods which have implemented by its concrete classes.

|  |  |
| --- | --- |
| S.No. | Method with Description |
| 1 | int available() |
| It returns the number of bytes that can be read from the input stream. |
| 2 | int read() |
| It reads the next byte from the input stream. |
| 3 | int read(byte[] b) |
| It reads a chunk of bytes from the input stream and store them in its byte array, b. |
| 4 | void close() |
| It closes the input stream and also frees any resources connected with this input stream. |

**OutputStream class**

The OutputStream class has defined as an abstract class, and it has the following methods which have implemented by its concrete classes.

|  |  |
| --- | --- |
| S.No. | Method with Description |
| 1 | void write(int n) |
| It writes byte(contained in an int) to the output stream. |
| 2 | void write(byte[] b) |
| It writes a whole byte array(b) to the output stream. |
| 3 | void flush() |
| It flushes the output steam by forcing out buffered bytes to be written out. |
| 4 | void close() |
| It closes the output stream and also frees any resources connected with this output stream. |

Reading data using BufferedInputStream

We can use the BufferedInputStream class to read data from the console. The BufferedInputStream class use a method read( ) to read a value from the console, or file, or socket.

Let's look at an example code to illustrate reading data using BufferedInputStream.

Example 1 - Reading from console

import java.io.\*;

public class ReadingDemo {

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

BufferedInputStream read = new BufferedInputStream(System.in);

try {

System.out.print("Enter any character: ");

char c = (char)read.read();

System.out.println("You have entered '" + c + "'");

}

catch(Exception e) {

System.out.println(e);

}

finally {

read.close();}}}

**Writing data using BufferedOutputStream**

We can use the BufferedOutputStream class to write data into the console, file, socket. The BufferedOutputStream class use a method write( ) to write data.

Let's look at an example code to illustrate writing data into a file using BufferedOutputStream.

Example - Writing data into a file

import java.io.\*;

public class WritingDemo {

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

String data = "Java tutorials by BTech Smart Class";

BufferedOutputStream out = null;

try {

FileOutputStream fileOutputStream = new FileOutputStream(new File("C:\\Raja\\dataFile.txt"));

out = new BufferedOutputStream(fileOutputStream);

out.write(data.getBytes());

System.out.println("Writing data into a file is success!");

}

catch(Exception e) {

System.out.println(e);

}

finally {

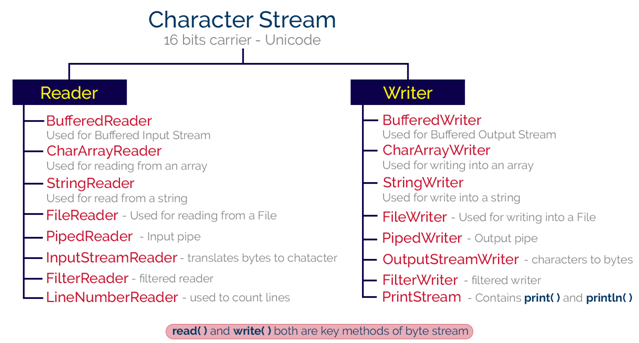
out.close();}}}

**Character Stream classes**

In java, when the IO stream manages 16-bit Unicode characters, it is called a character stream. The unicode set is basically a type of character set where each character corresponds to a specific numeric value within the given character set, and every programming language has a character set.In java, the character stream is a 16 bits carrier. The character stream in java allows us to transmit 16 bits of data.The character stream was introduced in Java 1.1 version. The charater stream. The java character stream is defined by two abstract classes, Reader and Writer. The Reader class used for character stream based input operations, and the Writer class used for charater stream based output operations.

The Reader and Writer classes have several concreate classes to perform various IO operations based on the character stream.

The following picture shows the classes used for character stream operations.



Reader class

The Reader class has defined as an abstract class, and it has the following methods which have implemented by its concrete classes.

|  |  |
| --- | --- |
| S.No. | Method with Description |
| 1 | int read() |
| It reads the next character from the input stream. |
| 2 | int read(char[] cbuffer) |
| It reads a chunk of charaters from the input stream and store them in its byte array, cbuffer. |
| 3 | int read(char[] cbuf, int off, int len) |
| It reads charaters into a portion of an array. |
| 4 | int read(CharBuffer target) |
| It reads charaters into into the specified character buffer. |
| 5 | String readLine() |
| It reads a line of text. A line is considered to be terminated by any oneof a line feed ('\n'),a carriage return ('\r'), or a carriage returnfollowed immediately by a linefeed. |
| 6 | boolean ready() |
| It tells whether the stream is ready to be read. |
| 7 | void close() |
| It closes the input stream and also frees any resources connected with this input stream. |

**Writer class**

The Writer class has defined as an abstract class, and it has the following methods which have implemented by its concrete classes.

|  |  |
| --- | --- |
| **S.No.** | **Method with Description** |
| 1 | void flush() |
| It flushes the output steam by forcing out buffered bytes to be written out. |
| 2 | void write(char[] cbuf) |
| It writes a whole array(cbuf) to the output stream. |
| 3 | void write(char[] cbuf, int off, int len) |
| It writes a portion of an array of characters. |
| 4 | void write(int c) |
| It writes single character. |
| 5 | void write(String str) |
| It writes a string. |
| 6 | void write(String str, int off, int len) |
| It writes a portion of a string. |
| 7 | Writer append(char c) |
| It appends the specified character to the writer. |

Reading data using BufferedReader

We can use the BufferedReader class to read data from the console. The BufferedInputStream class needs InputStreamReaderclass. The BufferedReader use a method read( ) to read a value from the console, or file, or socket.

Let's look at an example code to illustrate reading data using BufferedReader.

Example 1 - Reading from console

import java.io.\*;

public class ReadingDemo {

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

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader in = new BufferedReader(isr);

String name = "";

System.out.print("Please enter your name: ");

name = in.readLine();

System.out.println("Hello, " + name + "!");

}

}

Writing data using FileWriter

We can use the FileWriter class to write data into the file. The FileWriter class use a method write( ) to write data.

Let's look at an example code to illustrate writing data into a file using FileWriter.

Example - Writing data into a file

import java.io.\*;

public class WritingDemo {

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

Writer out = new FileWriter("C:\\Raja\\dataFile.txt");

String msg = "The sample data";

try {

out.write(msg);

System.out.println("Writing done!!!");

}

catch(Exception e) {

System.out.println(e);

}

finally {

out.close();

}

}

}

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

* **InPutStream** − The InputStream is used to read data from a source.
* **OutPutStream** − The OutputStream is used for writing data to a destination.



### Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file −

**Example**

import java.io.\*;

public class CopyFile {

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

FileInputStream in = null;

FileOutputStream out = null;

try {

in = new FileInputStream("input.txt");

out = new FileOutputStream("output.txt");

int c;

while ((c = in.read()) != -1) {

out.write(c);

}

}finally {

if (in != null) {

in.close();

}

if (out != null) {

out.close();

}

}

}

}

Now let's have a file **input.txt** with the following content −

This is test for copy file.

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following −

$javac CopyFile.java

$java CopyFile

### Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file −

**Example**

import java.io.\*;

public class CopyFile {

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

FileReader in = null;

FileWriter out = null;

try {

in = new FileReader("input.txt");

out = new FileWriter("output.txt");

int c;

while ((c = in.read()) != -1) {

out.write(c);

}

}finally {

if (in != null) {

in.close();

}

if (out != null) {

out.close();

}

}

}

}

Now let's have a file **input.txt** with the following content −

This is test for copy file.

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following −

$javac CopyFile.java

$java CopyFile