

ACEENGINEERING COLLEGE

UNIT IV

Collections in Java

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](#), [Vector](#), [LinkedList](#), [PriorityQueue](#), [HashSet](#), [LinkedHashSet](#), [TreeSet](#)).

What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

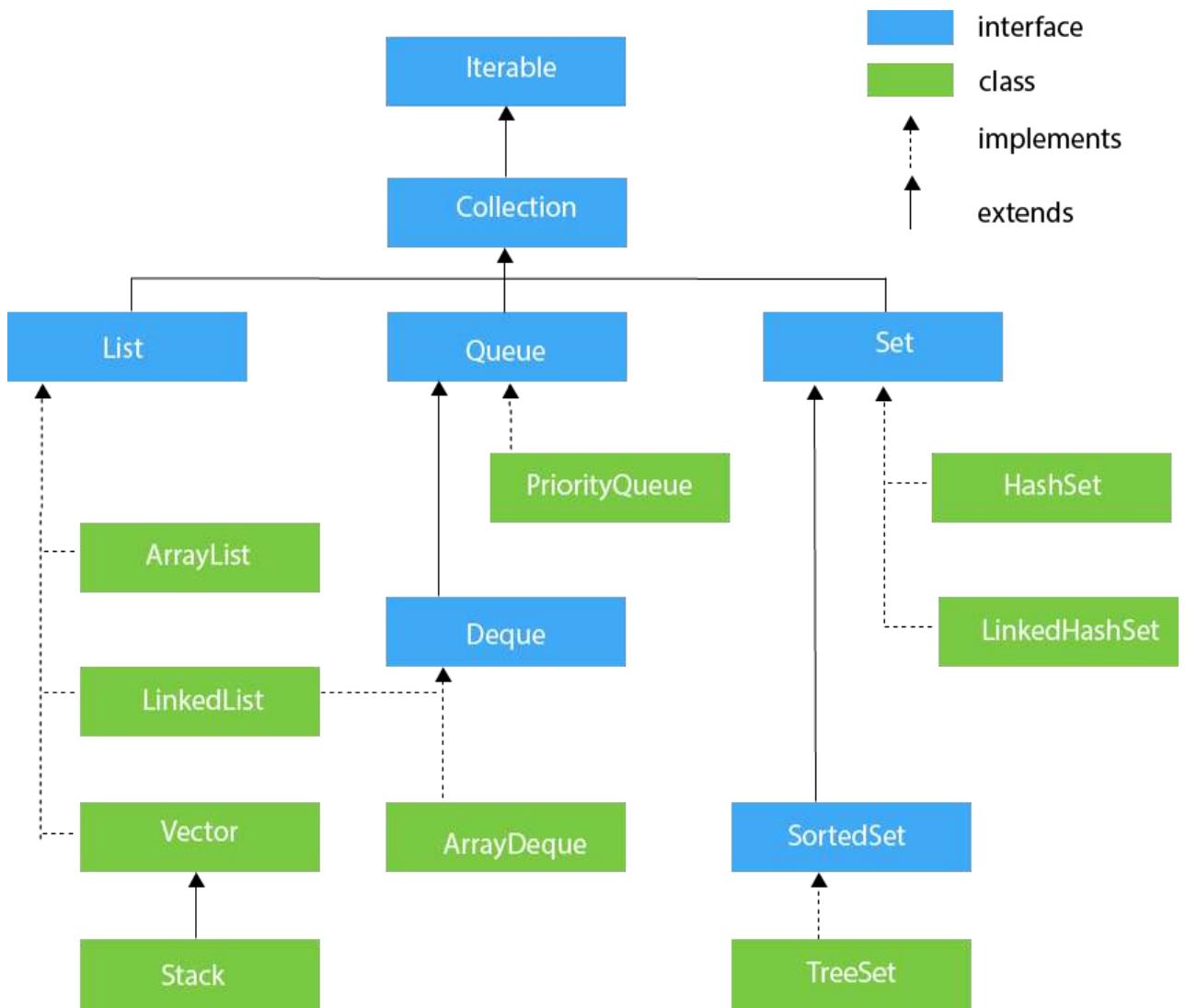
What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the [classes](#) and [interfaces](#) for the Collection framework.



Methods of Collection interface

No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.

4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T>T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.
15	default Stream<E>parallelStream()	It returns a possibly parallel Stream with the collection as its source.
16	default Stream<E>stream()	It returns a sequential Stream with the collection as its source.

17	default Spliterator<E>spliterator()	It generates a Spliterator over the specified elements in the collection.
18	public boolean equals(Object element)	It matches two collections.
19	public int hashCode()	It returns the hash code number of the collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in a forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

Iterable Interface

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

1. `Iterator<T> iterator()`

It returns the iterator over the elements of type T.

Collection Interface

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

The classes that implement the List interface are given below.

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 the ArrayList class can be randomly accessed. Consider the following example.

1. **import** java.util.*;
2. **class** TestJavaCollection1{
3. **public static void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();*//Creating arraylist*

```

5. list.add("Ravi");//Adding object in arraylist
6. list.add("Vijay");
7. list.add("Ravi");
8. list.add("Ajay");
9. //Traversing list through Iterator
10. Iterator itr=list.iterator();
11. while(itr.hasNext()){
12. System.out.println(itr.next());
13. }
14. }
15. }

```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

Method	Description
void <u>add</u> (int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean <u>add</u> (E e)	It is used to append the specified element at the end of a list.
boolean <u>addAll</u> (Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean <u>addAll</u> (int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void <u>clear</u> ()	It is used to remove all of the elements from this list.

void ensureCapacity(int requiredCapacity)	It is used to enhance the capacity of an ArrayList instance.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
<u>Iterator()</u>	
<u>listIterator()</u>	
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T>T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
Object clone()	It is used to return a shallow copy of an ArrayList.
boolean contains(Object o)	It returns true if the list contains the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.

boolean <u>remove</u> (Object o)	It is used to remove the first occurrence of the specified element.
boolean <u>removeAll</u> (Collection<?> c)	It is used to remove all the elements from the list.
boolean <u>removeIf</u> (Predicate<? super E> filter)	It is used to remove all the elements from the list that satisfies the given predicate.
protected void <u>removeRange</u> (int fromIndex, int toIndex)	It is used to remove all the elements lies within the given range.
void <u>replaceAll</u> (UnaryOperator<E> operator)	It is used to replace all the elements from the list with the specified element.
void <u>retainAll</u> (Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E <u>set</u> (int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void <u>sort</u> (Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.
Spliterator<E> <u>spliterator</u> ()	It is used to create spliterator over the elements in a list.
List<E> <u>subList</u> (int fromIndex, int toIndex)	It is used to fetch all the elements lies within the given range.
int <u>size</u> ()	It is used to return the number of elements present in the list.
void <u>trimToSize</u> ()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Change an Item

To modify an element, use the `set()` method and refer to the index number:

Example

```
cars.set(0, "Opel");
```

Remove an Item

To remove an element, use the `remove()` method and refer to the index number:

Example

```
cars.remove(0);
```

To remove all the elements in the `ArrayList`, use the `clear()` method:

Example

```
cars.clear();
```

ArrayList Size

To find out how many elements an `ArrayList` have, use the `size` method:

Example

```
cars.size();
```

Loop Through an ArrayList

Loop through the elements of an `ArrayList` with a `for` loop, and use the `size()` method to specify how many times the loop should run:

Example

```
public class Main{  
    public static void main(String[] args){  
        ArrayList<String> cars = new ArrayList<String>();  
        cars.add("Volvo");  
        cars.add("BMW");  
        cars.add("Ford");  
        cars.add("Mazda");  
        for(int i=0; i<cars.size(); i++){  
            System.out.println(cars.get(i));  
        }  
    }  
}
```

You can also loop through an `ArrayList` with the **for-each** loop:

Example

```
public class Main{  
    public static void main(String[] args){  
        ArrayList<String> cars = new ArrayList<String>();  
        cars.add("Volvo");  
        cars.add("BMW");  
        cars.add("Ford");  
    }  
}
```

```
cars.add("Mazda");

for(String i: cars){
    System.out.println(i);
}

}
```

Other Types

Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent [wrapper class](#): Integer. For other primitive types, use: Boolean for boolean, Character for char, Double for double, etc:

Example

Create an `ArrayList` to store numbers (add elements of type `Integer`):

```
import java.util.ArrayList;

public class Main{
    public static void main(String[] args){
        ArrayList<Integer> myNumbers = new ArrayList<Integer>();
        myNumbers.add(10);
        myNumbers.add(15);
        myNumbers.add(20);
        myNumbers.add(25);
        for(int i: myNumbers){
            System.out.println(i);
        }
    }
}
```

```
}
```

```
}
```

Sort an ArrayList

Another useful class in the `java.util` package is the `Collections` class, which include the `sort()` method for sorting lists alphabetically or numerically:

Example

Sort an ArrayList of Strings:

```
import java.util.ArrayList;  
  
import java.util.Collections;// Import the Collections class  
  
  
public class Main{  
    public static void main(String[] args){  
        ArrayList<String> cars = new ArrayList<String>();  
        cars.add("Volvo");  
        cars.add("BMW");  
        cars.add("Ford");  
        cars.add("Mazda");  
        Collections.sort(cars);// Sort cars  
        for(String i: cars){  
            System.out.println(i);  
        }  
    }  
}
```

Example

Sort an ArrayList of Integers:

```
import java.util.ArrayList;
import java.util.Collections;// Import the Collections class

public class Main{
    public static void main(String[] args){
        ArrayList<Integer> myNumbers = new ArrayList<Integer>();
        myNumbers.add(33);
        myNumbers.add(15);
        myNumbers.add(20);
        myNumbers.add(34);
        myNumbers.add(8);
        myNumbers.add(12);

        Collections.sort(myNumbers);// Sort myNumbers

        for(int i:myNumbers){
            System.out.println(i);
        }
    }
}
```

LinkedList

Method	Description
booleanadd(E e)	It is used to append the specified element to the end of a list.
void add(int index, E element)	It is used to insert the specified element at the specified position index in a list.
booleanaddAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
booleanaddAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
booleanaddAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void addFirst(E e)	It is used to insert the given element at the beginning of a list.
void addLast(E e)	It is used to append the given element to the end of a list.
void clear()	It is used to remove all the elements from a list.
Object clone()	It is used to return a shallow copy of an ArrayList.
booleancontains(Object o)	It is used to return true if a list contains a specified element.
Iterator<E>descendingIterator()	It is used to return an iterator over the elements in a deque in reverse sequential

	order.
E element()	It is used to retrieve the first element of a list.
E get(int index)	It is used to return the element at the specified position in a list.
E getFirst()	It is used to return the first element in a list.
E getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
ListIterator<E>listIterator(int index)	It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list.
boolean offer(E e)	It adds the specified element as the last element of a list.
boolean offerFirst(E e)	It inserts the specified element at the front of a list.
boolean offerLast(E e)	It inserts the specified element at the end of a list.
E peek()	It retrieves the first element of a list
E peekFirst()	It retrieves the first element of a list or returns null if a list is empty.
E peekLast()	It retrieves the last element of a list or returns null if a list is empty.

E poll()	It retrieves and removes the first element of a list.
E pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
E pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.
E pop()	It pops an element from the stack represented by a list.
void push(E e)	It pushes an element onto the stack represented by a list.
E remove()	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
booleanremove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst()	It removes and returns the first element from a list.
booleanremoveFirstOccurrence(Object o)	It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast()	It removes and returns the last element from a list.
booleanremoveLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.

Object[] toArray()	It is used to return an array containing all the elements in a list in proper sequence (from first to the last element).
<T>T[] toArray(T[] a)	It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.
int size()	It is used to return the number of elements in a list.

Java LinkedList Example

```

1. import java.util.*;
2. public class LinkedList1{
3.     public static void main(String args[]){
4.
5.         LinkedList<String> al=new LinkedList<String>();
6.         al.add("Ravi");
7.         al.add("Vijay");
8.         al.add("Ravi");
9.         al.add("Ajay");
10.
11.        Iterator<String> itr=al.iterator();
12.        while(itr.hasNext()){
13.            System.out.println(itr.next());
14.        }
15.    }
16.}
```

Output: Ravi
Vijay
Ravi
Ajay

Java LinkedList example to add elements

Here, we see different ways to add elements.

```

1. import java.util.*;
2. public class LinkedList2{
3.     public static void main(String args[]){
4.         LinkedList<String> ll=new LinkedList<String>();
5.         System.out.println("Initial list of elements: "+ll);
6.         ll.add("Ravi");
7.         ll.add("Vijay");
8.         ll.add("Ajay");
9.         System.out.println("After invoking add(E e) method: "+ll);
10.        //Adding an element at the specific position
11.        ll.add(1, "Gaurav");
12.        System.out.println("After invoking add(int index, E element) method:
13. "+ll);
14.        LinkedList<String> ll2=new LinkedList<String>();
15.        ll2.add("Sonoo");
16.        ll2.add("Hanumat");
17.        //Adding second list elements to the first list
18.        ll.addAll(ll2);
19.        System.out.println("After invoking addAll(Collection<? extends E> c)
20. method: "+ll);
21.        LinkedList<String> ll3=new LinkedList<String>();
22.        ll3.add("John");
23.        ll3.add("Rahul");
24.        //Adding second list elements to the first list at specific position
25.        ll.addAll(1, ll3);
26.        System.out.println("After invoking addAll(int index, Collection<? exte
27. nds E> c) method: "+ll);
28.        //Adding an element at the first position
29.        ll.addFirst("Lokesh");
30.        System.out.println("After invoking addFirst(E e) method: "+ll);
31.        //Adding an element at the last position
32.        ll.addLast("Harsh");
33.        System.out.println("After invoking addLast(E e) method: "+ll);
34.    }
35. }
```

Initial list of elements: []

```
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay,
Ajay]
After invoking addAll(Collection<? extends E> c) method:
[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addFirst(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addLast(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat, Harsh]
```

Java LinkedList example to remove elements

Here, we see different ways to remove an element.

```
1. import java.util.*;
2. public class LinkedList3 {
3.
4.     public static void main(String [] args)
5.     {
6.         LinkedList<String> ll=new LinkedList<String>();
7.         ll.add("Ravi");
8.         ll.add("Vijay");
9.         ll.add("Ajay");
10.        ll.add("Anuj");
11.        ll.add("Gaurav");
12.        ll.add("Harsh");
13.        ll.add("Virat");
14.        ll.add("Gaurav");
15.        ll.add("Harsh");
16.        ll.add("Amit");
17.        System.out.println("Initial list of elements: "+ll);
18.        //Removing specific element from arraylist
19.        ll.remove("Vijay");
20.        System.out.println("After invoking remove(object) method: "+ll);
21.        //Removing element on the basis of specific position
22.        ll.remove(0);
23.        System.out.println("After invoking remove(index) method: "+ll);
24.        LinkedList<String> ll2=new LinkedList<String>();
25.        ll2.add("Ravi");
```

```

26.         ll2.add("Hanumat");
27.         // Adding new elements to arraylist
28.         ll.addAll(ll2);
29.         System.out.println("Updated list : "+ll);
30.         //Removing all the new elements from arraylist
31.         ll.removeAll(ll2);
32.         System.out.println("After invoking removeAll() method: "+ll);
33.         //Removing first element from the list
34.         ll.removeFirst();
35.         System.out.println("After invoking removeFirst() method: "+ll);
36.         //Removing first element from the list
37.         ll.removeLast();
38.         System.out.println("After invoking removeLast() method: "+ll);
39.         //Removing first occurrence of element from the list
40.         ll.removeFirstOccurrence("Gaurav");
41.         System.out.println("After invoking removeFirstOccurrence() method
: "+ll);
42.         //Removing last occurrence of element from the list
43.         ll.removeLastOccurrence("Harsh");
44.         System.out.println("After invoking removeLastOccurrence() method
: "+ll);
45.
46.         //Removing all the elements available in the list
47.         ll.clear();
48.         System.out.println("After invoking clear() method: "+ll);
49.     }
50. }
```

```

Initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav, Harsh,
Virat, Gaurav, Harsh, Amit]
After invoking remove(index) method: [Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit]
Updated list : [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit,
Ravi, Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit]
After invoking removeFirst() method: [Gaurav, Harsh, Virat, Gaurav, Harsh,
Amit]
After invoking removeLast() method: [Gaurav, Harsh, Virat, Gaurav, Harsh]
After invoking removeFirstOccurrence() method: [Harsh, Virat, Gaurav,
Harsh]
After invoking removeLastOccurrence() method: [Harsh, Virat, Gaurav]
```

```
After invoking clear() method: []
```

Java LinkedList Example to reverse a list of elements

```
1. import java.util.*;
2. public class LinkedList4{
3.     public static void main(String args[]){
4.
5.         LinkedList<String> ll=new LinkedList<String>();
6.         ll.add("Ravi");
7.         ll.add("Vijay");
8.         ll.add("Ajay");
9.         //Traversing the list of elements in reverse order
10.        Iterator i=ll.descendingIterator();
11.        while(i.hasNext())
12.        {
13.            System.out.println(i.next());
14.        }
15.
16.    }
17.}
```

```
Output: Ajay
Vijay
Ravi
```

Java LinkedList Example: Book

```
1. import java.util.*;
2. class Book {
3.     int id;
4.     String name,author,publisher;
5.     int quantity;
6.     public Book(int id, String name, String author, String publisher, int quantity
7.     ) {
8.         this.id = id;
9.         this.name = name;
10.        this.author = author;
```

```

10.    this.publisher = publisher;
11.    this.quantity = quantity;
12.}
13.}
14.public class LinkedListExample {
15.public static void main(String[] args) {
16.    //Creating list of Books
17.    List<Book> list=new LinkedList<Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications & Networking","Forouzan"
    , "Mc Graw Hill",4);
21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to list
23.    list.add(b1);
24.    list.add(b2);
25.    list.add(b3);
26.    //Traversing list
27.    for(Book b:list){
28.        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.
        quantity);
29.    }
30.}
31.}

```

Output:

```

101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6

```

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. In LinkedList, the manipulation is fast because no shifting is required.

Consider the following example.

```

1. import java.util.*;
2. public class TestJavaCollection2{
3.     public static void main(String args[]){
4.         LinkedList<String> al=new LinkedList<String>();
5.         al.add("Ravi");
6.         al.add("Vijay");
7.         al.add("Ravi");
8.         al.add("Ajay");
9.         Iterator<String> itr=al.iterator();
10.        while(itr.hasNext()){
11.            System.out.println(itr.next());
12.        }
13.    }
14. }
```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

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

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

ArrayList	LinkedList
1) ArrayList internally uses a dynamic array to store the elements.	LinkedList internally uses a doubly linked list to store the elements.
2) Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.

4) ArrayList is **better for storing and accessing** data.

LinkedList is **better for manipulating** data.

Vector

Constructors:

1. Vector(): Creates a default vector of the initial capacity is 10.

Vector<E> v = new Vector<E>();

2. Vector(int size): Creates a vector whose initial capacity is specified by size.

Vector<E> v = new Vector<E>(int size);

3. Vector(int size, int incr): Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time that a vector is resized upward.

Vector<E> v = new Vector<E>(int size, int incr);

4. Vector(Collection c): Creates a vector that contains the elements of collection c.

Vector<E> v = new Vector<E>(Collection c);

Example: The following implementation demonstrates **how to create and use a Vector**.

- Java

```
// Java program to demonstrate the
```

```
// working of Vector
```

```
importjava.io.*;
importjava.util.*;

classVectorExample {

    publicstaticvoidmain(String[] args)
    {
        // Size of the
        // Vector
        intn = 5;

        // Declaring the Vector with
        // initial size n
        Vector<Integer> v = newVector<Integer>(n);

        // Appending new elements at
        // the end of the vector
        for(inti = 1; i<= n; i++)
            v.add(i);
```

```
// Printing elements

System.out.println(v);

// Remove element at index 3

v.remove(3);

// Displaying the vector

// after deletion

System.out.println(v);

// Printing elements one by one

for(int i = 0; i < v.size(); i++)

    System.out.print(v.get(i) + " ");

}

}
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Performing Various Operations on Vector class in Java

1. Adding Elements: In order to add the elements to the Vector, we use the [add\(\)](#) method. This method is overloaded to perform multiple operations based on different parameters. They are:

- **add(Object):** This method is used to add an element at the end of the Vector.
- **add(int index, Object):** This method is used to add an element at a specific index in the Vector.

- Java

```
// Java code for adding the

// elements in Vector Class

importjava.util.*;
importjava.io.*;

classAddElementsToVector {

    publicstaticvoidmain(String[] arg)
    {
        // create default vector

        Vector v1 = newVector();

        // Add elements using add() method

        v1.add(1);
```

```

    v1.add(2);

    v1.add("geeks");

    v1.add("forGeeks");

    v1.add(3);

    // print the vector to the console

    System.out.println("Vector v1 is "+ v1);

// create generic vector

Vector<Integer> v2 = newVector<Integer>();

v2.add(1);

v2.add(2);

v2.add(3);

System.out.println("Vector v2 is "+ v2);

}

}

```

Output:

Vector v1 is [1, 2, geeks, forGeeks, 3]

Vector v2 is [1, 2, 3]

2. Changing Elements: After adding the elements, if we wish to change the element, it can be done using the [set\(\)](#) method. Since a Vector is indexed, the element which we wish to change is referenced by the index of the element. Therefore, this method takes an index and the updated element which needs to be inserted at that index.

- Java

```
// Java code to change the  
  
// elements in vector class  
  
  
  
import java.util.*;  
  
  
  
  
public class UpdatingVector {  
  
  
  
  
    public static void main(String args[])  
  
    {  
  
        // Creating an empty Vector  
  
        Vector<Integer> vec_tor = new Vector<Integer>();  
  
  
  
  
        // Use add() method to add elements in the vector  
  
        vec_tor.add(12);  
  
        vec_tor.add(23);  
  
        vec_tor.add(22);  
  
        vec_tor.add(10);  
  
        vec_tor.add(20);
```

```

// Displaying the Vector

System.out.println("Vector: "+ vec_tor);

// Using set() method to replace 12 with 21

System.out.println("The Object that is replaced is:
"
+ vec_tor.set(0, 21));

// Using set() method to replace 20 with 50

System.out.println("The Object that is replaced is:
"
+ vec_tor.set(4, 50));

// Displaying the modified vector

System.out.println("The new Vector is:"+ vec_tor);

}
}

```

Output

```

Vector: [12, 23, 22, 10, 20]
The Object that is replaced is: 12
The Object that is replaced is: 20
The new Vector is:[21, 23, 22, 10, 50]

```

3. Removing Elements: In order to remove an element from a Vector, we can use the [remove\(\)](#) method. This method is overloaded to perform multiple operations based on different parameters. They are:

- **remove(Object):** This method is used to simply remove an object from the Vector. If there are multiple such objects, then the first occurrence of the object is removed.
- **remove(int index):** Since a Vector is indexed, this method takes an integer value which simply removes the element present at that specific index in the Vector. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

- Java

```
// Java code illustrating the removal  
  
// of elements from vector  
  
import java.util.*;  
  
import java.io.*;  
  
class RemovingElementsFromVector {  
  
    public static void main(String[] args)  
    {  
  
        // create default vector of capacity 10  
  
        Vector v = new Vector();  
  
        // Add elements using add() method  
  
        v.add("A");  
        v.add("B");  
        v.add("C");  
        v.add("D");  
        v.add("E");  
        v.add("F");  
        v.add("G");  
        v.add("H");  
        v.add("I");  
        v.add("J");  
  
        System.out.println("Initial Vector: " + v);  
  
        // Remove elements from vector  
  
        v.remove("B");  
        v.remove(3);  
  
        System.out.println("Vector after removal: " + v);  
    }  
}
```

```

        v.add(1);

        v.add(2);

        v.add("Geeks");

        v.add("forGeeks");

        v.add(4);

        // removing first occurrence element at 1

        v.remove(1);

        // checking vector

        System.out.println("after removal: "+ v);

    }

}

```

Output:

after removal: [1, Geeks, forGeeks, 4]

4. Iterating the Vector: There are multiple ways to iterate through the Vector. The most famous ways are by using the basic for loop in combination with a [get\(\)](#) method to get the element at a specific index and the [advanced for loop](#).

- Java

```
// Java program to iterate the elements
```

```
// in a Vector

import java.util.*;

public class IteratingVector {

    public static void main(String args[])
    {
        // create an instance of vector
        Vector<String> v = new Vector<>();

        // Add elements using add() method
        v.add("Geeks");
        v.add("Geeks");
        v.add(1, "For");

        // Using the Get method and the
        // for loop
        for(int i = 0; i < v.size(); i++) {
            System.out.print(v.get(i) + " ");
        }
    }
}
```

```

    }

    System.out.println();

    // Using the for each loop

    for(String str : v)

        System.out.print(str + " ");

    }

}

```

Output

Geeks For Geeks

Geeks For Geeks

Important points regarding Increment of vector capacity:

If the increment is specified, Vector will expand according to it in each allocation cycle but if the increment is not specified then the vector's capacity gets doubled in each allocation cycle. Vector defines three protected data member:

- ***int capacityIncrement:*** Contains the increment value.
- ***int elementCount:*** Number of elements currently in vector stored in it.
- ***Object elementData[]:*** Array that holds the vector is stored in it.

Common Errors in Declaration of Vectors

- Vector throws an **IllegalArgumentException** if the InitialSize of the vector defined is negative.
- If the specified collection is null, It throws **NullPointerException**.

Methods in Vector Class

METHOD	DESCRIPTION
<u>add(E e)</u>	Appends the specified element to the end of this Vector.
<u>add(int index, E element)</u>	Inserts the specified element at the specified position in this Vector.
<u>addAll(Collection<? extends E> c)</u>	Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
<u>addAll(int index, Collection<? extends E> c)</u>	Inserts all of the elements in the specified Collection into this Vector at the specified position.
<u>addElement(E obj)</u>	Adds the specified component to the end of this vector, increasing its size by one.
<u>capacity()</u>	Returns the current capacity of this vector.
<u>clear()</u>	Removes all of the elements from this Vector.
<u>clone()</u>	Returns a clone of this vector.
<u>contains(Object o)</u>	Returns true if this vector contains the specified element.
<u>containsAll(Collection<?> c)</u>	Returns true if this Vector contains all of the elements in the specified Collection.
<u>copyInto(Object[] anArray)</u>	Copies the components of this vector into the specified array.

METHOD	DESCRIPTION
<u>elementAt(int index)</u>	Returns the component at the specified index.
<u>elements()</u>	Returns an enumeration of the components of this vector.
<u>ensureCapacity(int minCapacity)</u>	Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
<u>equals(Object o)</u>	Compares the specified Object with this Vector for equality.
<u>firstElement()</u>	Returns the first component (the item at index 0) of this vector.
<u>forEach(Consumer<? super E> action)</u>	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
<u>get(int index)</u>	Returns the element at the specified position in this Vector.
<u>hashCode()</u>	Returns the hash code value for this Vector.
<u>indexOf(Object o)</u>	Returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
<u>indexOf(Object o, int index)</u>	Returns the index of the first occurrence of the specified element in this vector, searching forwards from the index, or returns -1 if the element is not found.

METHOD	DESCRIPTION
<u>insertElementAt(E obj, int index)</u>	Inserts the specified object as a component in this vector at the specified index.
<u>isEmpty()</u>	Tests if this vector has no components.
<u>iterator()</u>	Returns an iterator over the elements in this list in a proper sequence.
<u>lastElement()</u>	Returns the last component of the vector.
<u>lastIndexOf(Object o)</u>	Returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
<u>lastIndexOf(Object o, int index)</u>	Returns the index of the last occurrence of the specified element in this vector, searching backward from the index, or returns -1 if the element is not found.
<u>listIterator()</u>	Returns a list iterator over the elements in this list (in proper sequence).
<u>listIterator(int index)</u>	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

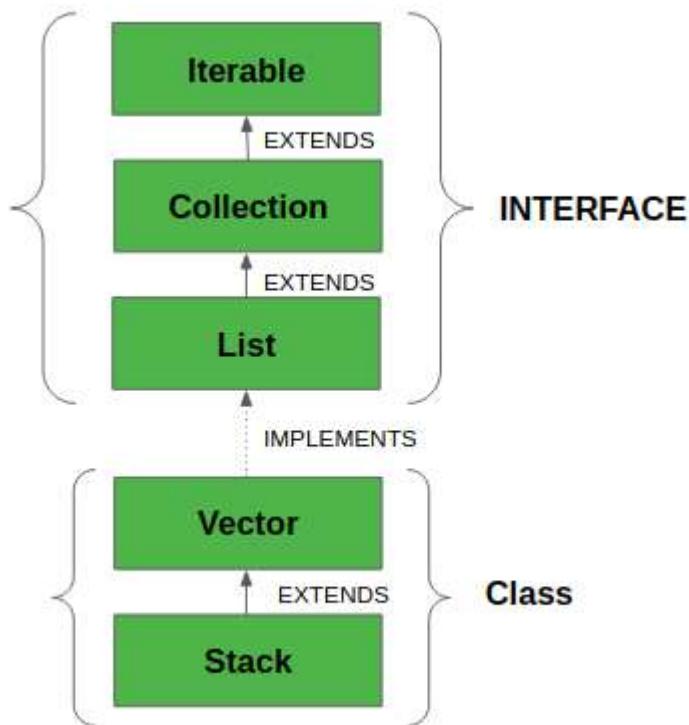
Consider the following example.

```
1. import java.util.*;
2. public class TestJavaCollection3{
3.     public static void main(String args[]){
4.         Vector<String> v=new Vector<String>();
5.         v.add("Ayush");
6.         v.add("Amit");
7.         v.add("Ashish");
8.         v.add("Garima");
9.         Iterator<String> itr=v.iterator();
10.        while(itr.hasNext()){
11.            System.out.println(itr.next());
12.        }
13.    }
14. }
```

Output:

```
Ayush
Amit
Ashish
Garima
```

Stack



In order to create a stack, we must import **java.util.stack** package and use the **Stack()** constructor of this class. The below example creates an empty Stack.

Stack<E> stack = new Stack<E>();

Here E is the type of Object.

Example:

- Java

```
// Java code for stack implementation
```

```
import java.io.*;
```

```
import java.util.*;
```

```
classTest
```

```
{

// Pushing element on the top of the stack

static void stack_push(Stack<Integer> stack)

{

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

    {

        stack.push(i);

    }

}

// Popping element from the top of the stack

static void stack_pop(Stack<Integer> stack)

{

    System.out.println("Pop Operation:");

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

    {

        Integer y = (Integer) stack.pop();

        System.out.println(y);

    }

}
```

```
// Displaying element on the top of the stack

static void stack_peek(Stack<Integer> stack)

{

    Integer element = (Integer) stack.peek();

    System.out.println("Element on stack top: "+

element);

}

// Searching element in the stack

static void stack_search(Stack<Integer> stack, int element)

{

    Integer pos = (Integer) stack.search(element);

    if(pos == -1)

        System.out.println("Element not found");

    else

        System.out.println("Element is found at

position: "+ pos);

}
```

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

{
    Stack<Integer> stack = newStack<Integer>();

    stack_push(stack);

    stack_pop(stack);

    stack_push(stack);

    stack_peek(stack);

    stack_search(stack, 2);

    stack_search(stack, 6);

}

}
```

Output:

Pop Operation:

```
4  
3  
2  
1  
0
```

```
Element on stack top: 4
```

```
Element is found at position: 3
```

```
Element not found
```

Performing various operations on Stack class

1. Adding Elements: In order to add an element to the stack, we can use the *push()* method. This **push()** operation place the element at the top of the stack.

- Java

```
// Java program to add the  
  
// elements in the stack  
  
import java.io.*;  
  
import java.util.*;  
  
  
  
class StackDemo {  
  
    // Main Method  
  
    public static void main(String[] args)  
    {  
  
        // Default initialization of Stack  
  
        Stack stack1 = new Stack();  
  
  
  
        // Initialization of Stack  
  
        // using Generics  
  
        Stack<String> stack2 = new Stack<String>();
```

```

// pushing the elements

stack1.push(4);

stack1.push("All");

stack1.push("Geeks");

stack2.push("Geeks");

stack2.push("For");

stack2.push("Geeks");

// Printiting the Stack Elements

System.out.println(stack1);

System.out.println(stack2);

}

}

```

Output:

[4, All, Geeks]

[Geeks, For, Geeks]

2. Accessing the Element: To retrieve or fetch the first element of the Stack or the element present at the top of the Stack, we can use [peek\(\)](#) method. The element retrieved does not get deleted or removed from the Stack.

- Java


```

    // Displaying the Stack

    System.out.println("Initial Stack: "+ stack);

    // Fetching the element at the head of the Stack

    System.out.println("The element at the top of the"
        + " stack is: "+ stack.peek());

    // Displaying the Stack after the Operation

    System.out.println("Final Stack: "+ stack);

}

}

```

Output:

Initial Stack: [Welcome, To, Geeks, For, Geeks]

The element at the top of the stack is: Geeks

Final Stack: [Welcome, To, Geeks, For, Geeks]

3. Removing Elements: To pop an element from the stack, we can use the [pop\(\)](#) method. The element is popped from the top of the stack and is removed from the same.

- Java

```

// Java program to demonstrate the removing

// of the elements from the stack

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

```

```
public class StackDemo {  
  
    public static void main(String args[])  
  
    {  
  
        // Creating an empty Stack  
  
        Stack<Integer> stack = new Stack<Integer>();  
  
  
  
  
        // Use add() method to add elements  
  
        stack.push(10);  
  
        stack.push(15);  
  
        stack.push(30);  
  
        stack.push(20);  
  
        stack.push(5);  
  
  
  
  
        // Displaying the Stack  
  
        System.out.println("Initial Stack: " + stack);  
  
  
  
  
        // Removing elements using pop() method  
  
        System.out.println("Popped element: "  
                          + stack.pop());  
  
        System.out.println("Popped element: "
```

```

        + stack.pop()));

// Displaying the Stack after pop operation

System.out.println("Stack after pop operation "
+ stack);

}

}

```

Output:

```

Initial Stack: [10, 15, 30, 20, 5]
Popped element: 5
Popped element: 20
Stack after pop operation [10, 15, 30]

```

METHOD	DESCRIPTION
<u>empty()</u>	It returns true if nothing is on the top of the stack. Else, returns false.

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like booleanpush(), boolean peek(), boolean push(object o), which defines its properties.

Consider the following example.

1. **import** java.util.*;
2. **public class** TestJavaCollection4{
3. **public static void** main(String args[]){
4. Stack<String> stack = **new** Stack<String>();
5. stack.push("Ayush");

```
6. stack.push("Garvit");
7. stack.push("Amit");
8. stack.push("Ashish");
9. stack.push("Garima");
10.stack.pop();
11.Iterator<String> itr=stack.iterator();
12.while(itr.hasNext()){
13.System.out.println(itr.next());
14.}
15.}
16.}
```

Output:

```
Ayush
Garvit
Amit
Ashish
```

Queue Interface

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

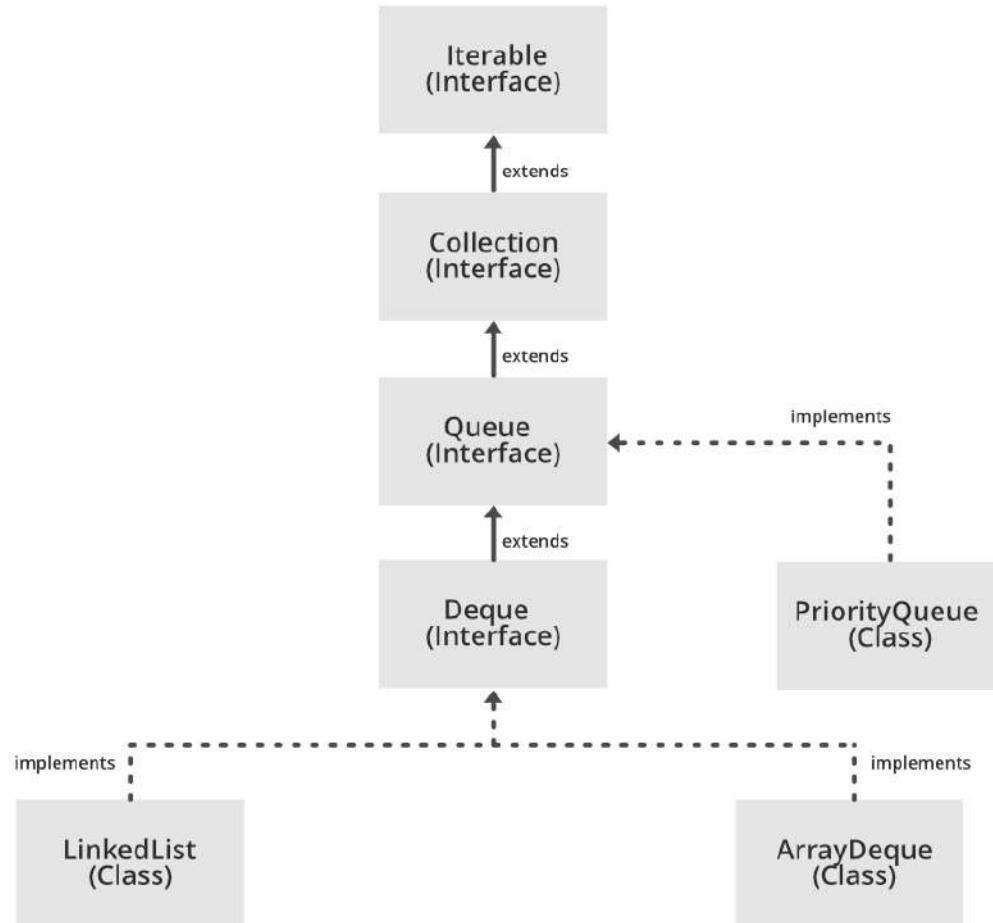
1. Queue<String> q1 = **new** PriorityQueue();
2. Queue<String> q2 = **new** ArrayDeque();

There are various classes that implement the Queue interface, some of them are given below.

PriorityQueue

PriorityQueue in Java

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a [Queue](#) follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play. The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.



In the below priority queue, an element with maximum ASCII value will have the highest priority.

Priority Queue

Initial Queue = {}

Operation	Return value	Queue Content
insert (C)		C
insert (O)		C O
insert (D)		C O D
remove max	O	C D
insert (I)		C D I
insert (N)		C D I N
remove max	N	C D I
insert (G)		C D I G

DG

Declaration:

```
public class PriorityQueue<E> extends AbstractQueue<E> implements  
Serializable
```

where E is the type of elements held in this queue

The class implements **Serializable**, **Iterable<E>**, **Collection<E>**, **Queue<E>** interfaces

Few important points on Priority Queue are as follows:

- PriorityQueue doesn't permit null.
- We can't create PriorityQueue of Objects that are non-comparable.
- PriorityQueue are unbound queues.
- The head of this queue is the least element with respect to the specified ordering. If multiple elements are tied for least value, the head is one of those elements — ties are broken arbitrarily.
- Since PriorityQueue is not thread-safe, so java provides [PriorityBlockingQueue](#) class that implements the [BlockingQueue](#) interface to use in java multithreading environment.
- The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
- It provides O(log(n)) time for add and poll methods.

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Consider the following example.

```
1. import java.util.*;
2. public class TestJavaCollection5{
3.     public static void main(String args[]){
4.         PriorityQueue<String> queue=new PriorityQueue<String>();
5.         queue.add("Amit Sharma");
6.         queue.add("Vijay Raj");
7.         queue.add("JaiShankar");
8.         queue.add("Raj");
9.         System.out.println("head:"+queue.element());
10.        System.out.println("head:"+queue.peek());
11.        System.out.println("iterating the queue elements:");
12.        Iterator itr=queue.iterator();
13.        while(itr.hasNext()){
14.            System.out.println(itr.next());
15.        }
16.        queue.remove();
17.        queue.poll();
18.        System.out.println("after removing two elements:");
19.        Iterator<String> itr2=queue.iterator();
20.        while(itr2.hasNext()){
21.            System.out.println(itr2.next());
22.        }
23.    }
24. }
```

Output:

```
head:Amit Sharma
head:Amit Sharma
iterating the queue elements:
Amit Sharma
Raj
JaiShankar
Vijay Raj
after removing two elements:
Raj
Vijay Raj
```

Deque Interface

Constructors of ArrayDeque class

1. ArrayDeque(): This constructor is used to create an empty ArrayDeque and by default holds an initial capacity to hold 16 elements.

ArrayDeque<E>dq = new ArrayDeque<E>();

2. ArrayDeque(Collection<? extends E> c): This constructor is used to create an ArrayDeque containing all the elements the same as that of the specified collection.

ArrayDeque<E>dq = new ArrayDeque<E>(Collection col);

3. ArrayDeque(int numofElements): This constructor is used to create an empty ArrayDeque and holds the capacity to contain a specified number of elements.

ArrayDeque<E>dq = new ArrayDeque<E>(int numofElements);

Example:

- Java

```
// Java program to demonstrate few functions of  
  
// ArrayDeque in Java  
  
import java.util.*;  
  
public class ArrayDequeDemo  
  
{  
  
    public static void main(String[] args)
```

```
{\n\n    // Initializing an deque\n\n    Deque<Integer>de_QUE = newArrayDeque<Integer>(10);\n\n\n\n    // add() method to insert\n\n    de_QUE.add(10);\n\n    de_QUE.add(20);\n\n    de_QUE.add(30);\n\n    de_QUE.add(40);\n\n    de_QUE.add(50);\n\n    for(Integer element :de_QUE)\n\n    {\n\n        System.out.println("Element : "+ element);\n\n    }\n\n\n\n    System.out.println("Using clear() ");\n\n\n\n    // clear() method\n\n    de_QUE.clear();\n\n\n\n    // addFirst() method to insert at start
```

```
de_QUE.addFirst(564);

de_QUE.addFirst(291);

// addLast() method to insert at end

de_QUE.addLast(24);

de_QUE.addLast(14);

System.out.println("Above elements are removed now");

// Iterator():

System.out.println("Elements of deque using Iterator :");

for(Iterator itr = de_QUE.iterator(); itr.hasNext();)

{

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

}

// descendingIterator() : to reverse the deque order

System.out.println("Elements of deque in reverse order :");

for(Iterator dItr = de_QUE.descendingIterator();

    dItr.hasNext();)
```

```
{  
    System.out.println(dItr.next());  
}  
  
// element() method : to get Head element  
  
System.out.println("\nHead Element using element(): "+  
    de_QUE.element());  
  
// getFirst() method : to get Head element  
  
System.out.println("Head Element using getFirst(): "+  
    de_QUE.getFirst());  
  
// getLast() method : to get last element  
  
System.out.println("Last Element using getLast(): "+  
    de_QUE.getLast());  
  
// toArray() method :  
  
Object[] arr = de_QUE.toArray();  
  
System.out.println("\nArraySize : "+ arr.length);  
  
System.out.print("Array elements : ");
```

```

for(int i=0; i<arr.length ; i++)

    System.out.print(" "+ arr[i]);



// peek() method : to get head

System.out.println("\nHead element : "+ de_QUE.peek());


// poll() method : to get head

System.out.println("Head element poll : "+ de_QUE.poll());


// push() method :

de_QUE.push(265);

de_QUE.push(984);

de_QUE.push(2365);


// remove() method : to get head

System.out.println("Head element remove : "+ de_QUE.remove());


System.out.println("The final array is: "+de_QUE);

}

}

```

Output:

```
Element : 10
```

```
Element : 20
```

```
Element : 30
```

```
Element : 40
```

```
Element : 50
```

```
Using clear()
```

```
Above elements are removed now
```

```
Elements of deque using Iterator :
```

```
291
```

```
564
```

```
24
```

```
14
```

```
Elements of deque in reverse order :
```

```
14
```

```
24
```

```
564
```

```
291
```

```
Head Element using element(): 291
```

```
Head Element using getFirst(): 291
```

```
Last Element using getLast(): 14
```

```
Array Size : 4
```

```
Array elements : 291 564 24 14
```

```
Head element : 291
```

```
Head element poll : 291
```

```
Head element remove : 2365
```

```
The final array is: [984, 265, 564, 24, 14]
```

Performing Various Operations on the ArrayDeque class

Let's see how to perform a few frequently used operations on the ArrayDeque.

1. Adding Elements: In order to add an element to the ArrayDeque, we can use the methods `add()`, `addFirst()`, `addLast()`, `offer()`, `offerFirst()`, `offerLast()` methods.

- [add\(\)](#)
- [addFirst\(\)](#)
- [addLast\(\)](#)
- [offer\(\)](#)
- [offerFirst\(\)](#)
- [offerLast\(\)](#)

- Java

```
// Java program to demonstrate the

// addition of elements in ArrayDeque

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

public class AddingElementsToArrayDeque {

    public static void main(String[] args)
    {
        // Initializing a deque
        // since deque is an interface
        // it is assigned the
        // ArrayDeque class

        Deque<String> dq = new ArrayDeque<String>();
    }
}
```

```

// add() method to insert

dq.add("The");

dq.addFirst("To");

dq.addLast("Geeks");



// offer() method to insert

dq.offer("For");

dq.offerFirst("Welcome");

dq.offerLast("Geeks");




// printing Elements of ArrayDeque to the console

System.out.println("ArrayDeque : "+ dq);

}

}

```

Output:

ArrayDeque : [Welcome, To, The, Geeks, For, Geeks]

2. Accessing the Elements: After adding the elements, if we wish to access the elements, we can use inbuilt methods like `getFirst()`, `getLast()`, etc.

- [getFirst\(\)](#)
- [getLast\(\)](#)
- [peek\(\)](#)
- [peekFirst\(\)](#)
- [peekLast\(\)](#)

- Java


```

        System.out.println("ArrayDeque: " + de_QUE);

        // Displaying the First element
        System.out.println("The first element is: "
                + de_QUE.getFirst());

        // Displaying the Last element
        System.out.println("The last element is: "
                + de_QUE.getLast());
    }
}

```

Output:

ArrayDeque: [Welcome, To, Geeks, 4, Geeks]

The first element is: Welcome

The last element is: Geeks

3. Removing Elements: In order to remove an element from a deque, there are various methods available. Since we can also remove from both the ends, the deque interface provides us with [removeFirst\(\)](#), [removeLast\(\)](#) methods. Apart from that, this interface also provides us with the [poll\(\)](#), [pop\(\)](#), [pollFirst\(\)](#), [pollLast\(\)](#) methods where pop() is used to remove and return the head of the deque. However, poll() is used because this offers the same functionality as pop() and doesn't return an exception when the deque is empty.

- [remove\(\)](#)
- [removeFirst\(\)](#)
- [removeLast\(\)](#)
- [poll\(\)](#)
- [pollFirst\(\)](#)
- [pollLast\(\)](#)
- [pop\(\)](#)

- Java

```
// Java program to demonstrate the  
  
// removal of elements in deque  
  
  
import java.util.*;  
  
  
  
  
public class RemoveElementsFromArrayDeque {  
  
  
  
  
    public static void main(String[] args)  
    {  
  
        // Initializing a deque  
  
        Deque<String> dq = newArrayDeque<String>();  
  
  
  
  
        // add() method to insert  
  
        dq.add("One");  
  
  
  
  
        // addFirst inserts at the front  
  
        dq.addFirst("Two");  
  
  
  
  
        // addLast inserts at the back  
  
        dq.addLast("Three");  
  
        // printing the elements of the deque  
        System.out.println(dq);  
    }  
}
```

```
        dq.addLast("Three");

        // print elements to the console
        System.out.println("ArrayDeque : "+ dq);

        // remove element as a stack from top/front
        System.out.println(dq.pop());

        // remove element as a queue from front
        System.out.println(dq.poll());

        // remove element from front
        System.out.println(dq.pollFirst());

        // remove element from back
        System.out.println(dq.pollLast());
    }
}
```

Output:

ArrayDeque : [Two, One, Three]

Two

One

Three

null

4. Iterating through the Deque: Since a deque can be iterated from both the directions, the iterator method of the deque interface provides us two ways to iterate. One from the first and the other from the back.

- [iterator\(\)](#)
- [descendingIterator\(\)](#)

- Java

```
// Java program to demonstrate the

// iteration of elements in deque

import java.util.*;

public class IterateArrayDeque {

    public static void main(String[] args)
    {
        // Initializing an deque
        Deque<String> dq = newArrayDeque<String>();

        // add() method to insert
        // at the back
        dq.add("For");
    }
}
```

```
// add element at the front

dq.addFirst("Geeks");

// add element at the back

dq.addLast("Geeks");

dq.add("is so good");

// Iterate using Iterator interface

// from the front of the queue

for(Iterator itr = dq.iterator(); itr.hasNext();) {

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

}

System.out.println();

// Iterate in reverse

// sequence in a queue

for(Iterator itr = dq.descendingIterator();

    itr.hasNext();) {

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

}
```

```
    }  
  
}  
  
}
```

Output:

Geeks For Geeks is so good
is so good Geeks For Geeks

Methods in ArrayDeque

Here, **Element** is the type of elements stored by ArrayDeque.

METHOD	DESCRIPTION
<u>add(Element e)</u>	The method inserts a particular element at the end of the deque.
<u>addAll(Collection<? extends E> c)</u>	Adds all of the elements in the specified collection at the end of this deque, as if by calling addLast(E) on each one, in the order that they are returned by the collection's iterator.
<u>addFirst(Element e)</u>	The method inserts particular element at the start of the deque.
<u>addLast(Element e)</u>	The method inserts a particular element at the end of the deque. It is similar to the add() method
<u>clear()</u>	The method removes all deque elements.
<u>clone()</u>	The method copies the deque.

METHOD	DESCRIPTION
<u>contains(Obj)</u>	The method checks whether a deque contains the element or not
<u>element()</u>	The method returns element at the head of the deque
<u>forEach(Consumer<? super E> action)</u>	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
<u>getFirst()</u>	The method returns first element of the deque
<u>getLast()</u>	The method returns last element of the deque
<u>isEmpty()</u>	The method checks whether the deque is empty or not.
<u>iterator()</u>	Returns an iterator over the elements in this deque.
<u>offer(Element e)</u>	The method inserts element at the end of deque.
<u>offerFirst(Element e)</u>	The method inserts element at the front of deque.
<u>offerLast(Element e)</u>	The method inserts element at the end of the deque.
<u>peek()</u>	The method returns head element without removing it.
<u>poll()</u>	The method returns head element and also removes it
<u>pop()</u>	The method pops out an element for stack represented by deque

METHOD	DESCRIPTION
<u>push(Element e)</u>	The method pushes an element onto stack represented by deque
<u>remove()</u>	The method returns head element and also removes it
<u>remove(Object o)</u>	Removes a single instance of the specified element from this deque.
<u>removeAll(Collection<?> c)</u>	Removes all of this collection's elements that are also contained in the specified collection (optional operation).
<u>removeFirst()</u>	The method returns the first element and also removes it
<u>removeFirstOccurrence (Object o)</u>	Removes the first occurrence of the specified element in this deque (when traversing the deque from head to tail).
<u>removeIf(Predicate<? super Element> filter)</u>	Removes all of the elements of this collection that satisfy the given predicate.
<u>removeLast()</u>	The method returns the last element and also removes it
<u>removeLastOccurrence (Object o)</u>	Removes the last occurrence of the specified element in this deque (when traversing the deque from head to tail).
<u>retainAll(Collection<?> c)</u>	Retains only the elements in this collection that are contained in the specified collection (optional operation).

METHOD	DESCRIPTION
<u>size()</u>	Returns the number of elements in this deque.
<u>spliterator()</u>	Creates a late-binding and fail-fast Spliterator over the elements in this deque.
<u>toArray()</u>	Returns an array containing all of the elements in this deque in proper sequence (from first to the last element).
<u>toArray(T[] a)</u>	Returns an array containing all of the elements in this deque in proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.

Methods declared in interface java.util.Deque

METHOD	DESCRIPTION
<u>descendingIterator()</u>	Returns an iterator over the elements in this deque in reverse sequential order.
<u>peekFirst()</u>	Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty.
<u>peekLast()</u>	Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty.
<u>pollFirst()</u>	Retrieves and removes the first element of this deque, or returns null if this deque is empty.

METHOD	DESCRIPTION
pollLast()	Retrieves and removes the last element of this deque, or returns null if this deque is empty.

Gautam
Karan
Ajay

Set Interface

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashCode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

Difference between List and Set

A list can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class

The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

HashSet class declaration

Let's see the declaration for java.util.HashSet class.

1. **public class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Clonable, Serializable

Constructors of Java HashSet class

SN	Constructor	Description
1)	HashSet()	It is used to construct a default HashSet.
2)	HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
3)	HashSet(int capacity, float loadFactor)	It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
4)	HashSet(Collection<? extends E> c)	It is used to initialize the hash set by using the elements of the collection c.

Methods of Java HashSet class

Various methods of Java HashSet class are as follows:

SN	Modifier & Type	Method	Description
1)	boolean	<u>add(E e)</u>	It is used to add the specified element to this set if it is not already present.
2)	void	<u>clear()</u>	It is used to remove all of the elements from the set.
3)	object	<u>clone()</u>	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	<u>contains(Object</u>	It is used to return true if this set contains

		<u>o)</u>	the specified element.
5)	boolean	<u>isEmpty()</u>	It is used to return true if this set contains no elements.
6)	Iterator<E>	<u>iterator()</u>	It is used to return an iterator over the elements in this set.
7)	boolean	<u>remove(Object o)</u>	It is used to remove the specified element from this set if it is present.
8)	int	<u>size()</u>	It is used to return the number of elements in the set.
9)	Spliterator<E>	<u>spliterator()</u>	It is used to create a late-binding and fail-fast Spliterator over the elements in the set.

Java HashSet Example

Let's see a simple example of HashSet. Notice, the elements iterate in an unordered collection.

```

1. import java.util.*;
2. class HashSet1{
3.     public static void main(String args[]){
4.         //Creating HashSet and adding elements
5.         HashSet<String> set=new HashSet();
6.         set.add("One");
7.         set.add("Two");
8.         set.add("Three");
9.         set.add("Four");
10.        set.add("Five");
11.        Iterator<String> i=set.iterator();
12.        while(i.hasNext())
13.        {
14.            System.out.println(i.next());
15.        }
16.    }

```

```
17.}
```

```
Five  
One  
Four  
Two  
Three
```

Java HashSet example ignoring duplicate elements

In this example, we see that HashSet doesn't allow duplicate elements.

```
1. import java.util.*;  
2. class HashSet2{  
3.     public static void main(String args[]){  
4.         //Creating HashSet and adding elements  
5.         HashSet<String> set=new HashSet<String>();  
6.         set.add("Ravi");  
7.         set.add("Vijay");  
8.         set.add("Ravi");  
9.         set.add("Ajay");  
10.        //Traversing elements  
11.        Iterator<String> itr=set.iterator();  
12.        while(itr.hasNext()){  
13.            System.out.println(itr.next());  
14.        }  
15.    }  
16.}
```

```
Ajay  
Vijay  
Ravi
```

Java HashSet example to remove elements

Here, we see different ways to remove an element.

```
1. import java.util.*;  
2. class HashSet3{  
3.     public static void main(String args[]){  
4.         HashSet<String> set=new HashSet<String>();  
5.         set.add("Ravi");  
6.         set.add("Vijay");
```

```

7.      set.add("Arun");
8.      set.add("Sumit");
9.      System.out.println("An initial list of elements: "+set);
10.     //Removing specific element from HashSet
11.     set.remove("Ravi");
12.     System.out.println("After invoking remove(object) method: "+set);
13.     HashSet<String> set1=new HashSet<String>();
14.     set1.add("Ajay");
15.     set1.add("Gaurav");
16.     set.addAll(set1);
17.     System.out.println("Updated List: "+set);
18.     //Removing all the new elements from HashSet
19.     set.removeAll(set1);
20.     System.out.println("After invoking removeAll() method: "+set);
21.     //Removing elements on the basis of specified condition
22.     set.removeIf(str->str.contains("Vijay"));
23.     System.out.println("After invoking removeIf() method: "+set);
24.     //Removing all the elements available in the set
25.     set.clear();
26.     System.out.println("After invoking clear() method: "+set);
27. }
28.

```

```

An initial list of elements: [Vijay, Ravi, Arun, Sumit]
After invoking remove(object) method: [Vijay, Arun, Sumit]
Updated List: [Vijay, Arun, Gaurav, Sumit, Ajay]
After invoking removeAll() method: [Vijay, Arun, Sumit]
After invoking removeIf() method: [Arun, Sumit]
After invoking clear() method: []

```

Java HashSet from another Collection

```

1. import java.util.*;
2. class HashSet4{
3.     public static void main(String args[]){
4.         ArrayList<String> list=new ArrayList<String>();
5.         list.add("Ravi");
6.         list.add("Vijay");
7.         list.add("Ajay");
8.

```

```
9.         HashSet<String> set=new HashSet(list);
10.        set.add("Gaurav");
11.        Iterator<String> i=set.iterator();
12.        while(i.hasNext())
13.        {
14.            System.out.println(i.next());
15.        }
16.    }
17.}
```

```
Vijay
Ravi
Gaurav
Ajay
```

Java HashSet Example: Book

Let's see a HashSet example where we are adding books to set and printing all the books.

```
1. import java.util.*;
2. class Book {
3.     int id;
4.     String name,author,publisher;
5.     int quantity;
6.     public Book(int id, String name, String author, String publisher, int quantity
7. ) {
8.     this.id = id;
9.     this.name = name;
10.    this.author = author;
11.    this.publisher = publisher;
12.    this.quantity = quantity;
13.}
14. public class HashSetExample {
15.     public static void main(String[] args) {
16.         HashSet<Book> set=new HashSet<Book>();
17.         //Creating Books
18.         Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
```

```
19. Book b2=new Book(102,"Data Communications & Networking","Forouzan"  
,"Mc Graw Hill",4);  
20. Book b3=new Book(103,"Operating System","Galvin","Wiley",6);  
21. //Adding Books to HashSet  
22. set.add(b1);  
23. set.add(b2);  
24. set.add(b3);  
25. System.out.println(set);  
26. //Traversing HashSet  
27. for(Book b:set){  
28. System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.  
quantity);  
29. }  
30. }  
31. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8  
102 Data Communications & Networking Forouzan Mc Graw Hill 4  
103 Operating System Galvin Wiley 6
```

Java HashSet

A HashSet is a collection of items where every item is unique, and it is found in the `java.util` package:

Example

Create a `HashSet` object called `cars` that will store strings:

```
import java.util.HashSet;// Import the HashSet class  
  
HashSet<String> cars =newHashSet<String>();
```

Add Items

The `HashSet` class has many useful methods. For example, to add items to it, use the `add()` method:

Example

```
// Import the HashSet class
import java.util.HashSet;

public class Main{
    public static void main(String[] args){
        HashSet<String> cars = new HashSet<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("BMW");
        cars.add("Mazda");
        System.out.println(cars);
    }
}
```

[Try it Yourself »](#)

Note: In the example above, even though BMW is added twice it only appears once in the set because every item in a set has to be unique.

Check If an Item Exists

To check whether an item exists in a `HashSet`, use the `contains()` method:

Example

```
cars.contains("Mazda");
```

[Try it Yourself »](#)

Remove an Item

To remove an item, use the `remove()` method:

Example

```
cars.remove("Volvo");
```

[Try it Yourself »](#)

To remove all items, use the `clear()` method:

Example

```
cars.clear();
```

[Try it Yourself »](#)

HashSet Size

To find out how many items there are, use the `size` method:

Example

```
cars.size();
```

[Try it Yourself »](#)

Loop Through a HashSet

Loop through the items of an `HashSet` with a **for-each** loop:

Example

```
for(String i: cars){  
    System.out.println(i);  
}
```

[Try it Yourself »](#)

Other Types

Items in an HashSet are actually objects. In the examples above, we created items (objects) of type "String". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent [wrapper class](#): `Integer`. For other primitive types, use: `Boolean` for boolean, `Character` for char, `Double` for double, etc:

Example

Use a `HashSet` that stores `Integer` objects:

```
import java.util.HashSet;  
  
public class Main{  
    public static void main(String[] args){  
  
        // Create a HashSet object called numbers  
        HashSet<Integer> numbers = new HashSet<Integer>();  
  
        // Add values to the set  
        numbers.add(4);  
        numbers.add(7);  
        numbers.add(8);  
    }  
}
```

```
// Show which numbers between 1 and 10 are in the set
for(int i=1;i<=10;i++){
    if(numbers.contains(i)){
        System.out.println(i+" was found in the set.");
    }else{
        System.out.println(i+" was not found in the set.");
    }
}
}
}
}
```

Constructors of LinkedHashSet class:

1. LinkedHashSet(): This constructor is used to create a default HashSet
LinkedHashSet<E>hs = new LinkedHashSet<E>();

2. LinkedHashSet(Collection C): Used in initializing the HashSet with the elements of the collection C.

LinkedHashSet<E>hs = new LinkedHashSet<E>(Collection c);

3. LinkedHashSet(int size): Used to initialize the size of the LinkedHashSet with the integer mentioned in the parameter.

LinkedHashSet<E>hs = new LinkedHashSet<E>(int size);

4. LinkedHashSet(int capacity, float fillRatio): Can be used to initialize both the capacity and the fill ratio, also called the load capacity of the LinkedHashSet with the arguments mentioned in the parameter. When the number of elements exceeds the capacity of the hash set is multiplied with the fill ratio thus expanding the capacity of the LinkedHashSet.

LinkedHashSet<E>hs = new LinkedHashSet<E>(int capacity, int fillRatio);

Example:

- Java

```
// Java Program to illustrate the LinkedHashSet

import java.util.LinkedHashSet;

public class LinkedHashSetExample

{

    // Main Method

    public static void main(String[] args)

    {

        LinkedHashSet<String> linkedset =

            new LinkedHashSet<String>();




        // Adding element to LinkedHashSet

        linkedset.add("A");

        linkedset.add("B");

        linkedset.add("C");

        linkedset.add("D");




        // This will not add new element as A already exists

        linkedset.add("A");

        linkedset.add("E");
    }
}
```

```

        System.out.println("Size of LinkedHashSet = "+
                            linkedset.size());

        System.out.println("Original LinkedHashSet:"+ linkedset);

        System.out.println("Removing D from LinkedHashSet: "+
                            linkedset.remove("D"));

        System.out.println("Trying to Remove Z which is not "+
                            "present: "+ linkedset.remove("Z"));

        System.out.println("Checking if A is present="+
                            linkedset.contains("A"));

        System.out.println("Updated LinkedHashSet: "+ linkedset);

    }

}

```

Performing various operations on the LinkedHashSet class

Let's see how to perform a few frequently used operations on the LinkedHashSet.

1. Adding Elements: In order to add an element to the LinkedHashSet, we can use the [add\(\)](#) method. This is different from HashSet because in HashSet, the insertion order is not retained but it is retained in the LinkedHashSet.

- Java

```

// Java program for adding
// elements to LinkedHashSet

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

class AddingElementsToLinkedHashSet {

```

```

public static void main(String[] args)
{
    // create an instance of
    // LinkedHashSet
    LinkedHashSet<String>hs
        = new LinkedHashSet<String>();

    // Elements are added using add() method
    // insertion order is maintained
    hs.add("Geek");
    hs.add("For");
    hs.add("Geeks");

    // print elements to the console
    System.out.println("LinkedHashSet : " + hs);
}
}

```

Output:

LinkedHashSet : [Geek, For, Geeks]

2. Removing the Elements: The values can be removed from the LinkedHashSet using the [remove\(\)](#) method.

- Java

```

// Java program to remove elements
// from LinkedHashSet
import java.io.*;
import java.util.*;

class RemoveElementsFromLinkedHashSet {

```

```
public static void main(String[] args)
{
    // create an instance of
    // LinkedHashSet
    LinkedHashSet<String>hs
        = new LinkedHashSet<String>();

    // Elements are added using add() method
    hs.add("Geek");
    hs.add("For");
    hs.add("Geeks");
    hs.add("A");
    hs.add("B");
    hs.add("Z");

    // print elements to the console
    System.out.println("Initial HashSet " + hs);

    // Removing the element b
    hs.remove("B");

    System.out.println("After removing element " + hs);

    // Returns false if the element is not present
    System.out.println(hs.remove("AC"));
}

}
```

Output:

Initial HashSet [Geek, For, Geeks, A, B, Z]

After removing element [Geek, For, Geeks, A, Z]

false

3. Iterating through the LinkedHashSet: Iterate through the elements of LinkedHashSet using the [iterator\(\)](#) method. The most famous one is to use the [enhanced for loop](#).

- Java

```
// Java code to demonstrate  
// the iterating over LinkedHashSet  
  
import java.io.*;  
import java.util.*;  
  
class IteratingLinkedHashSet {  
  
    public static void main(String[] args)  
    {  
        // Instantiate an object of Set  
        // Since LinkedHashSet implements Set  
        // Set points to LinkedHashSet  
        Set<String>hs = new LinkedHashSet<String>();  
  
        // Elements are added using add() method  
        hs.add("Geek");  
        hs.add("For");  
        hs.add("Geeks");  
        hs.add("A");  
        hs.add("B");  
        hs.add("Z");
```

```

// Iterating though the LinkedHashSet
Iterator itr = hs.iterator();
while (itr.hasNext())
    System.out.print(itr.next() + ", ");
System.out.println();
//Object
// Using enhanced for loop
for (String s : hs)
    System.out.print(s + ", ");
System.out.println();
}
}

```

Output:

Geek, For, Geeks, A, B, Z,

Geek, For, Geeks, A, B, Z,

Methods of LinkedHashSet

Here, **E** is the type of element stored.

METHOD	DESCRIPTION
--------	-------------

[spliterator\(\)](#) Creates a late-binding and fail-fast Spliterator over the elements in this set.

Methods declared in class java.util.AbstractSet

METHOD	DESCRIPTION
--------	-------------

[equals\(Object o\)](#) Compares the specified object with this set for equality.

[hashCode\(\)](#) Returns the hash code value for this set.

METHOD	DESCRIPTION
removeAll(Collection c)	Removes from this set all of its elements that are contained in the specified collection (optional operation).
Methods declared in class java.util.AbstractCollection	
METHOD	DESCRIPTION
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this collection (optional operation).
containsAll(Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
retainAll(Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
toArray()	Returns an array containing all of the elements in this collection.
toArray(T[] a)	Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.
toString()	Returns a string representation of this collection.
Methods declared in interface java.util.Collection	
METHOD	DESCRIPTION
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
removeIf(Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.

METHOD	DESCRIPTION
stream()	Returns a sequential Stream with this collection as its source.

Methods declared in class java.util.HashSet

METHOD	DESCRIPTION
add(E e)	Adds the specified element to this set if it is not already present.
clear()	Removes all of the elements from this set.
clone()	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
contains(Object o)	Returns true if this set contains the specified element.
isEmpty()	Returns true if this set contains no elements.
iterator()	Returns an iterator over the elements in this set.
remove(Object o)	Removes the specified element from this set if it is present.
size()	Returns the number of elements in this set (its cardinality).

Methods declared in interface java.lang.Iterable

METHOD	DESCRIPTION
forEach(Consumer<? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Methods declared in interface java.util.Set

METHOD	DESCRIPTION
<u>add(element)</u>	This method is used to add a specific element to the set. The function adds the element only if the specified element is not already present in the set else the function returns False if the element is already present in the Set.
<u>addAll(Collection c)</u>	This method is used to append all of the elements from the mentioned collection to the existing set. The elements are added randomly without following any specific order.
<u>clear()</u>	This method is used to remove all the elements from the set but not delete the set. The reference for the set still exists.
<u>contains(element)</u>	This method is used to check whether a specific element is present in the Set or not.
<u>containsAll(Collection c)</u>	This method is used to check whether the set contains all the elements present in the given collection or not. This method returns true if the set contains all the elements and returns false if any of the elements are missing.
<u>hashCode()</u>	This method is used to get the hashCode value for this instance of the Set. It returns an integer value which is the hashCode value for this instance of the Set.
<u>isEmpty()</u>	This method is used to check whether the set is empty or not.
<u>iterator()</u>	This method is used to return the iterator of the set. The elements from the set are returned in random order.
<u>remove(element)</u>	This method is used to remove the given element from the set. This method returns True if the specified element is present in the Set otherwise it returns False.
<u>removeAll(collection)</u>	This method is used to remove all the elements from the collection which are present in the set. This method returns true if this set changed as a

METHOD	DESCRIPTION
	result of the call.
<u>retainAll(collection)</u>	This method is used to retain all the elements from the set which are mentioned in the given collection. This method returns true if this set changed as a result of the call.
<u>size()</u>	This method is used to get the size of the set. This returns an integer value which signifies the number of elements.
<u>toArray()</u>	This method is used to form an array of the same elements as that of the Set.
<u>toArray(T[] a)</u>	Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

Following is the difference between [LinkedHashMap](#) and LinekdHashSet:

```

importjava.util.Vector;

importjava.util.Enumeration;

publicclassEnumerationClass {

    publicstaticvoidmain(String args[])
    {

```

```
Enumeration months;

Vector<String>monthNames = newVector<>();

monthNames.add("January");

monthNames.add("Febraury");

monthNames.add("March");

monthNames.add("April");

monthNames.add("May");

monthNames.add("June");

monthNames.add("July");

monthNames.add("August");

monthNames.add("September");

monthNames.add("Octobor");

monthNames.add("November");

monthNames.add("December");

months = monthNames.elements();

while(months.hasMoreElements()) {

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

}

}
```

Output

January

Febrary

March

April

May

June

July

August

September

Octobor

November

December

Set Interface in Java is present in `java.util` package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by `HashSet`, `LinkedHashSet`, and `TreeSet`.

Set can be instantiated as:

1. `Set<data-type> s1 = new HashSet<data-type>();`
2. `Set<data-type> s2 = new LinkedHashSet<data-type>();`
3. `Set<data-type> s3 = new TreeSet<data-type>();`

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 unique items.

Consider the following example.

1. `import java.util.*;`
2. `public class TestJavaCollection7{`
3. `public static void main(String args[]){`

```
4. //Creating HashSet and adding elements
5. HashSet<String> set=new HashSet<String>();
6. set.add("Ravi");
7. set.add("Vijay");
8. set.add("Ravi");
9. set.add("Ajay");
10.//Traversing elements
11.Iterator<String> itr=set.iterator();
12.while(itr.hasNext()){
13.System.out.println(itr.next());
14.}
15.}
16.}
```

Output:

```
Vijay
Ravi
Ajay
```

LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

Consider the following example.

```
1. import java.util.*;
2. public class TestJavaCollection8{
3. public static void main(String args[]){
4. LinkedHashSet<String> set=new LinkedHashSet<String>();
5. set.add("Ravi");
6. set.add("Vijay");
7. set.add("Ravi");
8. set.add("Ajay");
9. Iterator<String> itr=set.iterator();
10.while(itr.hasNext()){
11.System.out.println(itr.next());
12.}}
```

```
13.}  
14.}
```

Output:

```
Ravi  
Vijay  
Ajay
```

SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

```
1. SortedSet<data-type> set = new TreeSet();
```

TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

Hierarchy of TreeSet class

As shown in the above diagram, Java TreeSet class implements the NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

1. **public class** TreeSet<E> **extends** AbstractSet<E> **implements** NavigableSet<E>, Comparable, Serializable

Constructors of Java TreeSet class

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set.
TreeSet(Collection<? extends E> c)	It is used to build a new tree set that contains the elements of the collection c.
TreeSet(Comparator<? super E> comparator)	It is used to construct an empty tree set that will be sorted according to given comparator.
TreeSet(SortedSet<E> s)	It is used to build a TreeSet that contains the elements of the given SortedSet.

Methods of Java TreeSet class

Method	Description
boolean add(E e)	It is used to add the specified element to this set if it is not already present.
boolean addAll(Collection<? extends E> c)	It is used to add all of the elements in the specified collection to this set.
E ceiling(E e)	It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.
Comparator<? super E> comparator()	It returns comparator that arranged elements in order.

Iterator descendingIterator()	It is used iterate the elements in descending order.
NavigableSet descendingSet()	It returns the elements in reverse order.
E floor(E e)	It returns the equal or closest least element of the specified element from the set, or null there is no such element.
SortedSet headSet(E toElement)	It returns the group of elements that are less than the specified element.
NavigableSet headSet(E toElement, boolean inclusive)	It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.
Iterator iterator()	It is used to iterate the elements in ascending order.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.
Spliterator spliterator()	It is used to create a late-binding and fail-fast spliterator over the elements.
NavigableSet subSet(E fromElement, E toElement, boolean fromInclusive, boolean toInclusive)	It returns a set of elements that lie between the given range.
SortedSet subSet(E fromElement, E toElement, boolean fromInclusive, boolean toInclusive)	It returns a set of elements that lie between the given range.

toElement))	between the given range which includes fromElement and excludes toElement.
SortedSet tailSet(E fromElement)	It returns a set of elements that are greater than or equal to the specified element.
NavigableSet tailSet(E fromElement, boolean inclusive)	It returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element.
boolean contains(Object o)	It returns true if this set contains the specified element.
boolean isEmpty()	It returns true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It returns a shallow copy of this TreeSet instance.
E first()	It returns the first (lowest) element currently in this sorted set.
E last()	It returns the last (highest) element currently in this sorted set.
int size()	It returns the number of elements in this set.

Java TreeSet Examples

Java TreeSet Example 1:

Let's see a simple example of Java TreeSet.

1. **import** java.util.*;
2. **class** TreeSet1{

```
3. public static void main(String args[]){
4.     //Creating and adding elements
5.     TreeSet<String> al=new TreeSet<String>();
6.     al.add("Ravi");
7.     al.add("Vijay");
8.     al.add("Ravi");
9.     al.add("Ajay");
10.    //Traversing elements
11.    Iterator<String> itr=al.iterator();
12.    while(itr.hasNext()){
13.        System.out.println(itr.next());
14.    }
15. }
16. }
```

Test it Now

Output:

```
Ajay
Ravi
Vijay
```

Java TreeSet Example 2:

Let's see an example of traversing elements in descending order.

```
1. import java.util.*;
2. class TreeSet2{
3.     public static void main(String args[]){
4.         TreeSet<String> set=new TreeSet<String>();
5.         set.add("Ravi");
6.         set.add("Vijay");
7.         set.add("Ajay");
8.         System.out.println("Traversing element through Iterator in descending order");
9.         Iterator i=set.descendingIterator();
10.        while(i.hasNext())
11.        {
```

```
12.         System.out.println(i.next());
13.     }
14.
15. }
16.}
```

Test it Now

Output:

```
Traversing element through Iterator in descending order
Vijay
Ravi
Ajay
Traversing element through NavigableSet in descending order
Vijay
Ravi
Ajay
```

Java TreeSet Example 3:

Let's see an example to retrieve and remove the highest and lowest Value.

```
1. import java.util.*;
2. class TreeSet3{
3.     public static void main(String args[]){
4.         TreeSet<Integer> set=new TreeSet<Integer>();
5.         set.add(24);
6.         set.add(66);
7.         set.add(12);
8.         set.add(15);
9.         System.out.println("Highest Value: "+set.pollFirst());
10.        System.out.println("Lowest Value: "+set.pollLast());
11.    }
12.}
```

Output:

```
Highest Value: 12
Lowest Value: 66
```

Java TreeSet Example 4:

In this example, we perform various NavigableSet operations.

```
1. import java.util.*;
2. class TreeSet4{
3.     public static void main(String args[]){
4.         TreeSet<String> set=new TreeSet<String>();
5.         set.add("A");
6.         set.add("B");
7.         set.add("C");
8.         set.add("D");
9.         set.add("E");
10.        System.out.println("Initial Set: "+set);
11.
12.        System.out.println("Reverse Set: "+set.descendingSet());
13.
14.        System.out.println("Head Set: "+set.headSet("C", true));
15.
16.        System.out.println("SubSet: "+set.subSet("A", false, "E", true));
17.
18.        System.out.println("TailSet: "+set.tailSet("C", false));
19.    }
20. }
```

Output:

```
Initial Set: [A, B, C, D, E]
Reverse Set: [E, D, C, B, A]
Head Set: [A, B, C]
SubSet: [B, C, D, E]
TailSet: [D, E]
```

Java TreeSet Example 4:

In this example, we perform various SortedSetSet operations.

```
1. import java.util.*;
2. class TreeSet4{
3.     public static void main(String args[]){
4.         TreeSet<String> set=new TreeSet<String>();
5.         set.add("A");
```

```

6.     set.add("B");
7.     set.add("C");
8.     set.add("D");
9.     set.add("E");
10.
11.    System.out.println("Initial Set: " +set);
12.
13.    System.out.println("Head Set: " +set.headSet("C"));
14.
15.    System.out.println("SubSet: " +set.subSet("A", "E"));
16.
17.    System.out.println("TailSet: " +set.tailSet("C"));
18. }
19. }
```

Output:

```

Initial Set: [A, B, C, D, E]
Head Set: [A, B]
SubSet: [A, B, C, D]
TailSet: [C, D, E]
```

Java TreeSet Example: Book

Let's see a TreeSet example where we are adding books to set and printing all the books. The elements in TreeSet must be of a Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in TreeSet, you need to implement the Comparable interface.

```

1. import java.util.*;
2. class Book implements Comparable<Book>{
3.     int id;
4.     String name,author,publisher;
5.     int quantity;
6.     public Book(int id, String name, String author, String publisher, int quantity) {
```



```

7.         this.id = id;
8.         this.name = name;
9.         this.author = author;
10.        this.publisher = publisher;
11.        this.quantity = quantity;
```

```

12. }
13. public int compareTo(Book b) {
14.     if(id>b.id){
15.         return 1;
16.     }else if(id<b.id){
17.         return -1;
18.     }else{
19.         return 0;
20.     }
21. }
22. }

23. public class TreeSetExample {
24.     public static void main(String[] args) {
25.         Set<Book> set=new TreeSet<Book>();
26.         //Creating Books
27.         Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);
28.         Book b2=new Book(233,"Operating System","Galvin","Wiley",6);
29.         Book b3=new Book(101,"Data Communications & Networking","Forouzan",
   "Mc Graw Hill",4);
30.         //Adding Books to TreeSet
31.         set.add(b1);
32.         set.add(b2);
33.         set.add(b3);
34.         //Traversing TreeSet
35.         for(Book b:set){
36.             System.out.println(b.id+ " "+b.name+ " "+b.author+ " "+b.publisher+ " "+b.qu
   antity);
37.         }
38.     }
39. }

```

Output:

```

101 Data Communications & Networking Forouzan Mc Graw Hill 4
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6

```

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains 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:

```
1. import java.util.*;
2. public class TestJavaCollection9{
3.     public static void main(String args[]){
4.         //Creating and adding elements
5.         TreeSet<String> set=new TreeSet<String>();
6.         set.add("Ravi");
7.         set.add("Vijay");
8.         set.add("Ravi");
9.         set.add("Ajay");
10.        //traversing elements
11.        Iterator<String> itr=set.iterator();
12.        while(itr.hasNext()){
13.            System.out.println(itr.next());
14.        }
15.    }
16. }
```

Output:

```
Ajay
Ravi

Vijay
```

The class Date represents a specific instant in time, with millisecond precision. The Date class of java.util package implements Serializable, Cloneable and Comparable interface. It provides constructors and methods to deal with date and time with java.

Constructors

- **Date()** : Creates date object representing current date and time.
- **Date(long milliseconds)** : Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.
- **Date(int year, int month, int date)**
- **Date(int year, int month, int date, int hrs, int min)**
- **Date(int year, int month, int date, int hrs, int min, int sec)**
- **Date(String s)**

Note : The last 4 constructors of the Date class are Deprecated.

```
// Java program to demonstrate constructors of Date

import java.util.*;

public class Main

{

    public static void main(String[] args)

    {

        Date d1 = new Date();

        System.out.println("Current date is "+ d1);

        Date d2 = new Date(2323223232L);

        System.out.println("Date represented is "+ d2 );

    }

}
```

Output:

```
Current date is Tue Jul 12 18:35:37 IST 2016
Date represented is Wed Jan 28 02:50:23 IST 1970
```

Important Methods

- **boolean after(Date date)** : Tests if current date is after the given date.
- **D1//current date**
- **D2// previous date**
- **D2.after(d1);//false**
- **boolean before(Date date)** : Tests if current date is before the given date.
- **int compareTo(Date date)** : Compares current date with given date. Returns 0 if the argument Date is equal to the Date; a value less than 0 if the Date is before the Date argument; and a value greater than 0 if the Date is after the Date argument.
- **D1 Date d1=new Date();**
- **Date d2=new Date();**
- **D1.compareTo(d2)**
- **D2**
- **long getTime()** : Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.
- **void setTime(long time)** : Changes the current date and time to given time.

```
// Program to demonstrate methods of Date class

import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        // Creating date

        Date d1 = new Date(2000, 11, 21);

        Date d2 = new Date(); // Current date

        Date d3 = new Date(2010, 1, 3);
    }
}
```

```

booleana = d3.after(d1); // true return
System.out.println("Date d3 comes after "+
"date d2: "+ a);

booleanb = d3.before(d2);
System.out.println("Date d3 comes before "+
"date d2: "+ b);

intc = d1.compareTo(d2);
System.out.println(c);

System.out.println("Miliseconds from Jan 1 "+
"1970 to date d1 is "+ d1.getTime()));

System.out.println("Before setting "+d2);
d2.setTime(204587433443L);
System.out.println("After setting "+d2);

}
}

```

Output:

```

Date d3 comes after date d2: true
Date d3 comes before date d2: false

```

```
1
```

```
Miliseconds from Jan 1 1970 to date d1 is 60935500800000
```

```
Before setting Tue Jul 12 13:13:16 UTC 2016
```

```
After setting Fri Jun 25 21:50:33 UTC 1976
```

Calendar Class in Java with examples

- Difficulty Level : [Medium](#)
- Last Updated : 28 Aug, 2018

Calendar class in Java 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. It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.

As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

- Calendar.getInstance(): return a Calendar instance based on the current time in the default time zone with the default locale.
- Calendar.getInstance(TimeZone zone)
- Calendar.getInstance(Locale aLocale)
- Calendar.getInstance(TimeZone zone, Locale aLocale)

Java program to demonstrate getInstance() method:

```
// Date getTime(): It is used to return a

// Date object representing this

// Calendar's time value.

import java.util.*;

public class Calendar1 {

    public static void main(String args[])
    {

```

```

        Calendar c = Calendar.getInstance();

        System.out.println("The Current Date is:"+ c.getTime());

    }

}

```

Output:

The Current Date is:Tue Aug 28 11:10:40 UTC 2018

Important Methods and their usage

METHOD	DESCRIPTION
abstract void add(int field, int amount)	It is used to add or subtract the specified amount of time to the given calendar field, based on the calendar's rules.
int get(int field)	It is used to return the value of the given calendar field.
abstract int getMaximum(int field)	It is used to return the maximum value for the given calendar field of this Calendar instance.
abstract int getMinimum(int field)	It is used to return the minimum value for the given calendar field of this Calendar instance.
Date getTime()	It is used to return a Date object representing this Calendar's time value.</td>

Below programs illustrate the above methods:

Program 1: Java program to demonstrate get() method.

```

// Program to demonstrate get() method

// of Calendar class

import java.util.*;

```

```

public class Calendar2 {
    public static void main(String[] args) {
        // creating Calendar object
        Calendar calendar = Calendar.getInstance();

        // Demonstrate Calendar's get() method
        System.out.println("Current Calendar's Year: " +
                           calendar.get(Calendar.YEAR));
        System.out.println("Current Calendar's Day: " +
                           calendar.get(Calendar.DATE));
        System.out.println("Current MINUTE: " +
                           calendar.get(Calendar.MINUTE));
        System.out.println("Current SECOND: " +
                           calendar.get(Calendar.SECOND));
    }
}

```

Output:

Current Calendar's Year: 2018

Current Calendar's Day: 28

Current MINUTE: 10

Current SECOND: 45

Program 2: Java program to demonstrate getMaximum() method.

```

// Program to demonstrate getMaximum() method
// of Calendar class

```

```

import java.util.*;

public class Calendar3 {
    public static void main(String[] args) {
        // creating calendar object
        Calendar calendar = Calendar.getInstance();

        int max = calendar.getMaximum(Calendar.DAY_OF_WEEK);
        System.out.println("Maximum number of days in a week: " + max);

        max = calendar.getMaximum(Calendar.WEEK_OF_YEAR);
        System.out.println("Maximum number of weeks in a year: " + max);
    }
}

```

Output:

Maximum number of days in a week: 7

Maximum number of weeks in a year: 53

Program 3: Java program to demonstrate the getMinimum() method.

```

// Program to demonstrate getMinimum() method
// of Calendar class

```

```

import java.util.*;

public class Calendar4 {
    public static void main(String[] args) {
        // creating calendar object
        Calendar calendar = Calendar.getInstance();

        int min = calendar.getMinimum(Calendar.DAY_OF_WEEK);
        System.out.println("Minimum number of days in week: " + min);

        min = calendar.getMinimum(Calendar.WEEK_OF_YEAR);
        System.out.println("Minimum number of weeks in year: " + min);
    }
}

```

Output:

Minimum number of days in week: 1

Minimum number of weeks in year: 1

Program 4: Java program to demonstrate add() method.

```

// Program to demonstrate add() method
// of Calendar class

import java.util.*;

```

```

public class Calendar5 {
    public static void main(String[] args) {
        // creating calendar object
        Calendar calendar = Calendar.getInstance();
        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());
    }
}

```

Output:

```

15 days ago: Mon Aug 13 11:10:57 UTC 2018
4 months later: Thu Dec 13 11:10:57 UTC 2018
2 years later: Sun Dec 13 11:10:57 UTC 2020

```

A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.

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

Constructors:

StringTokenizer(String str) :
str is string to be tokenized.

Considers default delimiters like new line, space, tab, carriage return and form feed.

StringTokenizer(String str, String delim) :
delim is set of delimiters that are used to tokenize
the given string.

StringTokenizer(String str, String delim, boolean flag):
The first two parameters have same meaning. The flag
serves following purpose.

If the **flag** is **false**, delimiter characters serve to
separate tokens. For example, if string is "hello geeks"
and delimiter is " ", then tokens are "hello" and "geeks".

If the **flag** is **true**, delimiter characters are
considered to be tokens. For example, if string is "hello
geeks" and delimiter is " ", then tokens are "hello", " "
and "geeks".

```
/* A Java program to illustrate working of StringTokenizer

class:*/
import java.util.*;

public class NewClass

{
    public static void main(String args[])
    {
        System.out.println("Using Constructor 1 - ");

        StringTokenizer st1 =
            new StringTokenizer("Hello Geeks How are you", " ");
        while(st1.hasMoreTokens())
            System.out.println(st1.nextToken());
    }
}
```

```

        System.out.println("Using Constructor 2 - ");

        StringTokenizer st2 =
            new StringTokenizer("JAVA : Code : String", " :");

        while(st2.hasMoreTokens())
            System.out.println(st2.nextToken());
    }

    System.out.println("Using Constructor 3 - ");

    StringTokenizer st3 =
        new StringTokenizer("JAVA : Code : String", " :", true);

    while(st3.hasMoreTokens())
        System.out.println(st3.nextToken());
    }
}

```

Output :

Using Constructor 1 -

Hello

Geeks

How

are

you

Using Constructor 2 -

JAVA

Code

String

Using Constructor 3 -

JAVA

Class constructors

Sr.No.	Constructor & Description
1	 StringTokenizer(String str) This constructor creates a string tokenizer for the specified string.
2	 StringTokenizer(String str, String delim) This constructor constructs a string tokenizer for the specified string.
3	 StringTokenizer(String str, String delim, boolean returnDelims) This constructor constructs a string tokenizer for the specified string.

Class methods

Sr.No.	Method & Description
1	<u>int countTokens()</u> This method calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
2	<u>boolean hasMoreElements()</u> This method returns the same value as the hasMoreTokens method.
3	<u>boolean hasMoreTokens()</u> This method tests if there are more tokens available from this tokenizer's string.
4	<u>Object nextElement()</u> This method returns the same value as the nextToken method, except that its declared return value is Object rather than String.
5	<u>String nextToken()</u>

	This method returns the next token from this string tokenizer.
6	<u>String nextToken(String delim)</u> This method returns the next token in this string tokenizer's string.

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. This is a legacy class but it has been completely re-engineered in Java 2, version 1.4.

The BitSet defines the following two constructors.

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.

BitSet implements the Cloneable interface and defines the methods listed in the following table –

Sr.No.	Method & Description
1	void and(BitSet bitSet) ANDs the contents of the invoking BitSet object with those specified by bitSet. The result is placed into the invoking object.
2	void andNot(BitSet bitSet) For each 1 bit in bitSet, the corresponding bit in the invoking BitSet is cleared.
3	int cardinality() Returns the number of set bits in the invoking object.

4	void clear() Zeros all bits.
5	void clear(int index) Zeros the bit specified by index.
6	void clear(int startIndex, int endIndex) Zeros the bits from startIndex to endIndex.
7	Object clone() Duplicates the invoking BitSet object.
8	booleanequals(Object bitSet) Returns true if the invoking bit set is equivalent to the one passed in bitSet. Otherwise, the method returns false.
9	void flip(int index) Reverses the bit specified by the index.
10	void flip(int startIndex, int endIndex) Reverses the bits from startIndex to endIndex.
11	booleanganet(int index) Returns the current state of the bit at the specified index.
12	BitSetget(int startIndex, int endIndex) Returns a BitSet that consists of the bits from startIndex to endIndex. The invoking object is not changed.
13	int hashCode() Returns the hash code for the invoking object.
14	booleanintersects(BitSetbitSet) Returns true if at least one pair of corresponding bits within the invoking object

	and bitSet are 1.
15	boolean isEmpty() Returns true if all bits in the invoking object are zero.
16	int length() Returns the number of bits required to hold the contents of the invoking BitSet. This value is determined by the location of the last 1 bit.
17	int nextClearBit(int startIndex) Returns the index of the next cleared bit, (that is, the next zero bit), starting from the index specified by startIndex.
18	int nextSetBit(int startIndex) Returns the index of the next set bit (that is, the next 1 bit), starting from the index specified by startIndex. If no bit is set, -1 is returned.
19	void or(BitSet bitSet) ORs the contents of the invoking BitSet object with that specified by bitSet. The result is placed into the invoking object.
20	void set(int index) Sets the bit specified by index.
21	void set(int index, boolean v) Sets the bit specified by index to the value passed in v. True sets the bit, false clears the bit.
22	void set(int startIndex, int endIndex) Sets the bits from startIndex to endIndex.
23	void set(int startIndex, int endIndex, boolean v) Sets the bits from startIndex to endIndex, to the value passed in v. true sets the bits, false clears the bits.

24	int size() Returns the number of bits in the invoking BitSet object.
25	String toString() Returns the string equivalent of the invoking BitSet object.
26	void xor(BitSet bitSet) XORs the contents of the invoking BitSet object with that specified by bitSet. The result is placed into the invoking object.

Example

The following program illustrates several of the methods supported by this data structure –

[Live Demo](#)

```
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);
}
}

```

This will produce the following result –

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:
{ }

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. This is a legacy class but it has been completely re-engineered in Java 2, version 1.4.

The BitSet defines the following two constructors.

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.

BitSet implements the Cloneable interface and defines the methods listed in the following table –

Sr.No.	Method & Description
--------	----------------------

1	void and(BitSet bitSet) ANDs the contents of the invoking BitSet object with those specified by bitSet. The result is placed into the invoking object.
2	void andNot(BitSet bitSet) For each 1 bit in bitSet, the corresponding bit in the invoking BitSet is cleared.
3	int cardinality() Returns the number of set bits in the invoking object.
4	void clear() Zeros all bits.
5	void clear(int index) Zeros the bit specified by index.
6	void clear(int startIndex, int endIndex) Zeros the bits from startIndex to endIndex.
7	Object clone() Duplicates the invoking BitSet object.
8	boolean equals(Object bitSet) Returns true if the invoking bit set is equivalent to the one passed in bitSet. Otherwise, the method returns false.
9	void flip(int index) Reverses the bit specified by the index.
10	void flip(int startIndex, int endIndex) Reverses the bits from startIndex to endIndex.
11	boolean get(int index) Returns the current state of the bit at the specified index.

12	BitSetget(int startIndex, int endIndex) Returns a BitSet that consists of the bits from startIndex to endIndex. The invoking object is not changed.
13	int hashCode() Returns the hash code for the invoking object.
14	booleanintersects(BitSetbitSet) Returns true if at least one pair of corresponding bits within the invoking object and bitSet are 1.
15	booleanisEmpty() Returns true if all bits in the invoking object are zero.
16	int length() Returns the number of bits required to hold the contents of the invoking BitSet. This value is determined by the location of the last 1 bit.
17	int nextClearBit(int startIndex) Returns the index of the next cleared bit, (that is, the next zero bit), starting from the index specified by startIndex.
18	int nextSetBit(int startIndex) Returns the index of the next set bit (that is, the next 1 bit), starting from the index specified by startIndex. If no bit is set, -1 is returned.
19	void or(BitSetbitSet) ORs the contents of the invoking BitSet object with that specified by bitSet. The result is placed into the invoking object.
20	void set(int index) Sets the bit specified by index.
21	void set(int index, boolean v) Sets the bit specified by index to the value passed in v. True sets the bit, false

	clears the bit.
22	void set(int startIndex, int endIndex) Sets the bits from startIndex to endIndex.
23	void set(int startIndex, int endIndex, boolean v) Sets the bits from startIndex to endIndex, to the value passed in v. true sets the bits, false clears the bits.
24	int size() Returns the number of bits in the invoking BitSet object.
25	String toString() Returns the string equivalent of the invoking BitSet object.
26	void xor(BitSet bitSet) XORs the contents of the invoking BitSet object with that specified by bitSet. The result is placed into the invoking object.

Example

The following program illustrates several of the methods supported by this data structure –

[Live Demo](#)

```
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);
}
```

This will produce the following result –

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:
{ }

UNIT V

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

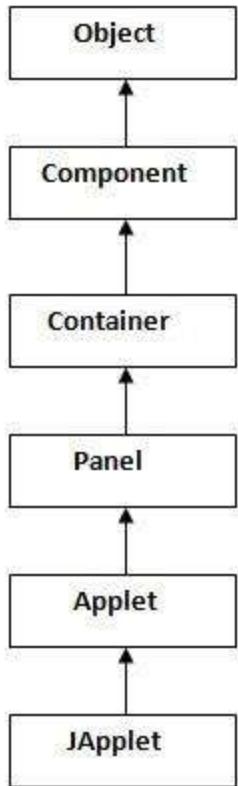
Drawback of Applet

- Plugin is required at client browser to execute applet.

Do You Know

- Who is responsible to manage the life cycle of an applet ?
- How to perform animation in applet ?
- How to paint like paint brush in applet ?
- How to display digital clock in applet ?
- How to display analog clock in applet ?
- How to communicate two applets ?

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subcl

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Applet Lifecycle



Lifecycle methods for Applet:

The `java.applet.Applet` class provides 4 life cycle methods and `java.awt.Component` class provides 1 life cycle method for an applet.

`java.applet.Applet` class

For creating any applet `java.applet.Applet` class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialize the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

`java.awt.Component` class

The Component class provides 1 life cycle method of applet.

-
1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

1. By html file.
 2. By appletViewer tool (for testing purpose).
-

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

1. //First.java
2. **import** java.applet.Applet;
3. **import** java.awt.Graphics;
4. **public class** First **extends** Applet{
- 5.
6. **public void** paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
- 9.
10. }

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

1. <html>
 2. <body>
 3. <applet code="**First.class**" width="300" height="300">
 4. </applet>
 5. </body>
 6. </html>
-

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

1. **//First.java**
2. **import** java.applet.Applet;
3. **import** java.awt.Graphics;
4. **public class** First **extends** Applet{
- 5.
6. **public void** paint(Graphics g){
7. g.drawString("welcome to applet",150,150);
8. }
- 9.
10. }
11. **/***
12. <applet code="First.class" width="300" height="300">
13. </applet>
14. ***/**

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java  
c:\>appletviewer First.java
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

1. **import** java.applet.Applet;
2. **import** java.awt.*;

```
3.  
4. public class GraphicsDemo extends Applet{  
5.  
6. public void paint(Graphics g){  
7.     g.setColor(Color.red);  
8.     g.drawString("Welcome",50, 50);  
9.     g.drawLine(20,30,20,300);  
10.    g.drawRect(70,100,30,30);  
11.    g.fillRect(170,100,30,30);  
12.    g.drawOval(70,200,30,30);  
13.  
14.    g.setColor(Color.pink);  
15.    g.fillOval(170,200,30,30);  
16.    g.drawArc(90,150,30,30,30,270);  
17.    g.fillArc(270,150,30,30,0,180);  
18.  
19. }  
20. }
```

```
1. <html>  
2. <body>  
3. <applet code="GraphicsDemo.class" width="300" height="300">  
4. </applet>  
5. </body>  
6. </html>
```

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

1. **public Image getImage(URL u, String image){}**

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

```
1. import java.awt.*;
2. import java.applet.*;
3.
4.
5. public class DisplayImage extends Applet {
6.
7.     Image picture;
8.
9.     public void init() {
10.         picture = getImage(getDocumentBase(), "sonoo.jpg");
11.     }
12.
13.     public void paint(Graphics g) {
14.         g.drawImage(picture, 30, 30, this);
15.     }
16.
17. }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method is ImageObserver object. The Component class implements ImageObserver interface. So current class object implements ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

1. <html>
2. <body>
3. <applet code="DisplayImage.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

A Simple Banner Applet

To demonstrate repaint(), a simple banner applet is developed. This applet scrolls a message, from right to left, across the applet's window. Since the scrolling of the message is a repetitive task, it is performed by a separate thread, created by the applet when it is initialized. The banner applet is shown here:

```
/* A simple banner applet.
```

This applet creates a thread that scrolls
the message contained in msg right to left
across the applet's window.

```
*/
```

```
import java.awt.*;
```

```
import java.applet.*;
```

```
/*
```

```
<applet code="SimpleBanner" width=300 height=50>
```

```
</applet>
```

```
*/  
  
public class SimpleBanner extends Applet implements Runnable {  
  
    String msg = " A Simple Moving Banner.";  
  
    Thread t = null;  
  
    int state;  
  
    volatile boolean stopFlag;  
  
    // Set colors and initialize thread.  
  
    public void init() {  
  
        setBackground(Color.cyan);  
  
        setForeground(Color.red);  
  
    }  

```

23-ch23.indd 757 14/02/14 5:12 PMCompRef_2010 / Java The Complete Reference, Ninth Edition /Schildt / 007180 855-8

758 PART II The Java Library

```
// Start thread  
  
public void start() {  
  
    t = new Thread(this);  
  
    stopFlag = false;  
  
    t.start();  
  
}  
  
// Entry point for the thread that runs the banner.  
  
public void run() {  
  
    // Redisplay banner  
  
    for( ; ; ) {  
  
        try {  

```

```
repaint();

Thread.sleep(250);

if(stopFlag)

break;

} catch(InterruptedException e) { }

}

}

// Pause the banner.

public void stop() {

stopFlag = true;

t = null;

}

// Display the banner.

public void paint(Graphics g) {

char ch;

ch = msg.charAt(0);

msg = msg.substring(1, msg.length());

msg += ch;

g.drawString(msg, 50, 30);

}

}

}
```

Following is sample output:

Let's take a close look at how this applet operates. First, notice that SimpleBanner extends Applet, as expected, but it also implements Runnable. This is necessary, since the

Chapter 23 The Applet Class 759

Part II

applet will be creating a second thread of execution that will be used to scroll the banner.

Inside `init()`, the foreground and background colors of the applet are set.

After initialization, the run-time system calls `start()` to start the applet running. Inside `start()`, a new thread of execution is created and assigned to the `Thread` variable `t`. Then, the boolean variable `stopFlag`, which controls the execution of the applet, is set to false.

Next, the thread is started by a call to `t.start()`. Remember that `t.start()` calls a method defined by `Thread`, which causes `run()` to begin executing. It does not cause a call to the version of `start()` defined by `Applet`. These are two separate methods.

Inside `run()`, a call to `repaint()` is made. This eventually causes the `paint()` method to be called, and the rotated contents of `msg` are displayed. Between each iteration, `run()` sleeps for a quarter of a second. The net effect is that the contents of `msg` are scrolled right to left in a constantly moving display. The `stopFlag` variable is checked on each iteration.

When it is true, the `run()` method terminates.

If a browser is displaying the applet when a new page is viewed, the `stop()` method is called, which sets `stopFlag` to true, causing `run()` to terminate. This is the mechanism used to stop the thread when its page is no longer in view. When the applet is brought back into view, `start()` is once again called, which starts a new thread to execute the banner.

Using the Status Window

In addition to displaying information in its window, an applet can also output a message to the status window of the browser or applet viewer on which it is running. To do so, call `showStatus()` with the string that you want displayed. The status window is a good place to

give the user feedback about what is occurring in the applet, suggest options, or possibly report some types of errors. The status window also makes an excellent debugging aid, because it gives you an easy way to output information about your applet.

The following applet demonstrates showStatus():

```
// Using the Status Window.
```

```
import java.awt.*;
```

```
import java.applet.*;
```

```
/*
```

```
<applet code="StatusWindow" width=300 height=50>
```

```
</applet>
```

```
*/
```

```
public class StatusWindow extends Applet {
```

```
    public void init() {
```

```
        setBackground(Color.cyan);
```

```
    }
```

```
    // Display msg in applet window.
```

```
    public void paint(Graphics g) {
```

```
        g.drawString("This is in the applet window.", 10, 20);
```

```
        showStatus("This is shown in the status window.");
```

```
    }
```

```
}
```

23-ch23.indd 759 14/02/14 5:12 PMCompRef_2010 / Java The Complete Reference, Ninth Edition /Schildt / 007180 855-8

760 PART II The Java Library

Sample output from this program is shown here:

The HTML APPLET Tag

As mentioned earlier, at the time of this writing, Oracle recommends that the APPLET tag be used to manually start an applet when JNLP is not used. An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers will allow many applets on a single page. So far, we have been using only a simplified form of the APPLET tag. Now it is time to take a closer look at it.

The syntax for a fuller form of the APPLET tag is shown here. Bracketed items are optional.

```
< APPLET  
[CODEBASE = codebaseURL]  
CODE = appletFile  
[ALT = alternateText]  
[NAME = appletInstanceName]  
WIDTH = pixels HEIGHT = pixels  
[ALIGN = alignment ]  
[VSPACE = pixels] [HSPACE = pixels]  
>  
[< PARAM NAME = AttributeName VALUE =AttributeValue>]  
[< PARAM NAME = AttributeName2 VALUE =AttributeValue>]  
...  
[HTML Displayed in the absence of Java]  
</APPLET>
```

Let's take a look at each part now.

CODEBASE CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class

file (specified by the CODE tag). The HTML document's URL directory is used as the CODEBASE if this attribute is not specified.

CODE CODE is a required attribute that gives the name of the file containing your applet's compiled .class file. This file is relative to the code base URL of the applet, which :

```
// Use Parameters

import java.awt.*;
import java.applet.*;

/*
<applet code="ParamDemo" width=300 height=80>
<param name=fontName value=Courier>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
</applet>
*/
public class ParamDemo extends Applet {
    String fontName;
    int fontSize;
    float leading;
    boolean active;
    // Initialize the string to be displayed.
    public void start() {
        String param;
        fontName = getParameter("fontName");
```

```
if(fontName == null)
    fontName = "Not Found";
param = getParameter("fontSize");
try {
    if(param != null)
        fontSize = Integer.parseInt(param);
    else
        fontSize = 0;
} catch(NumberFormatException e) {
    fontSize = -1;
}
param = getParameter("leading");
try {
    if(param != null)
        leading = Float.valueOf(param).floatValue();
    else
        leading = 0;
} catch(NumberFormatException e) {
    leading = -1;
}
param = getParameter("accountEnabled");
if(param != null)
    active = Boolean.valueOf(param).booleanValue();
}
// Display parameters.
```

```
public void paint(Graphics g) {  
    g.drawString("Font name: " + fontName, 0, 10);  
    g.drawString("Font size: " + fontSize, 0, 26);  
    g.drawString("Leading: " + leading, 0, 42);  
    g.drawString("Account Active: " + active, 0, 58);  
}  
}
```

Sample output from this program is shown here:

As the program shows, you should test the return values from `getParameter()`. If a parameter isn't available, `getParameter()` will return null. Also, conversions to numeric types must be attempted in a try statement that catches `NumberFormatException`. Uncaught exceptions should never occur within an applet.

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The `java.awt.event` package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes

Listener Interfaces

ActionEvent	ActionListener
-------------	----------------

MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
 - `public void addActionListener(ActionListener a){}`

- **MenuItem**
 - public void addActionListener(ActionListener a){}
 - **TextField**
 - public void addActionListener(ActionListener a){}
 - public void addTextListener(TextListener a){}
 - **TextArea**
 - public void addTextListener(TextListener a){}
 - **Checkbox**
 - public void addItemListener(ItemListener a){}
 - **Choice**
 - public void addItemListener(ItemListener a){}
 - **List**
 - public void addActionListener(ActionListener a){}
 - public void addItemListener(ItemListener a){}
-

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing ActionListener

1. **import** java.awt.*;
2. **import** java.awt.event.*;
3. **class** AEvent **extends** Frame **implements** ActionListener{
4. TextField tf;
5. AEvent(){
- 6.

```
7. //create components
8. tf=new TextField();
9. tf.setBounds(60,50,170,20);
10. Button b=new Button("click me");
11. b.setBounds(100,120,80,30);
12.
13. //register listener
14. b.addActionListener(this);//passing current instance
15.
16. //add components and set size, layout and visibility
17. add(b);add(tf);
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21. }
22. public void actionPerformed(ActionEvent e){
23. tf.setText("Welcome");
24. }
25. public static void main(String args[]){
26. new AEvent();
27. }
28. }
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.



2) Java event handling by outer class

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent2 extends Frame{
4.     TextField tf;
5.     AEvent2(){
6.         //create components
7.         tf=new TextField();
8.         tf.setBounds(60,50,170,20);
9.         Button b=new Button("click me");
10.        b.setBounds(100,120,80,30);
11.        //register listener
12.        Outer o=new Outer(this);
13.        b.addActionListener(o);//passing outer class instance
14.        //add components and set size, layout and visibility
15.        add(b);add(tf);
```

```
16. setSize(300,300);
17. setLayout(null);
18. setVisible(true);
19. }
20. public static void main(String args[]){
21. new AEvent2();
22. }
23. }

1. import java.awt.event.*;
2. class Outer implements ActionListener{
3. AEvent2 obj;
4. Outer(AEvent2 obj){
5. this.obj=obj;
6. }
7. public void actionPerformed(ActionEvent e){
8. obj.tf.setText("welcome");
9. }
10. }
```

3) Java event handling by anonymous class

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent3 extends Frame{
4. TextField tf;
5. AEvent3(){
6. tf=new TextField();
7. tf.setBounds(60,50,170,20);
8. Button b=new Button("click me");
9. b.setBounds(50,120,80,30);
10.
11. b.addActionListener(new ActionListener(){
12. public void actionPerformed(){
13. tf.setText("hello");
```

```
14. }
15. });
16. add(b);add(tf);
17. setSize(300,300);
18. setLayout(null);
19. setVisible(true);
20. }
21. public static void main(String args[]){
22. new AEvent3();
23. }
24. }
```

Event Class Description

ActionEvent Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

AdjustmentEvent Generated when a scroll bar is manipulated.

ComponentEvent Generated when a component is hidden, moved, resized, or becomes visible.

ContainerEvent Generated when a component is added to or removed from a container.

FocusEvent Generated when a component gains or loses keyboard focus.

InputEvent Abstract superclass for all component input event classes.

ItemEvent Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.

KeyEvent Generated when input is received from the keyboard.

MouseEvent Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.

MouseWheelEvent Generated when the mouse wheel is moved.

TextEvent Generated when the value of a text area or text field is changed.

WindowEvent Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit

Event Listener Interfaces

As explained, the delegation event model has two parts: sources and listeners. As it relates to this chapter, listeners are created by implementing one or more of the interfaces defined by the `java.awt.event` package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument. Table 24-3 lists

Event Source Examples

Event Source Description

Button Generates action events when the button is pressed.

Check box Generates item events when the check box is selected or deselected.

Choice Generates item events when the choice is changed.

List Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.

Menu item Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.

Scroll bar Generates adjustment events when the scroll bar is manipulated.

Text components Generates text events when the user enters a character.

Window Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Commonly Used Event Listener Interfaces

Interface Description

ActionListener Defines one method to receive action events.

AdjustmentListener Defines one method to receive adjustment events.

ComponentListener Defines four methods to recognize when a component is hidden, moved,

resized, or shown.

ContainerListener Defines two methods to recognize when a component is added to or removed from a container.

FocusListener Defines two methods to recognize when a component gains or loses keyboard focus.

ItemListener Defines one method to recognize when the state of an item changes.

KeyListener Defines three methods to recognize when a key is pressed, released, or typed.

MouseListener Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.

MouseMotionListener Defines two methods to recognize when the mouse is dragged or moved.

MouseWheelListener Defines one method to recognize when the mouse wheel is moved.

TextListener Defines one method to recognize when a text value changes.

WindowFocusListener Defines two methods to recognize when a window gains or loses input focus.

WindowListener Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit

The ActionListener Interface

This interface defines the actionPerformed() method that is invoked when an action event occurs. Its general form is shown here:

void actionPerformed(ActionEvent ae)

The AdjustmentListener Interface

This interface defines the adjustmentValueChanged() method that is invoked when an adjustment event occurs. Its general form is shown here:

void adjustmentValueChanged(AdjustmentEvent ae)

The ComponentListener Interface

This interface defines four methods that are invoked when a component is resized, moved, shown, or hidden. Their general forms are shown here:

void componentResized(ComponentEvent ce)

void componentMoved(ComponentEvent ce)

void componentShown(ComponentEvent ce)

void componentHidden(ComponentEvent ce)

The ContainerListener Interface

This interface contains two methods. When a component is added to a container, componentAdded() is invoked. When a component is removed from a container, componentRemoved() is invoked. Their general forms are shown here:

void componentAdded(ContainerEvent ce)

void componentRemoved(ContainerEvent ce)

The FocusListener Interface

This interface defines two methods. When a component obtains keyboard focus, `focusGained()` is invoked. When a component loses keyboard focus, `focusLost()` is called. Their general forms are shown here:

void focusGained(FocusEvent fe)

void focusLost(FocusEvent fe)

The ItemListener Interface

This interface defines the `itemStateChanged()` method that is invoked when the state of an item changes. Its general form is shown here:

void itemStateChanged(ItemEvent ie)

The KeyListener Interface

This interface defines three methods. The `keyPressed()` and `keyReleased()` methods are invoked when a key is pressed and released, respectively. The `keyTyped()` method is invoked when a character has been entered.

For example, if a user presses and releases the a key, three events are generated in sequence: key pressed, typed, and released. If a user presses and releases the home key, two key events are generated in sequence: key pressed and released.

The general forms of these methods are shown here:

void keyPressed(KeyEvent ke)

void keyReleased(KeyEvent ke)

void keyTyped(KeyEvent ke)

The MouseListener Interface

This interface defines five methods. If the mouse is pressed and released at the same point, `mouseClicked()` is invoked. When the mouse enters a component, the `mouseEntered()` method is called. When it leaves, `mouseExited()` is called. The `mousePressed()` and `mouseReleased()` methods are invoked when the mouse is pressed and released, respectively.

The general forms of these methods are shown here:

```
void mouseClicked(MouseEvent me)  
void mouseEntered(MouseEvent me)  
void mouseExited(MouseEvent me)  
void mousePressed(MouseEvent me)  
void mouseReleased(MouseEvent me)
```

The MouseMotionListener Interface

This interface defines two methods. The `mouseDragged()` method is called multiple times as the mouse is dragged. The `mouseMoved()` method is called multiple times as the mouse is moved. Their general forms are shown here:

```
void mouseDragged(MouseEvent me)  
void mouseMoved(MouseEvent me)
```

The MouseWheelListener Interface

This interface defines the `mouseWheelMoved()` method that is invoked when the mouse wheel is moved. Its general form is shown here:

```
void mouseWheelMoved(MouseWheelEvent mwe)
```

The TextListener Interface

This interface defines the `textValueChanged()` method that is invoked when a change occurs in a text area or text field. Its general form is shown here:

```
void textValueChanged(TextEvent te)
```

The WindowFocusListener Interface

This interface defines two methods: `windowGainedFocus()` and `windowLostFocus()`. These

are called when a window gains or loses input focus. Their general forms are shown here:

```
void windowGainedFocus(WindowEvent we)
```

```
void windowLostFocus(WindowEvent we)
```

The WindowListener Interface

This interface defines seven methods. The `windowActivated()` and `windowDeactivated()` methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the `windowIconified()` method is called. When a window is deiconified, the `windowDeiconified()` method is called. When a window is opened or closed, the `windowOpened()` or `windowClosed()` methods are called, respectively. The `windowClosing()`

method is called when a window is being closed. The general forms of these methods are

```
void windowActivated(WindowEvent we)
```

```
void windowClosed(WindowEvent we)
```

```
void windowClosing(WindowEvent we)
```

```
void windowDeactivated(WindowEvent we)
```

```
void windowDeiconified(WindowEvent we)
```

```
void windowIconified(WindowEvent we)
```

```
void windowOpened(WindowEvent we)

// Demonstrate the mouse event handlers.

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/

public class MouseEvents extends Applet
implements MouseListener, MouseMotionListener {
    String msg = "";
    int mouseX = 0, mouseY = 0; // coordinates of mouse

    public void init() {
        addMouseListener(this);
        addMouseMotionListener(this);
    }

    // Handle mouse clicked.

    public void mouseClicked(MouseEvent me) {
        // save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse clicked.";
        repaint();
    }
}
```

```
}

// Handle mouse entered.

public void mouseEntered(MouseEvent me) {

    // save coordinates

    mouseX = 0;

    mouseY = 10;

    msg = "Mouse entered.";

    repaint();

}

public void mouseExited(MouseEvent me) {

    // save coordinates

    mouseX = 0;

    mouseY = 10;

    msg = "Mouse exited.";

    repaint();

}

// Handle button pressed.

public void mousePressed(MouseEvent me) {

    // save coordinates

    mouseX = me.getX();

    mouseY = me.getY();

    msg = "Down";

    repaint();

}

// Handle button released.
```

```
public void mouseReleased(MouseEvent me) {  
    // save coordinates  
  
    mouseX = me.getX();  
  
    mouseY = me.getY();  
  
    msg = "Up";  
  
    repaint();  
}  
  
// Handle mouse dragged.  
  
public void mouseDragged(MouseEvent me) {  
    // save coordinates  
  
    mouseX = me.getX();  
  
    mouseY = me.getY();  
  
    msg = "*";  
  
    showStatus("Dragging mouse at " + mouseX + ", " + mouseY);  
  
    repaint();  
}  
  
// Handle mouse moved.  
  
public void mouseMoved(MouseEvent me) {  
    // show status  
  
    showStatus("Moving mouse at " + me.getX() + ", " + me.getY());  
}  
  
// Display msg in applet window at current X,Y location.  
  
public void paint(Graphics g) {  
    g.drawString(msg, mouseX, mouseY);  
}
```

```
}
```

```
// Demonstrate the key event handlers.

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*
<applet code="SimpleKey" width=300 height=100>
</applet>
*/

public class SimpleKey extends Applet
implements KeyListener {

String msg = "";
int X = 10, Y = 20; // output coordinates

public void init() {
addKeyListener(this);
}

public void keyPressed(KeyEvent ke) {
showStatus("Key Down");
}

public void keyReleased(KeyEvent ke) {
showStatus("Key Up");
}
```

```
public void keyTyped(KeyEvent ke) {  
    msg += ke.getKeyChar();  
    repaint();  
}  
  
// Display keystrokes.  
  
public void paint(Graphics g) {  
    g.drawString(msg, X, Y);  
}
```

Adapter Classes

Java provides a special feature, called an adapter class, that can simplify the creation of event

handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

You can define a new class to act as an event listener by extending one of the adapter classes

and implementing only those events in which you are interested.

For example, the MouseMotionAdapter class has two methods, mouseDragged() and mouseMoved(), which are the methods defined by the MouseMotionListener interface. If you

were interested in only mouse drag events, then you could simply extend MouseMotionAdapter

and override mouseDragged(). The empty implementation of mouseMoved() would handle the mouse motion events for you.

Table 24-4 lists several commonly used adapter classes in java.awt.event and notes the

interface that each implements.

24-ch24.indd 791 14/02/14 5:13 PMCompRef_2010 / Java The Complete Reference, Ninth Edition /Schildt / 007180 855-8

792 PART II The Java Library

The following example demonstrates an adapter. It displays a message in the status bar of an applet viewer or browser when the mouse is clicked or dragged. However, all other mouse events are silently ignored. The program has three classes. AdapterDemo extends Applet. Its init() method creates an instance of MyMouseAdapter and registers that object to

receive notifications of mouse events. It also creates an instance of MyMouseMotionAdapter

and registers that object to receive notifications of mouse motion events. Both of the constructors take a reference to the applet as an argument.

MyMouseAdapter extends MouseAdapter and overrides the mouseClicked() method. The other mouse events are silently ignored by code inherited from the MouseAdapter class. MyMouseMotionAdapter extends MouseMotionAdapter and overrides the mouseDragged() method. The other mouse motion event is silently ignored by code inherited from the MouseMotionAdapter class. (MouseAdaptor also provides an empty implementation for MouseMotionListener. However, for the sake of illustration, this example handles each separately.)

Note that both of the event listener classes save a reference to the applet. This information is provided as an argument to their constructors and is used later to invoke the showStatus() method.

```
// Demonstrate an adapter.
```

```
import java.awt.*;  
import java.awt.event.*;
```

```
import java.applet.*;

<applet code="AdapterDemo" width=300 height=100>
</applet>

*/



public class AdapterDemo extends Applet {

    public void init() {

        addMouseListener(new MyMouseAdapter(this));

        addMouseMotionListener(new MyMouseMotionAdapter(this));

    }

}
```

Adapter Class Listener Interface

ComponentAdapter **ComponentListener**

ContainerAdapter **ContainerListener**

FocusAdapter **FocusListener**

KeyAdapter **KeyListener**

MouseAdapter **MouseListener and (as of JDK 6)**

MouseMotionListener and MouseWheelListener

MouseMotionAdapter

MouseMotionListener

WindowAdapter

WindowListener,

WindowFocusListener,

WindowStateListene

```
// Demonstrate an adapter.

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*
<applet code="AdapterDemo" width=300 height=100>
</applet>
*/

public class AdapterDemo extends Applet {

    public void init() {
        addMouseListener(new MyMouseAdapter(this));
        addMouseMotionListener(new MyMouseMotionAdapter(this));
    }
}

class MyMouseAdapter extends MouseAdapter {

    AdapterDemo adapterDemo;

    public MyMouseAdapter(AdapterDemo adapterDemo) {
        this.adapterDemo = adapterDemo;
    }

    // Handle mouse clicked.

    public void mouseClicked(MouseEvent me) {
        adapterDemo.showStatus("Mouse clicked");
    }
}
```

```
class MyMouseMotionAdapter extends MouseMotionAdapter {  
    AdapterDemo adapterDemo;  
  
    public MyMouseMotionAdapter(AdapterDemo adapterDemo) {  
        this.adapterDemo = adapterDemo;  
    }  
  
    // Handle mouse dragged.  
  
    public void mouseDragged(MouseEvent me) {  
        adapterDemo.showStatus("Mouse dragged");  
    }  
}
```

Inner Classes

the basics of inner classes were explained. Here, you will see why they are important. Recall that an inner class is a class defined within another class, or even within an expression. This section illustrates how inner classes can be used to simplify the code when using event adapter classes.

To understand the benefit provided by inner classes, consider the applet shown in the following listing. It does not use an inner class. Its goal is to display the string "Mouse Pressed" in the status bar of the applet viewer or browser when the mouse is pressed. There are two top-level classes in this program. MousePressedDemo extends Applet, and MyMouseAdapter extends MouseAdapter. The init() method of MousePressedDemo instantiates MyMouseAdapter and provides this object as an argument to the addMouseListener()

method.

Notice that a reference to the applet is supplied as an argument to the MyMouseAdapter constructor. This reference is stored in an instance variable for later use by the mousePressed() method. When the mouse is pressed, it invokes the showStatus() method of the applet

```
// This applet does NOT use an inner class.

import java.applet.*;
import java.awt.event.*;
/*
<applet code="MousePressedDemo" width=200 height=100>
</applet>
*/
public class MousePressedDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter(this));
    }
}
class MyMouseAdapter extends MouseAdapter {
    MousePressedDemo mousePressedDemo;
    public MyMouseAdapter(MousePressedDemo mousePressedDemo) {
        this.mousePressedDemo = mousePressedDemo;
    }
    public void mousePressed(MouseEvent me) {
        mousePressedDemo.showStatus("Mouse Pressed.");
    }
}
```

```

}

}

// Inner class demo.

import java.applet.*;
import java.awt.event.*;
/*
<applet code="InnerClassDemo" width=200 height=100>
</applet>
*/
public class InnerClassDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter());
    }
    class MyMouseAdapter extends MouseAdapter {
        public void mousePressed(MouseEvent me) {
            showStatus("Mouse Pressed");
        }
    }
}

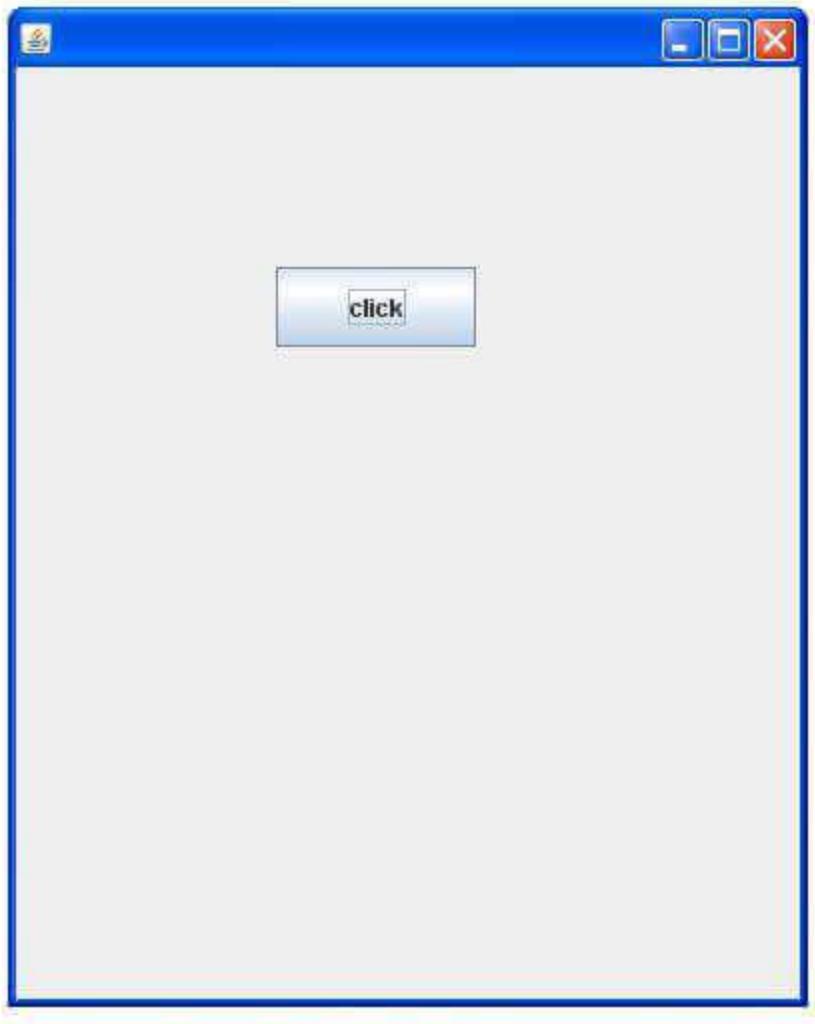
```

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;  
2. public class FirstSwingExample {  
3.     public static void main(String[] args) {  
4.         JFrame f=new JFrame(); //creating instance of JFrame  
5.         JButton b=new JButton("click"); //creating instance of JButton  
6.         b.setBounds(130,100,100, 40); //x axis, y axis, width, height  
7.         f.add(b); //adding button in JFrame  
8.         f.setSize(400,500); //400 width and 500 height  
9.         f.setLayout(null); //using no layout managers  
10.        f.setVisible(true); //making the frame visible  
11.    }  
12. }
```



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

1. **import** javax.swing.*;
2. **public class** Simple {
3. JFrame f;
4. Simple(){
5. f=**new** JFrame();*//creating instance of JFrame*
6. }

```
7. JButton b=new JButton("click");//creating instance of JButton
8. b.setBounds(130,100,100, 40);
9.
10. f.add(b);//adding button in JFrame
11.
12. f.setSize(400,500);//400 width and 500 height
13. f.setLayout(null);//using no layout managers
14. f.setVisible(true);//making the frame visible
15. }
16.
17. public static void main(String[] args) {
18. Simple s1=new Simple();
19. }
20. }
```

The setBounds(int xaxis, int yaxis, int width, int height) is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
1. import javax.swing.*;
2. public class Simple2 extends JFrame{//inheriting JFrame
3. JFrame f;
4. Simple2(){
5. JButton b=new JButton("click");//create button
6. b.setBounds(130,100,100, 40);
7.
8. add(b);//adding button on frame
9. setSize(400,500);
10.setLayout(null);
```

```
11. setVisible(true);  
12. }  
13. public static void main(String[] args) {  
14.     new Simple2();  
15. }
```

Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

```
1. public class JButton extends AbstractButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the <u>action listener</u> to this object

Java JButton Example

```

1. import javax.swing.*;
2. public class ButtonExample {
3.     public static void main(String[] args) {
4.         JFrame f=new JFrame("Button Example");
5.         JButton b=new JButton("Click Here");
6.         b.setBounds(50,100,95,30);
7.         f.add(b);
8.         f.setSize(400,400);
9.         f.setLayout(null);
10.        f.setVisible(true);
11.    }
12. }
```

Output:



Java JButton Example with ActionListener

```
1. import java.awt.event.*;
2. import javax.swing.*;
3. public class ButtonExample {
4.     public static void main(String[] args) {
5.         JFrame f=new JFrame("Button Example");
6.         final JTextField tf=new JTextField();
7.         tf.setBounds(50,50, 150,20);
8.         JButton b=new JButton("Click Here");
9.         b.setBounds(50,100,95,30);
10.        b.addActionListener(new ActionListener(){
11.            public void actionPerformed(ActionEvent e){
12.                tf.setText("Welcome to Javatpoint.");
13.            }
14.        });
15.        f.add(b);f.add(tf);
16.        f.setSize(400,400);
17.        f.setLayout(null);
18.        f.setVisible(true);
19.    }
20. }
```

Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string as its text.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.

void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X-axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X-axis.

Java JLabel Example

```

1. import javax.swing.*;
2. class LabelExample
3. {
4.     public static void main(String args[])
5.     {
6.         JFrame f= new JFrame("Label Example");
7.         JLabel l1,l2;
8.         l1=new JLabel("First Label.");
9.         l1.setBounds(50,50, 100,30);
10.        l2=new JLabel("Second Label.");
11.        l2.setBounds(50,100, 100,30);
12.        f.add(l1); f.add(l2);
13.        f.setSize(300,300);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.    }
17. }
```

Output:



Java JLabel Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class LabelExample extends Frame implements ActionListener{
5.     JTextField tf; JLabel l; JButton b;
6.     LabelExample(){
7.         tf=new JTextField();
8.         tf.setBounds(50,50, 150,20);
9.         l=new JLabel();
10.        l.setBounds(50,100, 250,20);
11.        b=new JButton("Find IP");
12.        b.setBounds(50,150,95,30);
13.        b.addActionListener(this);
14.        add(b);add(tf);add(l);
15.        setSize(400,400);
16.        setLayout(null);
17.        setVisible(true);
18.    }
19.    public void actionPerformed(ActionEvent e) {
20.        try{
```

```
21.     String host=tf.getText();
22.     String ip=java.net.InetAddress.getByName(host).getHostAddress();
23.     l.setText("IP of " +host+ " is: " +ip);
24.     }catch(Exception ex){System.out.println(ex);}
25.   }
26.   public static void main(String[] args) {
27.     new LabelExample();
28.   }
}
```

Output:



Java JLabel

The object of `JLabel` class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits `JComponent` class.

JLabel class declaration

Let's see the declaration for `javax.swing.JLabel` class.

1. **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string as its text.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X-axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X-axis.

Java JLabel Example

1. **import** javax.swing.*;

```
2. class LabelExample
3. {
4.     public static void main(String args[])
5.     {
6.         JFrame f= new JFrame("Label Example");
7.         JLabel l1,l2;
8.         l1=new JLabel("First Label.");
9.         l1.setBounds(50,50, 100,30);
10.        l2=new JLabel("Second Label.");
11.        l2.setBounds(50,100, 100,30);
12.        f.add(l1); f.add(l2);
13.        f.setSize(300,300);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.    }
17. }
```

Output:



Java JLabel Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.*;
```

```
3. import java.awt.event.*;
4. public class LabelExample extends Frame implements ActionListener{
5.     JTextField tf; JLabel l; JButton b;
6.     LabelExample(){
7.         tf=new JTextField();
8.         tf.setBounds(50,50, 150,20);
9.         l=new JLabel();
10.        l.setBounds(50,100, 250,20);
11.        b=new JButton("Find IP");
12.        b.setBounds(50,150,95,30);
13.        b.addActionListener(this);
14.        add(b);add(tf);add(l);
15.        setSize(400,400);
16.        setLayout(null);
17.        setVisible(true);
18.    }
19.    public void actionPerformed(ActionEvent e) {
20.        try{
21.            String host=tf.getText();
22.            String ip=java.net.InetAddress.getByName(host).getHostAddress();
23.            l.setText("IP of " +host+ " is: " +ip);
24.        }catch(Exception ex){System.out.println(ex);}
25.    }
26.    public static void main(String[] args) {
27.        new LabelExample();
28.    }
}
```

Output:



Java JTextArea

The object of a `JTextArea` class is a multi line region that displays text. It allows the editing of multiple line text. It inherits `JTextComponent` class

JTextArea class declaration

Let's see the declaration for `javax.swing.JTextArea` class.

1. `public class JTextArea extends JTextComponent`

Commonly used Constructors:

Constructor	Description
<code>JTextArea()</code>	Creates a text area that displays no text initially.
<code>JTextArea(String s)</code>	Creates a text area that displays specified text initially.

JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that contains the specified string.

Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example

```

1. import javax.swing.*;
2. public class TextAreaExample
3. {
4.     TextAreaExample(){
5.         JFrame f= new JFrame();
6.         JTextArea area=new JTextArea("Welcome to javatpoint");
7.         area.setBounds(10,30, 200,200);
8.         f.add(area);
9.         f.setSize(300,300);
10.        f.setLayout(null);

```

```
11.     f.setVisible(true);
12. }
13. public static void main(String args[])
14. {
15.     new TextAreaExample();
16. }
```

Output:

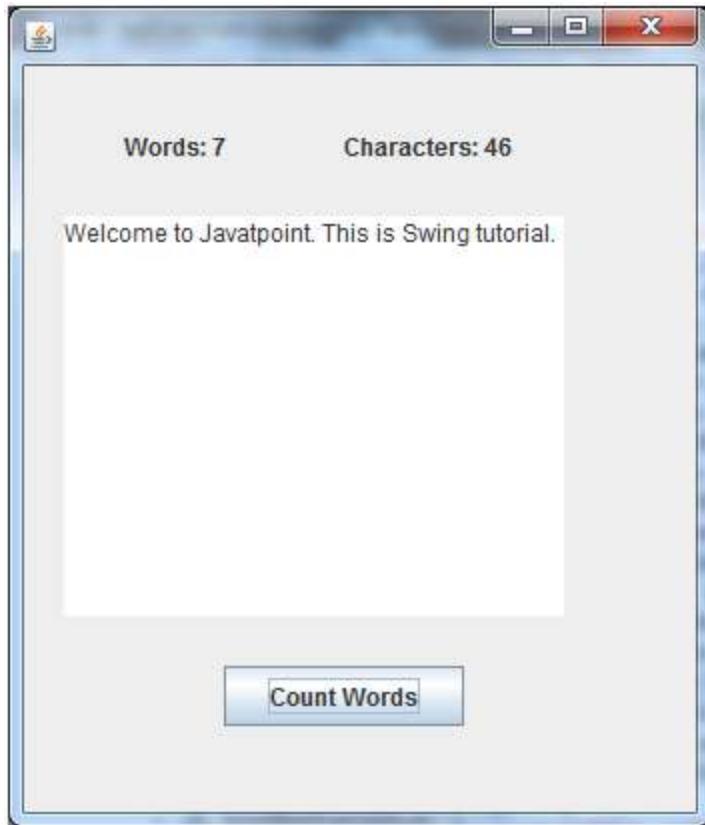


Java JTextArea Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class TextAreaExample implements ActionListener{
4.     JLabel l1,l2;
5.     JTextArea area;
6.     JButton b;
7.     TextAreaExample() {
8.         JFrame f= new JFrame();
9.         l1=new JLabel();
10.        l1.setBounds(50,25,100,30);
```

```
11. l2=new JLabel();
12. l2.setBounds(160,25,100,30);
13. area=new JTextArea();
14. area.setBounds(20,75,250,200);
15. b=new JButton("Count Words");
16. b.setBounds(100,300,120,30);
17. b.addActionListener(this);
18. f.add(l1);f.add(l2);f.add(area);f.add(b);
19. f.setSize(450,450);
20. f.setLayout(null);
21. f.setVisible(true);
22. }
23. public void actionPerformed(ActionEvent e){
24.     String text=area.getText();
25.     String words[]=text.split("\s");
26.     l1.setText("Words: "+words.length);
27.     l2.setText("Characters: "+text.length());
28. }
29. public static void main(String[] args) {
30.     new TextAreaExample();
31. }
32. }
```

Output:



Java JCheckBox

The `JCheckBox` class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a `CheckBox` changes its state from "on" to "off" or from "off" to "on". It inherits [JToggleButton](#) class.

JCheckBox class declaration

Let's see the declaration for `javax.swing.JCheckBox` class.

1. `public class JCheckBox extends JToggleButton implements Accessible`

Commonly used Constructors:

Constructor	Description
-------------	-------------

JJCheckBox()	Creates an initially unselected check box button with no text, no icon, and no mnemonic.
JCheckBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action object.

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this component.
protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example

```

1. import javax.swing.*;
2. public class CheckBoxExample
3. {
4.     CheckBoxExample(){
5.         JFrame f= new JFrame("CheckBox Example");
6.         JCheckBox checkBox1 = new JCheckBox("C++");
7.         checkBox1.setBounds(100,100, 50,50);
8.         JCheckBox checkBox2 = new JCheckBox("Java", true);
9.         checkBox2.setBounds(100,150, 50,50);
10.        f.add(checkBox1);
11.        f.add(checkBox2);
12.        f.setSize(400,400);
13.        f.setLayout(null);
14.        f.setVisible(true);

```

```
15.    }
16. public static void main(String args[])
17. {
18. new CheckBoxExample();
19. }}
```

Output:

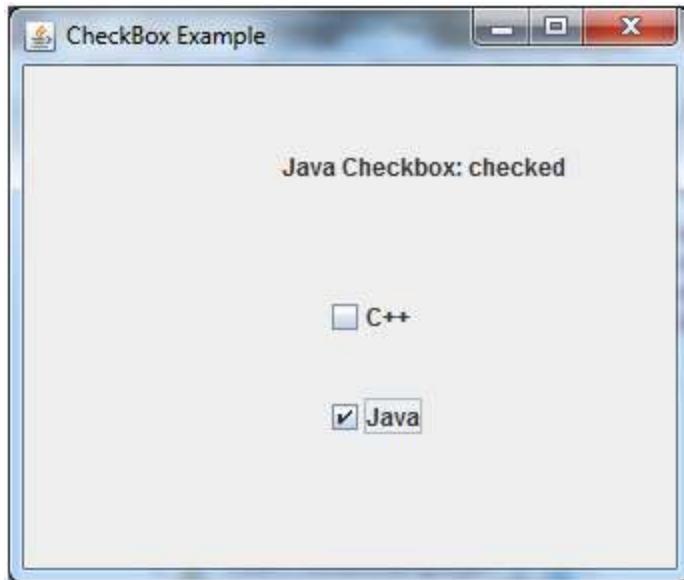


Java JCheckBox Example with ItemListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample
4. {
5.     CheckBoxExample(){
6.         JFrame f= new JFrame("CheckBox Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        JCheckBox checkbox1 = new JCheckBox("C++");
```

```
11. checkbox1.setBounds(150,100, 50,50);
12. JCheckBox checkbox2 = new JCheckBox("Java");
13. checkbox2.setBounds(150,150, 50,50);
14. f.add(checkbox1); f.add(checkbox2); f.add(label);
15. checkbox1.addItemListener(new ItemListener() {
16.     public void itemStateChanged(ItemEvent e) {
17.         label.setText("C++ Checkbox: "
18.             + (e.getStateChange() == 1 ? "checked" : "unchecked"));
19.     }
20. });
21. checkbox2.addItemListener(new ItemListener() {
22.     public void itemStateChanged(ItemEvent e) {
23.         label.setText("Java Checkbox: "
24.             + (e.getStateChange() == 1 ? "checked" : "unchecked"));
25.     }
26. });
27. f.setSize(400,400);
28. f.setLayout(null);
29. f.setVisible(true);
30. }
31. public static void main(String args[])
32. {
33.     new CheckBoxExample();
34. }
35. }
```

Output:



Java JCheckBox Example: Food Order

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample extends JFrame implements ActionListener{
4.     JLabel l;
5.     JCheckBox cb1,cb2,cb3;
6.     JButton b;
7.     CheckBoxExample(){
8.         l=new JLabel("Food Ordering System");
9.         l.setBounds(50,50,300,20);
10.        cb1=new JCheckBox("Pizza @ 100");
11.        cb1.setBounds(100,100,150,20);
12.        cb2=new JCheckBox("Burger @ 30");
13.        cb2.setBounds(100,150,150,20);
14.        cb3=new JCheckBox("Tea @ 10");
15.        cb3.setBounds(100,200,150,20);
16.        b=new JButton("Order");
17.        b.setBounds(100,250,80,30);
18.        b.addActionListener(this);
19.        add(l);add(cb1);add(cb2);add(cb3);add(b);
```

```
20.     setSize(400,400);
21.     setLayout(null);
22.     setVisible(true);
23.     setDefaultCloseOperation(EXIT_ON_CLOSE);
24. }
25. public void actionPerformed(ActionEvent e){
26.     float amount=0;
27.     String msg="";
28.     if(cb1.isSelected()){
29.         amount+=100;
30.         msg="Pizza: 100\n";
31.     }
32.     if(cb2.isSelected()){
33.         amount+=30;
34.         msg+="Burger: 30\n";
35.     }
36.     if(cb3.isSelected()){
37.         amount+=10;
38.         msg+="Tea: 10\n";
39.     }
40.     msg+="-----\n";
41.     JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
42. }
43. public static void main(String[] args) {
44.     new CheckBoxExample();
45. }
46. }
```

Output:



ava JButton

The JButton class is used to create a button. It is used to perform some action when clicked. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. **public class** JRadioButton **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected

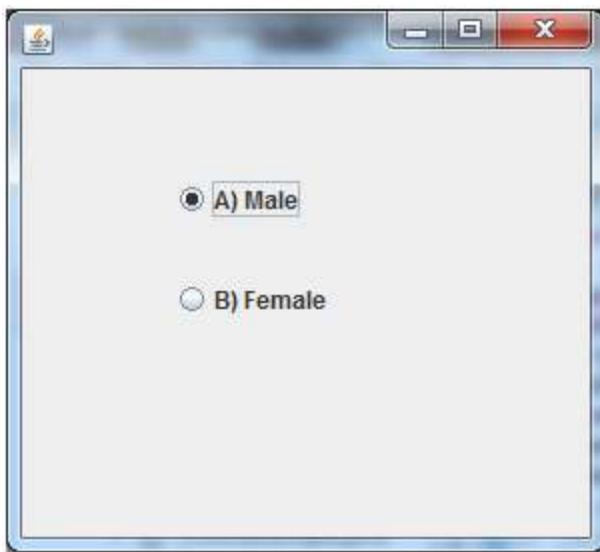
Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object

Java JRadioButton Example

```
1. import javax.swing.*;  
2. public class RadioButtonExample {  
3. JFrame f;  
4. RadioButtonExample(){  
5. f=new JFrame();  
6. JRadioButton r1=new JRadioButton("A) Male");  
7. JRadioButton r2=new JRadioButton("B) Female");  
8. r1.setBounds(75,50,100,30);  
9. r2.setBounds(75,100,100,30);  
10. ButtonGroup bg=new ButtonGroup();  
11. bg.add(r1);bg.add(r2);  
12. f.add(r1);f.add(r2);  
13. f.setSize(300,300);  
14. f.setLayout(null);  
15. f.setVisible(true);  
16. }  
17. public static void main(String[] args) {  
18. new RadioButtonExample();  
19. }  
20. }
```

Output:

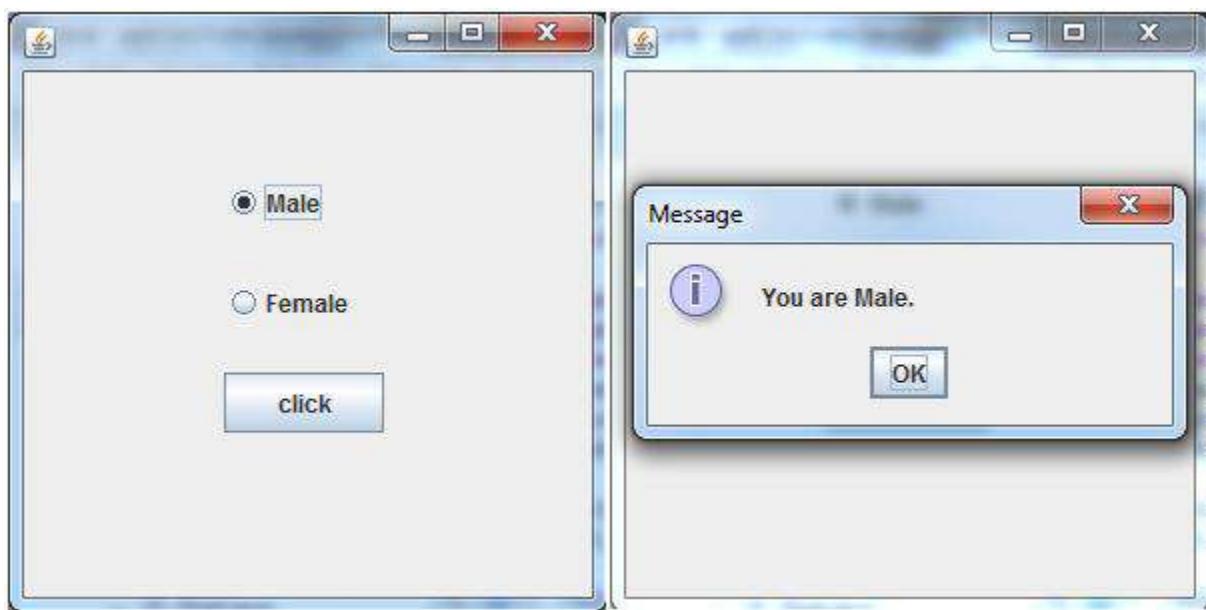


Java JRadioButton Example with ActionListener

1. `import javax.swing.*;`
2. `import java.awt.event.*;`
3. `class RadioButtonExample extends JFrame implements ActionListener{`
4. `JRadioButton rb1,rb2;`
5. `JButton b;`
6. `RadioButtonExample(){`
7. `rb1=new JRadioButton("Male");`
8. `rb1.setBounds(100,50,100,30);`
9. `rb2=new JRadioButton("Female");`
10. `rb2.setBounds(100,100,100,30);`
11. `ButtonGroup bg=new ButtonGroup();`
12. `bg.add(rb1);bg.add(rb2);`
13. `b=new JButton("click");`
14. `b.setBounds(100,150,80,30);`
15. `b.addActionListener(this);`
16. `add(rb1);add(rb2);add(b);`
17. `setSize(300,300);`
18. `setLayout(null);`
19. `setVisible(true);`
20. `}`

```
21. public void actionPerformed(ActionEvent e){  
22. if(rb1.isSelected()){  
23. JOptionPane.showMessageDialog(this,"You are Male.");  
24. }  
25. if(rb2.isSelected()){  
26. JOptionPane.showMessageDialog(this,"You are Female.");  
27. }  
28. }  
29. public static void main(String args[]){  
30. new RadioButtonExample();  
31. }}
```

Output:



Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a [menu](#). It inherits [JComponent](#) class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

1. **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array .
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector .

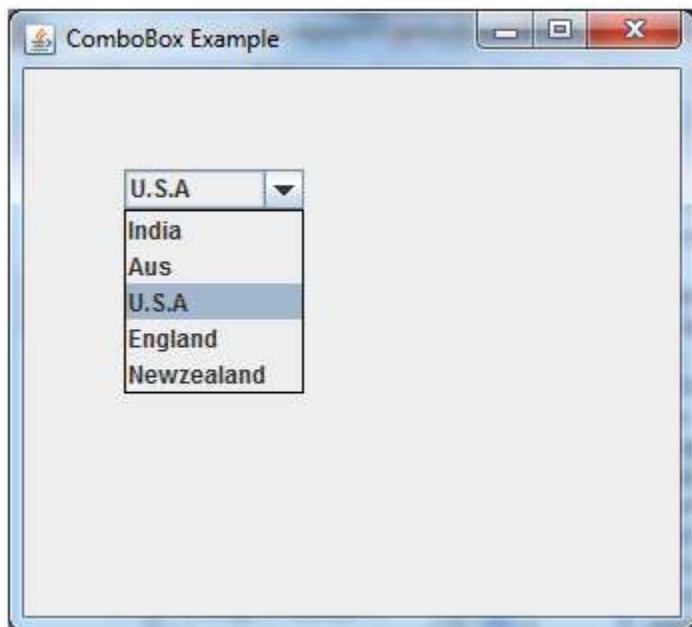
Commonly used Methods:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is edita
void addActionListener(ActionListener a)	It is used to add the ActionListener .
void addItemListener(ItemListener i)	It is used to add the ItemListener .

Java JComboBox Example

```
1. import javax.swing.*;
2. public class ComboBoxExample {
3. JFrame f;
4. ComboBoxExample(){
5.     f=new JFrame("ComboBox Example");
6.     String country[]={ "India", "Aus", "U.S.A", "England", "Newzealand"};
7.     JComboBox cb=new JComboBox(country);
8.     cb.setBounds(50, 50, 90, 20);
9.     f.add(cb);
10.    f.setLayout(null);
11.    f.setSize(400,500);
12.    f.setVisible(true);
13. }
14. public static void main(String[] args) {
15.     new ComboBoxExample();
16. }
17. }
```

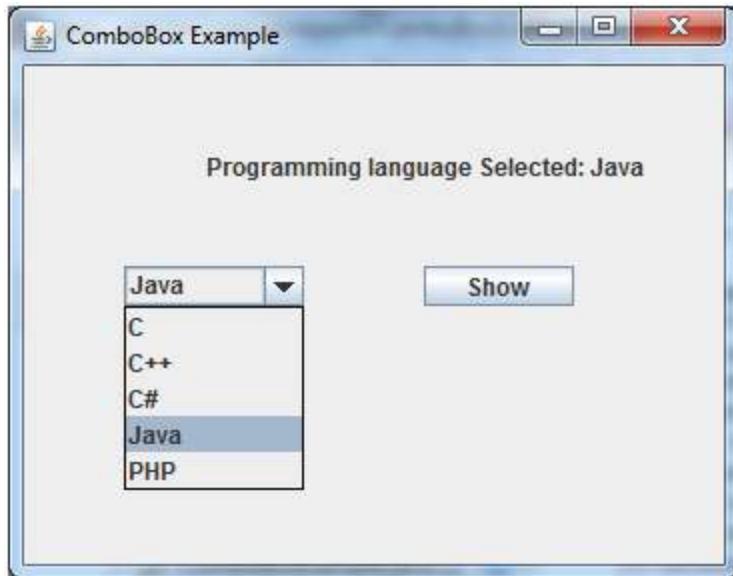
Output:



Java JComboBox Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class ComboBoxExample {
4.     JFrame f;
5.     ComboBoxExample(){
6.         f=new JFrame("ComboBox Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        JButton b=new JButton("Show");
11.        b.setBounds(200,100,75,20);
12.        String languages[]={ "C","C++","C#","Java","PHP"};
13.        final JComboBox cb=new JComboBox(languages);
14.        cb.setBounds(50, 100,90,20);
15.        f.add(cb); f.add(label); f.add(b);
16.        f.setLayout(null);
17.        f.setSize(350,350);
18.        f.setVisible(true);
19.        b.addActionListener(new ActionListener() {
20.            public void actionPerformed(ActionEvent e) {
21.                String data = "Programming language Selected: "
22.                + cb.getSelectedItem();
23.                label.setText(data);
24.            }
25.        });
26.    }
27.    public static void main(String[] args) {
28.        new ComboBoxExample();
29.    }
30. }
```

Output:



BorderLayout (LayoutManagers)

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. `java.awt.BorderLayout`
 2. `java.awtFlowLayout`
 3. `java.awt.GridLayout`
 4. `java.awt.CardLayout`
 5. `java.awt.GridBagLayout`
 6. `javax.swingBoxLayout`
 7. `javax.swing.GroupLayout`
 8. `javax.swing.ScrollPaneLayout`
 9. `javax.swing.SpringLayout` etc.
-

Java BorderLayout

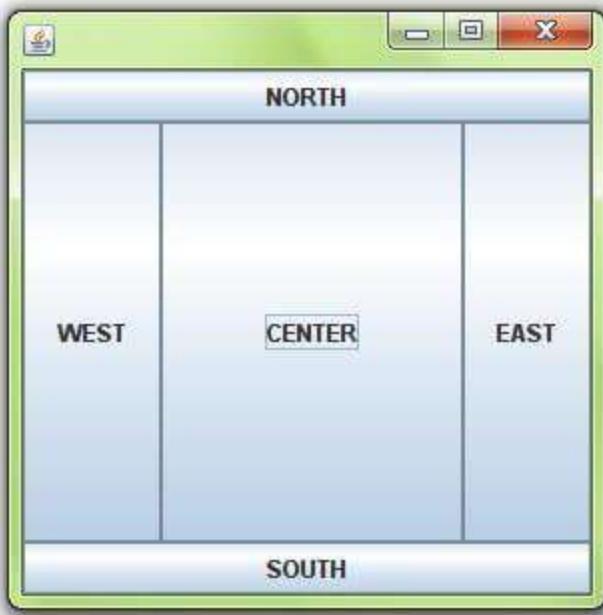
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
 - **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.
-

Example of BorderLayout class:



```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class Border {
5.     JFrame f;
6.     Border(){
7.         f=new JFrame();
8.
9.         JButton b1=new JButton("NORTH");
10.        JButton b2=new JButton("SOUTH");
11.        JButton b3=new JButton("EAST");
12.        JButton b4=new JButton("WEST");
13.        JButton b5=new JButton("CENTER");
14.
15.        f.add(b1,BorderLayout.NORTH);
16.        f.add(b2,BorderLayout.SOUTH);
17.        f.add(b3,BorderLayout.EAST);
18.        f.add(b4,BorderLayout.WEST);
19.        f.add(b5,BorderLayout.CENTER);
```

```
20.  
21. f.setSize(300,300);  
22. f.setVisible(true);  
23. }  
24. public static void main(String[] args) {  
25.     new Border();  
26. }  
27. }
```

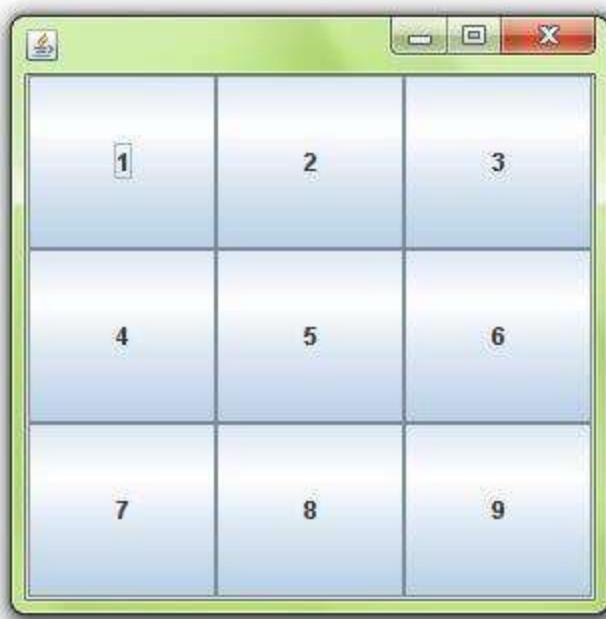
Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout()**: creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns)**: creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap)**: creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class



```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class MyGridLayout{
5.     JFrame f;
6.     MyGridLayout(){
7.         f=new JFrame();
8.
9.         JButton b1=new JButton("1");
10.        JButton b2=new JButton("2");
11.        JButton b3=new JButton("3");
12.        JButton b4=new JButton("4");
13.        JButton b5=new JButton("5");
14.        JButton b6=new JButton("6");
15.        JButton b7=new JButton("7");
16.        JButton b8=new JButton("8");
17.        JButton b9=new JButton("9");
18.
19.         f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
```

```

20. f.add(b6);f.add(b7);f.add(b8);f.add(b9);
21.
22. f.setLayout(new GridLayout(3,3));
23. //setting grid layout of 3 rows and 3 columns
24.
25. f.setSize(300,300);
26. f.setVisible(true);
27. }
28. public static void main(String[] args) {
29.   new MyGridLayout();
30. }
31. }
```

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class



```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class MyFlowLayout{
5.     JFrame f;
6.     MyFlowLayout(){
7.         f=new JFrame();
8.
9.         JButton b1=new JButton("1");
10.        JButton b2=new JButton("2");
11.        JButton b3=new JButton("3");
12.        JButton b4=new JButton("4");
13.        JButton b5=new JButton("5");
14.
15.        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
16.
```

```
17. f.setLayout(new FlowLayout(FlowLayout.RIGHT));  
18. //setting flow layout of right alignment  
19.  
20. f.setSize(300,300);  
21. f.setVisible(true);  
22. }  
23. public static void main(String[] args) {  
24.   new MyFlowLayout();  
25. }  
26. }
```

Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in javax.swing package.

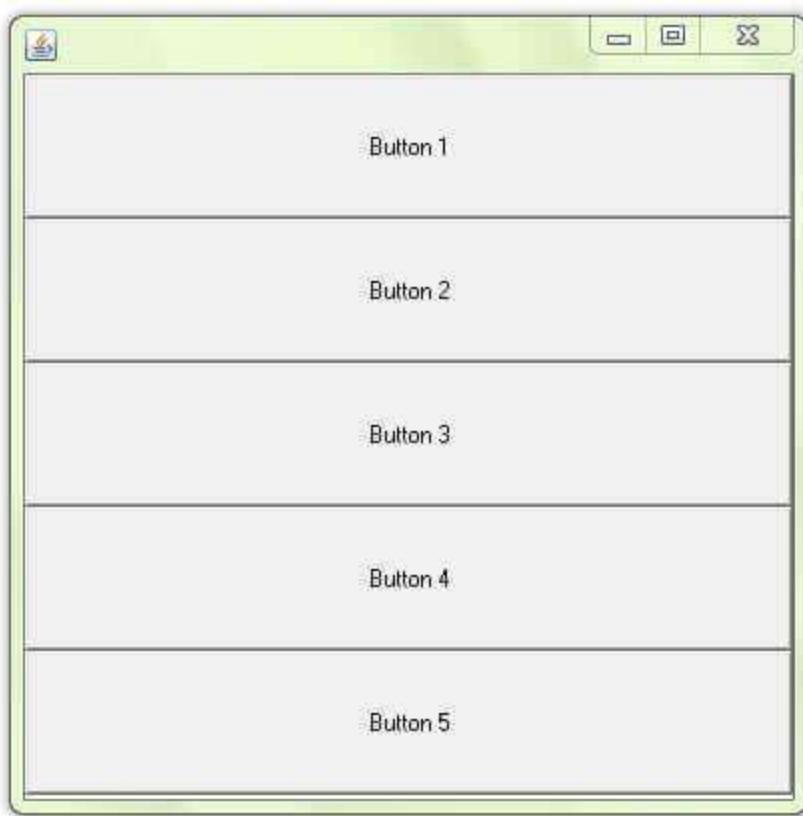
Fields of BoxLayout class

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Constructor of BoxLayout class

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BoxLayout class with Y-AXIS:

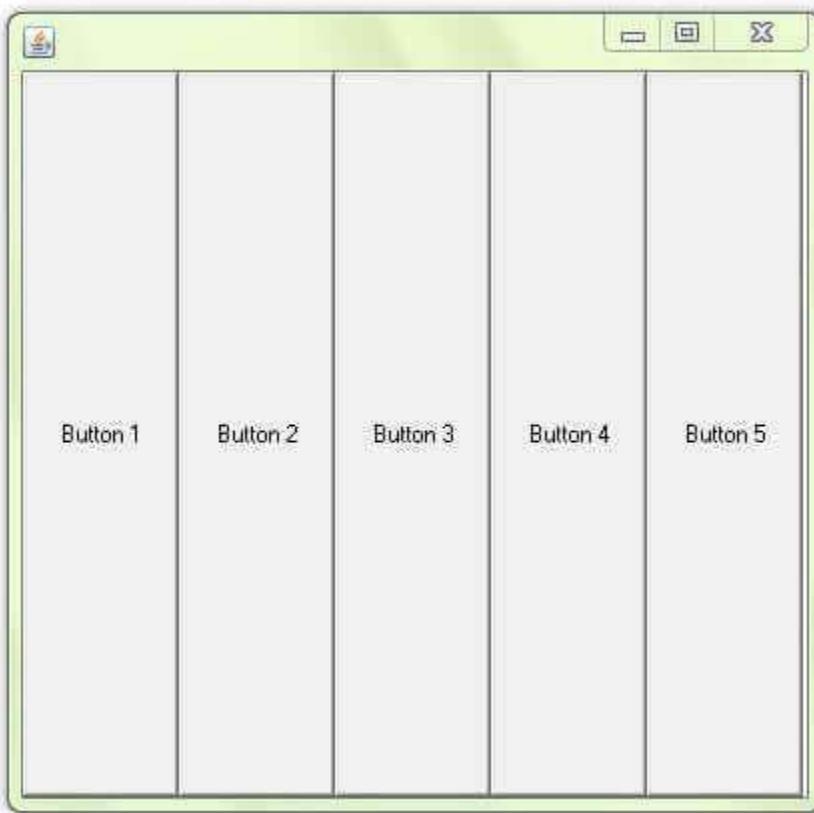


```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BoxLayoutExample1 extends Frame {
5.     Button buttons[];
6.
7.     public BoxLayoutExample1 () {
8.         buttons = new Button [5];
9.
10.    for (int i = 0;i<5;i++) {
11.        buttons[i] = new Button ("Button " + (i + 1));
12.        add (buttons[i]);
13.    }
14.
15.    setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
16.    setSize(400,400);
```

```
17. setVisible(true);  
18. }  
19.  
20. public static void main(String args[]){  
21. BoxLayoutExample1 b=new BoxLayoutExample1();  
22. }  
23. }
```

[download this example](#)

Example of BoxLayout class with X-AXIS



```
1. import java.awt.*;  
2. import javax.swing.*;  
3.  
4. public class BoxLayoutExample2 extends Frame {  
5.     Button buttons[];
```

```

6.
7. public BoxLayoutExample2() {
8.     buttons = new Button [5];
9.
10.    for (int i = 0;i<5;i++) {
11.        buttons[i] = new Button ("Button " + (i + 1));
12.        add (buttons[i]);
13.    }
14.
15. setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
16. setSize(400,400);
17. setVisible(true);
18. }
19.
20. public static void main(String args[]){
21.     BoxLayoutExample2 b=new BoxLayoutExample2();
22. }
23. }
```

Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

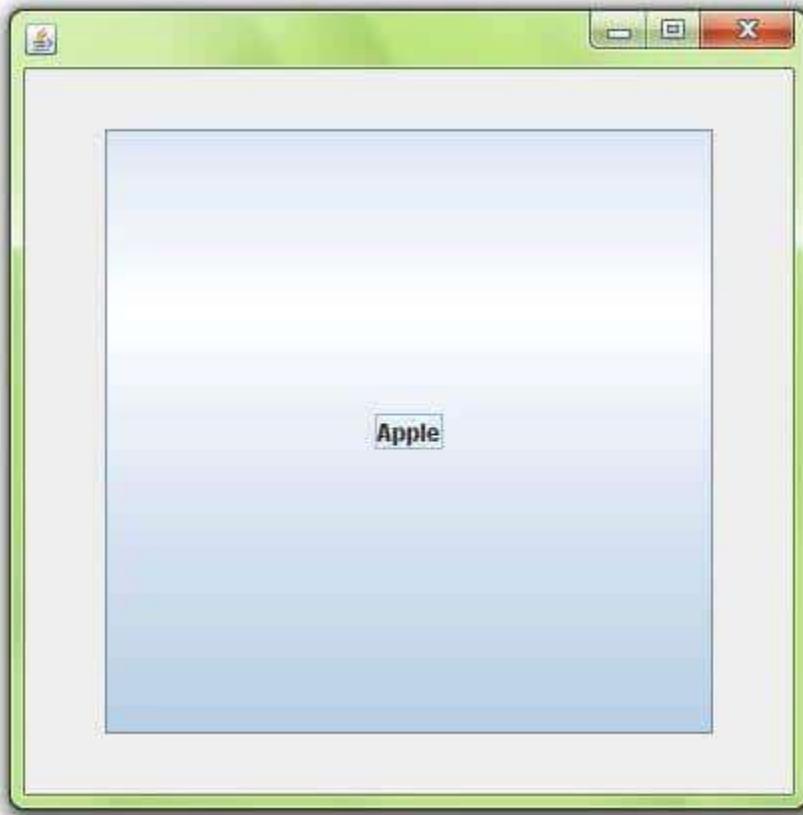
1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- o **public void next(Container parent):** is used to flip to the next card of the given container.

- **public void previous(Container parent):** is used to flip to the previous card of the given container.
 - **public void first(Container parent):** is used to flip to the first card of the given container.
 - **public void last(Container parent):** is used to flip to the last card of the given container.
 - **public void show(Container parent, String name):** is used to flip to the specified card with the given name.
-

Example of CardLayout class



1. **import** java.awt.*;
2. **import** java.awt.event.*;
- 3.
4. **import** javax.swing.*;
- 5.

```
6. public class CardLayoutExample extends JFrame implements ActionListener{  
7.     CardLayout card;  
8.     JButton b1,b2,b3;  
9.     Container c;  
10.    CardLayoutExample(){  
11.  
12.        c=getContentPane();  
13.        card=new CardLayout(40,30);  
14. //create CardLayout object with 40 hor space and 30 ver space  
15.        c.setLayout(card);  
16.  
17.        b1=new JButton("Apple");  
18.        b2=new JButton("Boy");  
19.        b3=new JButton("Cat");  
20.        b1.addActionListener(this);  
21.        b2.addActionListener(this);  
22.        b3.addActionListener(this);  
23.  
24.        c.add("a",b1);c.add("b",b2);c.add("c",b3);  
25.  
26.    }  
27.    public void actionPerformed(ActionEvent e) {  
28.        card.next(c);  
29.    }  
30.  
31.    public static void main(String[] args) {  
32.        CardLayoutExample cl=new CardLayoutExample();  
33.        cl.setSize(400,400);  
34.        cl.setVisible(true);  
35.        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);  
36.    }  
37. }
```

Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column widths.
protected Hashtable<Component,GridBagConstraints>	comptable	It is used to maintain the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraint with the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility.
protected static int	MINSIZE	It is smallest grid that can be laid out.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out.

int[]	rowHeights	It is used to hold the overrides to the row height.
double[]	rowWeights	It is used to hold the overrides to the row weight.

Useful Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to specified constraints object.
void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not support per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height values depending on the constraint.
protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility.
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and should not be used for compatibility.
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for a component.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x-axis.
float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y-axis.

int[][]	getLayoutDimensions()	It determines column widths and layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and subject to change.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and subject to change.
Point	getLayoutOrigin()	It determines the origin of the layout coordinate space of the target component.
double[][]	getLayoutWeights()	It determines the weights of the layout rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the components by getting the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and subject to change.

Example

```

1. import java.awt.Button;
2. import java.awt.GridBagConstraints;
3. import java.awt.GridBagLayout;
4.
5. import javax.swing.*;
6. public class GridBagLayoutExample extends JFrame{
7.     public static void main(String[] args) {
8.         GridBagLayoutExample a = new GridBagLayoutExample();
9.     }
10.    public GridBagLayoutExample() {

```

```
11.  GridBagLayoutgrid = new GridBagLayout();
12.  GridBagConstraints gbc = new GridBagConstraints();
13.  setLayout(grid);
14.  setTitle("GridBag Layout Example");
15.  GridBagLayout layout = new GridBagLayout();
16.  this.setLayout(layout);
17.  gbc.fill = GridBagConstraints.HORIZONTAL;
18.  gbc.gridx = 0;
19.  gbc.gridy = 0;
20.  this.add(new Button("Button One"), gbc);
21.  gbc.gridx = 1;
22.  gbc.gridy = 0;
23.  this.add(new Button("Button two"), gbc);
24.  gbc.fill = GridBagConstraints.HORIZONTAL;
25.  gbc.ipady = 20;
26.  gbc.gridx = 0;
27.  gbc.gridy = 1;
28.  this.add(new Button("Button Three"), gbc);
29.  gbc.gridx = 1;
30.  gbc.gridy = 1;
31.  this.add(new Button("Button Four"), gbc);
32.  gbc.gridx = 0;
33.  gbc.gridy = 2;
34.  gbc.fill = GridBagConstraints.HORIZONTAL;
35.  gbc.gridwidth = 2;
36.  this.add(new Button("Button Five"), gbc);
37.  setSize(300, 300);
38.  setPreferredSize(getSize());
39.  setVisible(true);
40.  setDefaultCloseOperation(EXIT_ON_CLOSE);
41.
42.  }
43.
44. }
```

Output:



Example 2

```
1. public class GridBagLayoutDemo {  
2.     final static boolean shouldFill = true;  
3.     final static boolean shouldWeightX = true;  
4.     final static boolean RIGHT_TO_LEFT = false;  
5.  
6.     public static void addComponentsToPane(Container pane) {  
7.         if (RIGHT_TO_LEFT) {  
8.             pane.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);  
9.         }  
10.  
11.        JButton button;  
12.        pane.setLayout(new GridBagLayout());  
13.        GridBagConstraints c = new GridBagConstraints();  
14.        if (shouldFill) {  
15.            //natural height, maximum width  
16.            c.fill = GridBagConstraints.HORIZONTAL;  
17.        }
```

```
18.  
19. button = new JButton("Button 1");  
20. if (shouldWeightX) {  
21.     c.weightx = 0.5;  
22. }  
23. c.fill = GridBagConstraints.HORIZONTAL;  
24. c.gridx = 0;  
25. c.gridy = 0;  
26. pane.add(button, c);  
27.  
28. button = new JButton("Button 2");  
29. c.fill = GridBagConstraints.HORIZONTAL;  
30. c.weightx = 0.5;  
31. c.gridx = 1;  
32. c.gridy = 0;  
33. pane.add(button, c);  
34.  
35. button = new JButton("Button 3");  
36. c.fill = GridBagConstraints.HORIZONTAL;  
37. c.weightx = 0.5;  
38. c.gridx = 2;  
39. c.gridy = 0;  
40. pane.add(button, c);  
41.  
42. button = new JButton("Long-Named Button 4");  
43. c.fill = GridBagConstraints.HORIZONTAL;  
44. c.ipady = 40; //make this component tall  
45. c.weightx = 0.0;  
46. c.gridwidth = 3;  
47. c.gridx = 0;  
48. c.gridy = 1;  
49. pane.add(button, c);  
50.  
51. button = new JButton("5");
```

```
52. c.fill = GridBagConstraints.HORIZONTAL;
53. c.ipady = 0;      //reset to default
54. c.weighty = 1.0;  //request any extra vertical space
55. c.anchor = GridBagConstraints.PAGE_END; //bottom of space
56. c.insets = new Insets(10,0,0,0); //top padding
57. c.gridx = 1;      //aligned with button 2
58. c.gridwidth = 2;  //2 columns wide
59. c.gridy = 2;      //third row
60. pane.add(button, c);
61. }
62.
63.
64. private static void createAndShowGUI() {
65. //Create and set up the window.
66. JFrame frame = new JFrame("GridBagLayoutDemo");
67. frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68.
69. //Set up the content pane.
70. addComponentsToPane(frame.getContentPane());
71.
72. //Display the window.
73. frame.pack();
74. frame.setVisible(true);
75. }
76.
77. public static void main(String[] args) {
78. javax.swing.SwingUtilities.invokeLater(new Runnable() {
79. public void run() {
80. createAndShowGUI();
81. }
82. });
83. }
84. }
```

