UNIT – III (LAST)

Biswajit Senapati,
Faculty,
Department of CSE,
NIT AP,
Tadepalligudem, AP.

Need Of Collections

An array is an indexed Collection of fixed number of homogeneous data elements.
The Main advantage of Arrays is we can represent multiple values with a single variable.
So that reusability of the code will be improved.

Limitations of Object type Arrays:
Arrays are fixed in size i.e. once we created an array with some size there is no chance of increasing or decreasing it's size based on our requirement. Hence to use arrays compulsory we should know the size in advance which may not possible always.

2) Arrays can hold only homogeneous data elements.
Ex:-
```
Student [] s = new Student [10000];
s[0] = new Student; (correct)
s[1] = new Customer(); (wrong)
```

But We can resolve this problem by using object Arrays.
```
Object [] o = new Object [10000];
o[0] = new Student();
o[1] = new Customer();
```

Arrays Concept is not implemented based on some standard data structure hence readymade method support is not available for every requirement we have to write the code explicitly. Which is complexity of programming.

To overcome the above limitations of Arrays we should go for Collections.

Collections are growbable in nature. i.e. Based on our requirement we can increase (or) Decrease the size.
Collections can hold both homogeneous & Heterogeneous elements.
Every Collection class is implemented based on some standard data structure. Hence readymade method support is available for every requirement. Being a programmer we have to use this method and we are not responsible to provide implementation.

Difference between Arrays and Collections:

Arrays	Collections
1. Arrays are fixed in size.	1. Collections are growable in nature. I.e. based on our requirement we can increase or decrease the size.
2. Wrt memory arrays are not recommended to use.	2. Wrt to memory collections are recommended to use.
3. Wrt Performance Arrays are recommended to use.	3. Wrt Performance collections are not recommended to use.
4. Array can hold only homogeneous datatype elements	4. Collections can hold both homogeneous and heterogeneous elements.
5. There is no underlying data structure for arrays and hence readymade method support is not available	5. Every Collections class is implemented based on some standard data structure. Hence readymade method support is available for every requirement.
6. Array can hold both primitives and object types	6. Collections can hold only objects but not primitives.

What is Collection?

If we want to represent a group of individual objects as a single entity then we should go for Collection.

What is Collection Framework?

It defines several classes and interfaces which can be used a group of objects as single entity.

* Collection is an interface which can be used to represent a group of individual objects as a single entity.

* Collections is an utility class present in java.util.package to define several utility methods (like Sorting, Searching..) for Collection objects.

List

* Duplicates are allowed
* Insertion order preserved

Set

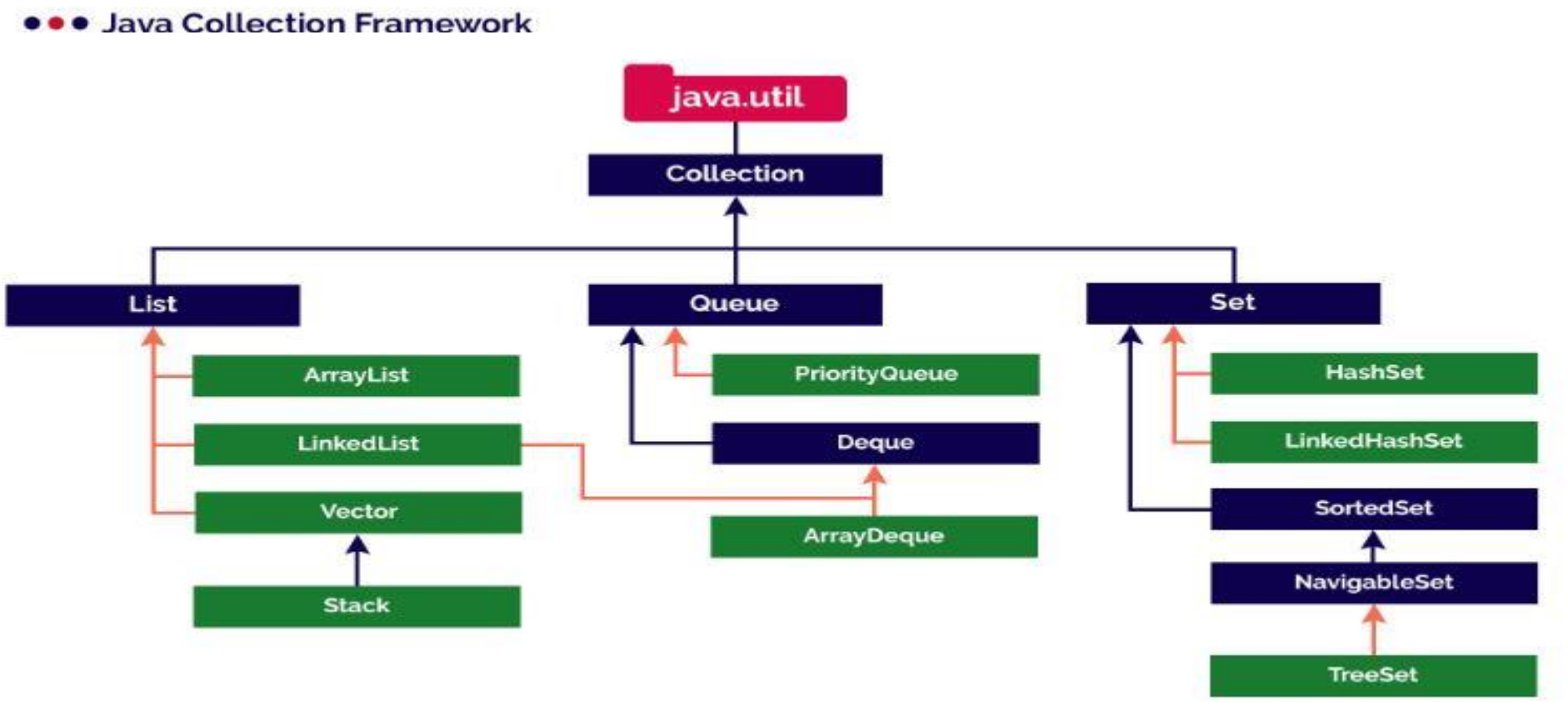
* Duplicates are not allowed
* Insertion order not preserved

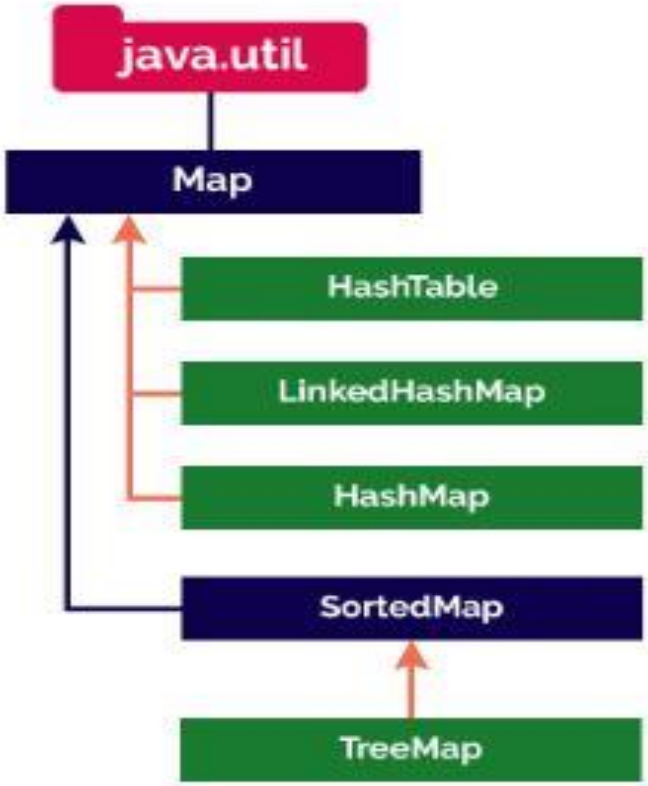
The Collections framework

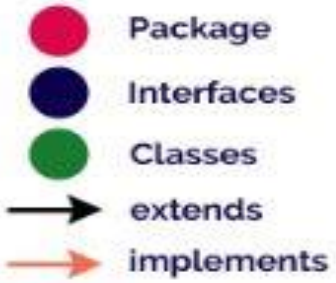
Java collection framework is a collection of interfaces and classes used to storing and processing a group of individual objects as a single unit. The java collection framework holds several classes that provide a large number of methods to store and process a group of objects. These classes make the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

Java collection framework was introduced in java 1.2 version.

Java collection framework has the following heirarchy.







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

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

LIST (The List is an indexed sequence)

Array List Class

The `ArrayList` class is a part of java collection framework. It is available inside the `java.util` package. The ArrayList class extends AbstractList class and implements List interface.

The elements of ArrayList are organized as an array internally. The default size of an ArrayList is 10.

The ArrayList class is used to create a dynamic array that can grow or shrunk as needed.

- The ArrayList is a child class of `AbstractList`
- The ArrayList implements interfaces like `List`, `Serializable`, `Cloneable`, and `RandomAccess`.
- The ArrayList allows to store duplicate data values.
- The ArrayList allows to access elements randomly using index-based accessing.
- The ArrayList maintains the order of insertion.

ArrayList class declaration

The ArrayList class has the following declaration.

Example

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable
```

ArrayList class constructors

The ArrayList class has the following constructors.

* `ArrayList( )` - Creates an empty ArrayList.
* `ArrayList(Collection c)` - Creates an ArrayList with given collection of elements.
* `ArrayList(int size)` - Creates an empty ArrayList with given size (capacity).

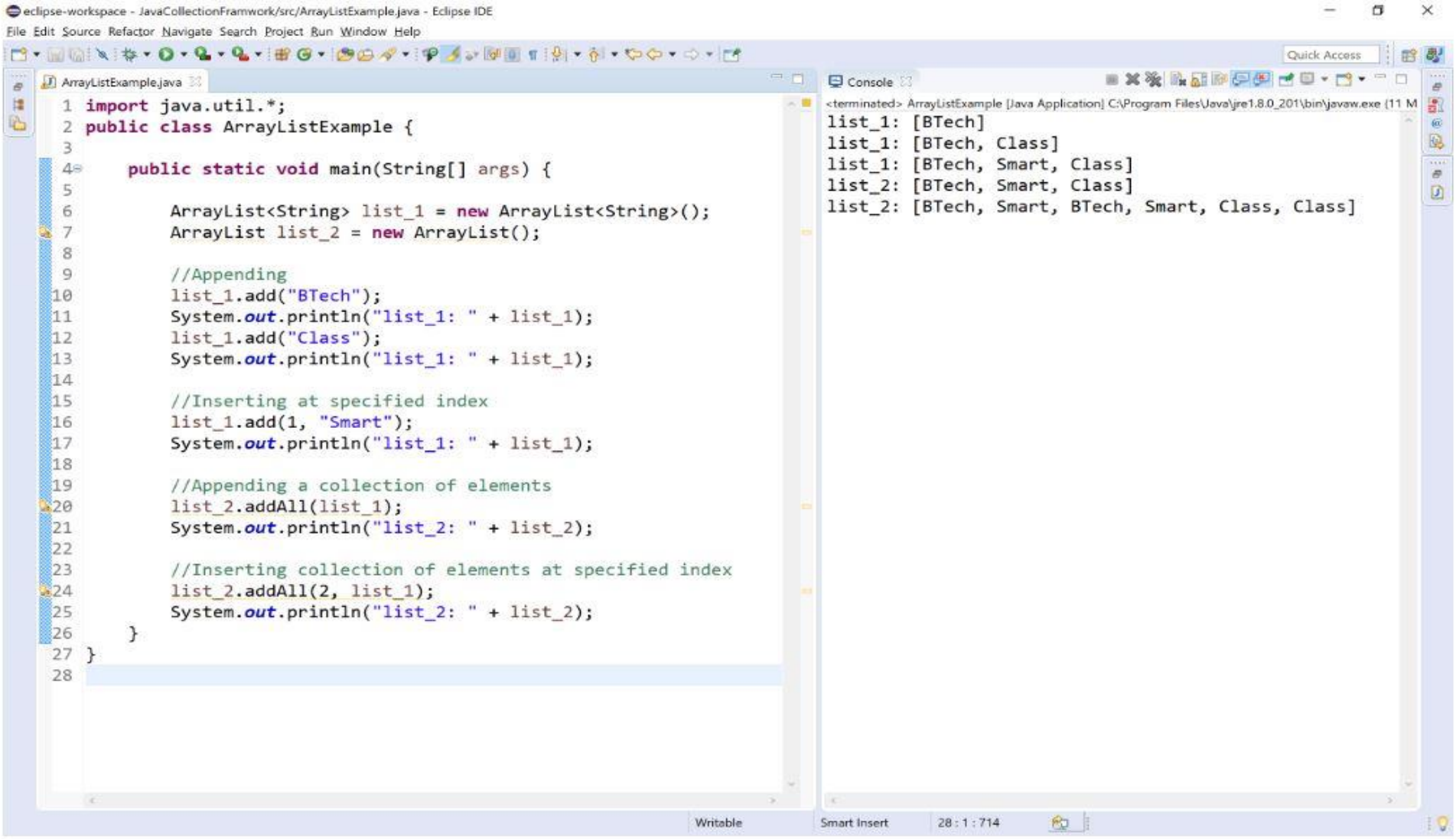
Operations on ArrayList

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

Adding Items

The ArrayList class has the following methods to add items.

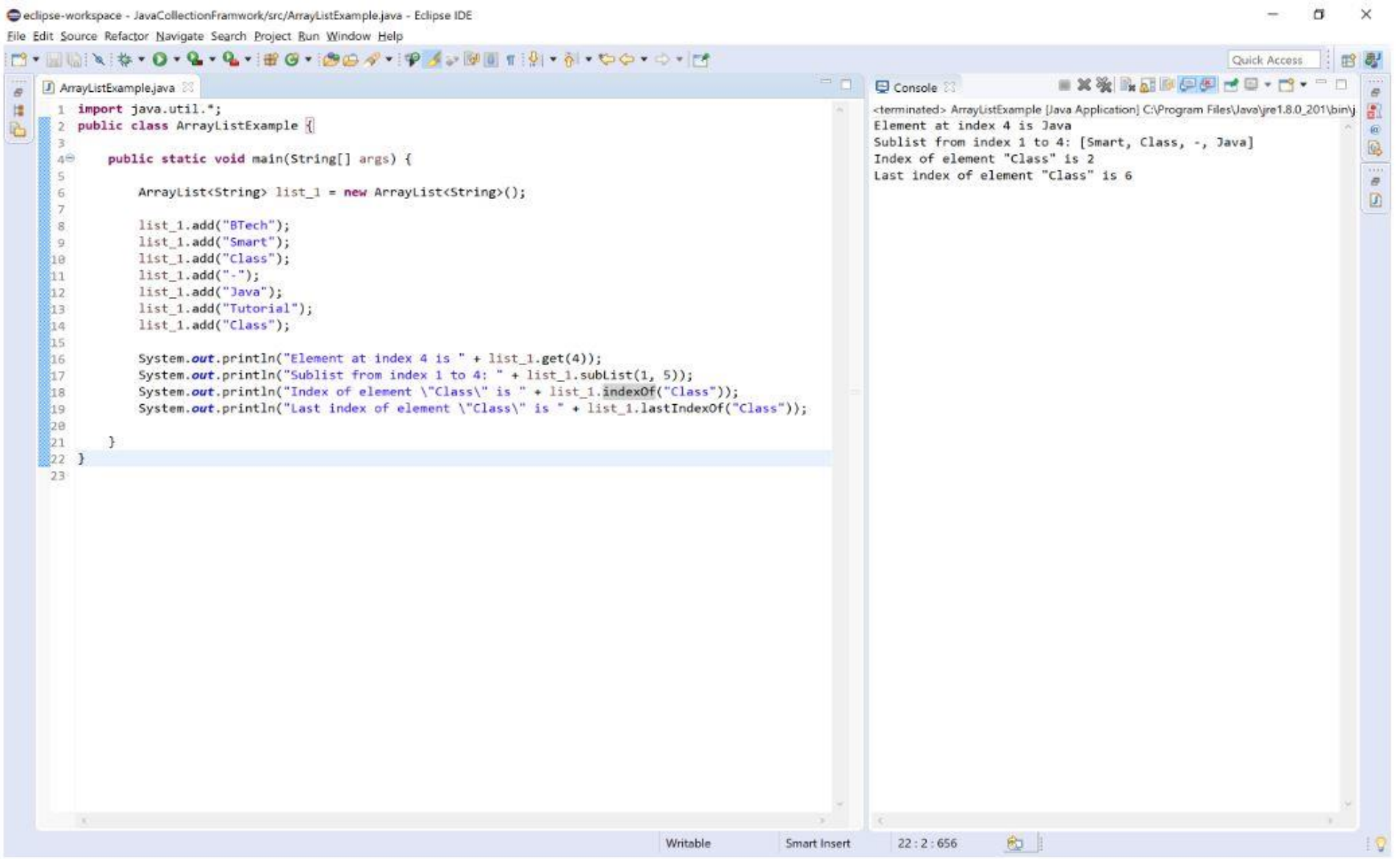
* `boolean add(E element)` - Appends given element to the ArrayList.
* `boolean addAll(Collection c)` - Appends given collection of elements to the ArrayList.
* `void add(int index, E element)` - Inserts the given element at specified index.
* `boolean addAll(int index, Collection c)` - Inserts the given collection of elements at specified index.



Accessing Items

The ArrayList class has the following methods to access items.

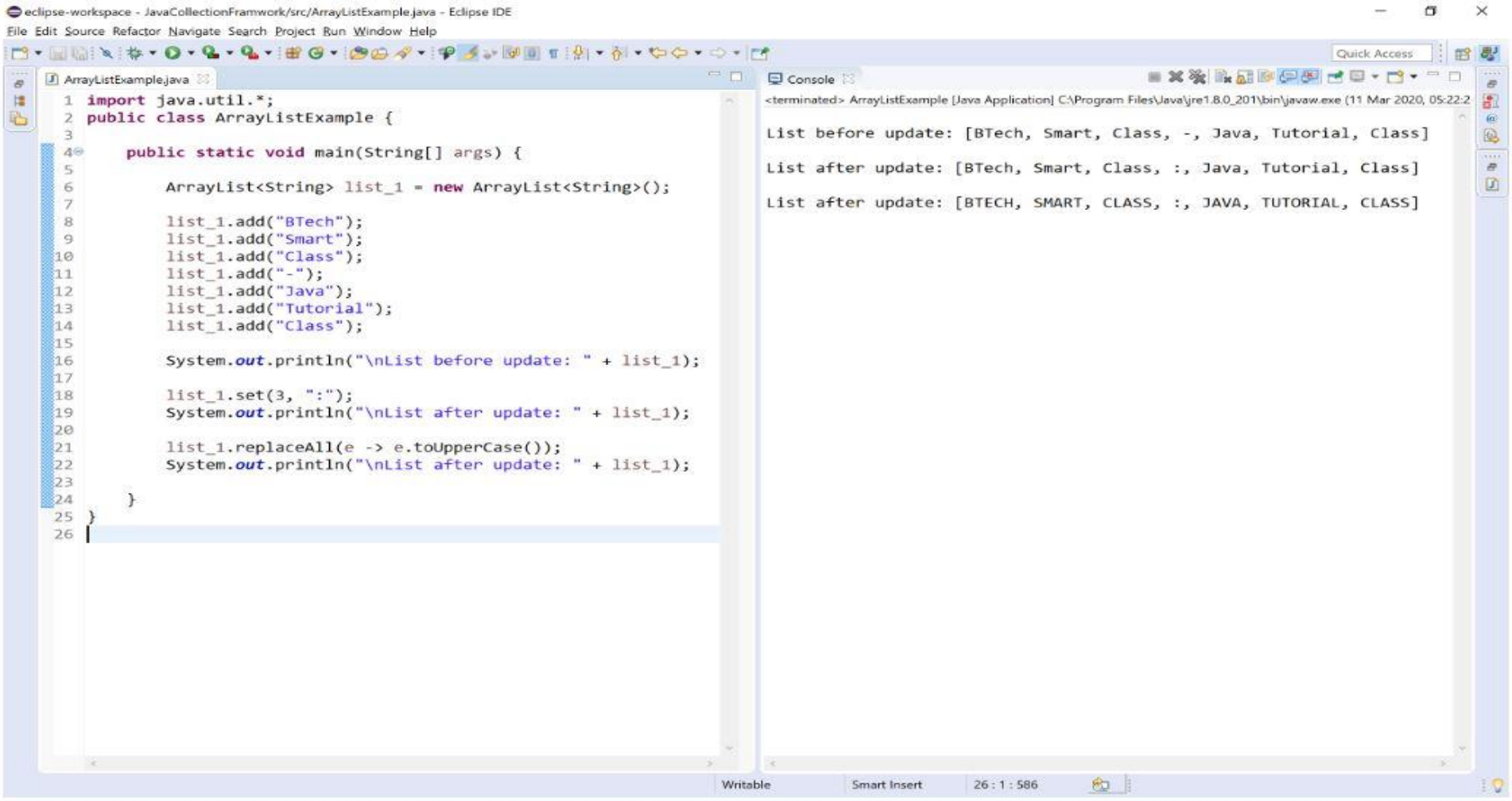
- `E get(int index)` - Returns element at specified index from the ArrayList.
- `ArrayList subList(int startIndex, int lastIndex)` - Returns an ArrayList that contails elements from specified startIndex to lastIndex-1 from the invoking ArrayList.
- `int indexOf(E element)` - Returns the index value of given element first occurence in the ArrayList.
- `int lastIndexOf(E element)` - Returns the index value of given element last occurence in the ArrayList.



Updating Items

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

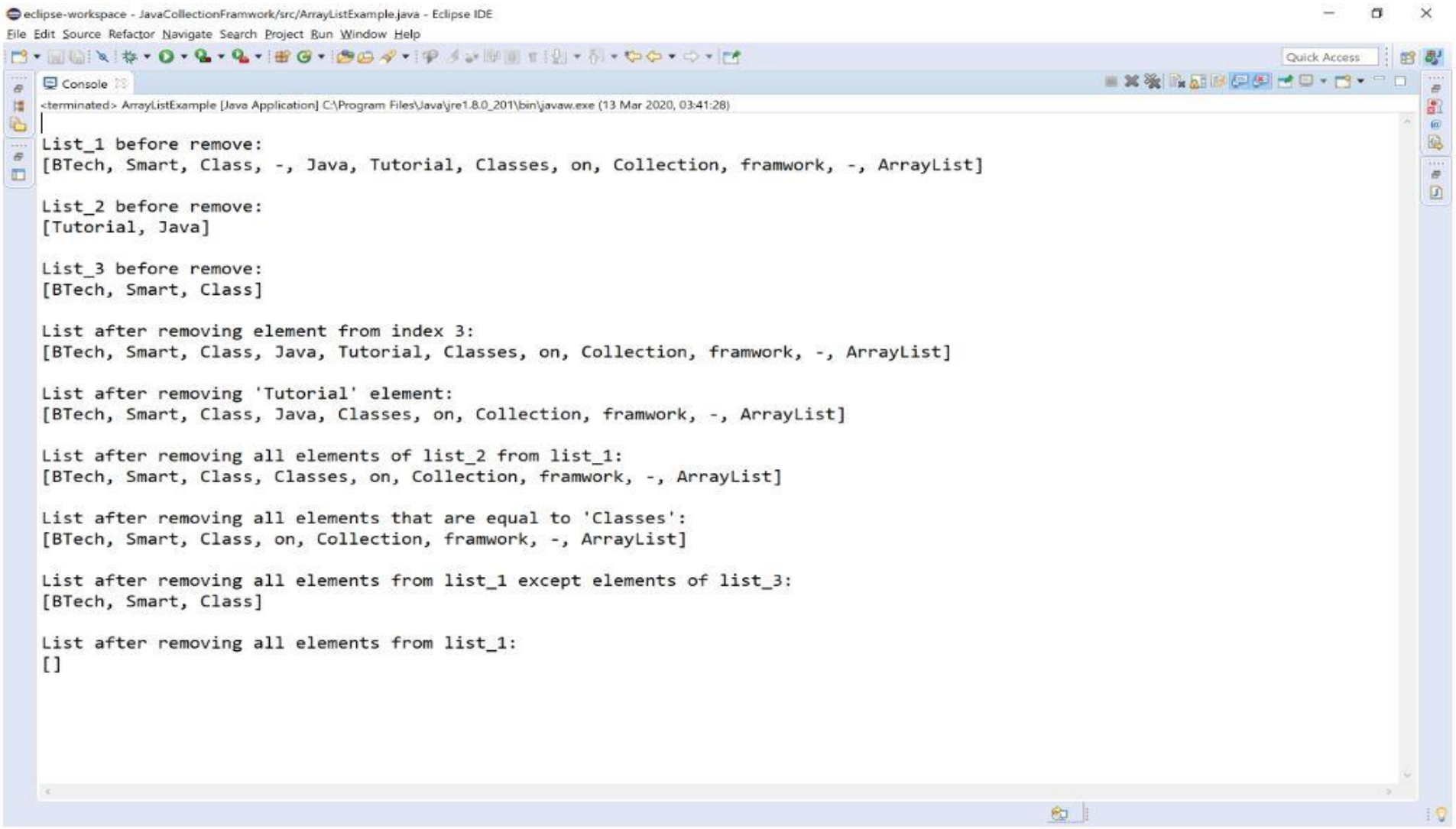
* `E set(int index, E newElement)` - Replace the element at specified index with newElement in the invoking ArrayList.
* `ArrayList replaceAll(UnaryOperator e)` - Replaces each element of invoking ArrayList with the result of applying the operator to that element.



Removing Items

The ArrayList class has the following methods to remove items.

- `E remove(int index)` - Removes the element at specified index in the invoking ArrayList.
- `boolean remove(Object element)` - Removes the first occurence of the given element from the invoking ArrayList.
- `boolean removeAll(Collection c)` - Removes the given collection of elements from the invoking ArrayList.
- `void retainAll(Collection c)` - Removes all the elements except the given collection of elements from the invoking ArrayList.
- `boolean removeIf(Predicate filter)` - Removes all the elements from the ArrayList that satisfies the given predicate.
- `void clear( )` - Removes all the elements from the ArrayList.


eclipse-workspace - JavaCollectionFramwork/src/ArrayListExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Console
<terminated> ArrayListExample [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (13 Mar 2020, 03:41:28)
List_1 before remove:
[BTech, Smart, Class, -, Java, Tutorial, Classes, on, Collection, framwork, -, ArrayList]
List_2 before remove:
[Tutorial, Java]
List_3 before remove:
[BTech, Smart, Class]
List after removing element from index 3:
[BTech, Smart, Class, Java, Tutorial, Classes, on, Collection, framwork, -, ArrayList]
List after removing 'Tutorial' element:
[BTech, Smart, Class, Java, Classes, on, Collection, framwork, -, ArrayList]
List after removing all elements of list_2 from list_1:
[BTech, Smart, Class, Classes, on, Collection, framwork, -, ArrayList]
List after removing all elements that are equal to 'Classes':
[BTech, Smart, Class, on, Collection, framwork, -, ArrayList]
List after removing all elements from list_1 except elements of list_3:
[BTech, Smart, Class]
List after removing all elements from list_1:
[]

Linked List Class

The `LinkedList` class is a part of java collection framework. It is available inside the `java.util` package. The LinkedList class extends `AbstractSequentialList` class and implements `List` and `Deque` interface.

The elements of LinkedList are organized as the elements of linked list data structure.

The LinkedList class is used to create a dynamic list of elements that can grow or shrunk as needed.

- The LinkedList is a child class of `AbstractSequentialList`
- The LinkedList implements interfaces like `List`, `Deque`, `Cloneable`, and `Serializable`.
- The LinkedList allows to store duplicate data values.
- The LinkedList maintains the order of insertion.

LinkedList class declaration

The LinkedList class has the following declaration.

Example

```
public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable
```

LinkedList class constructors

The LinkedList class has the following constructors.

* `LinkedList()` - Creates an empty List.
* `LinkedList(Collection c)` - Creates a List with given collection of elements.

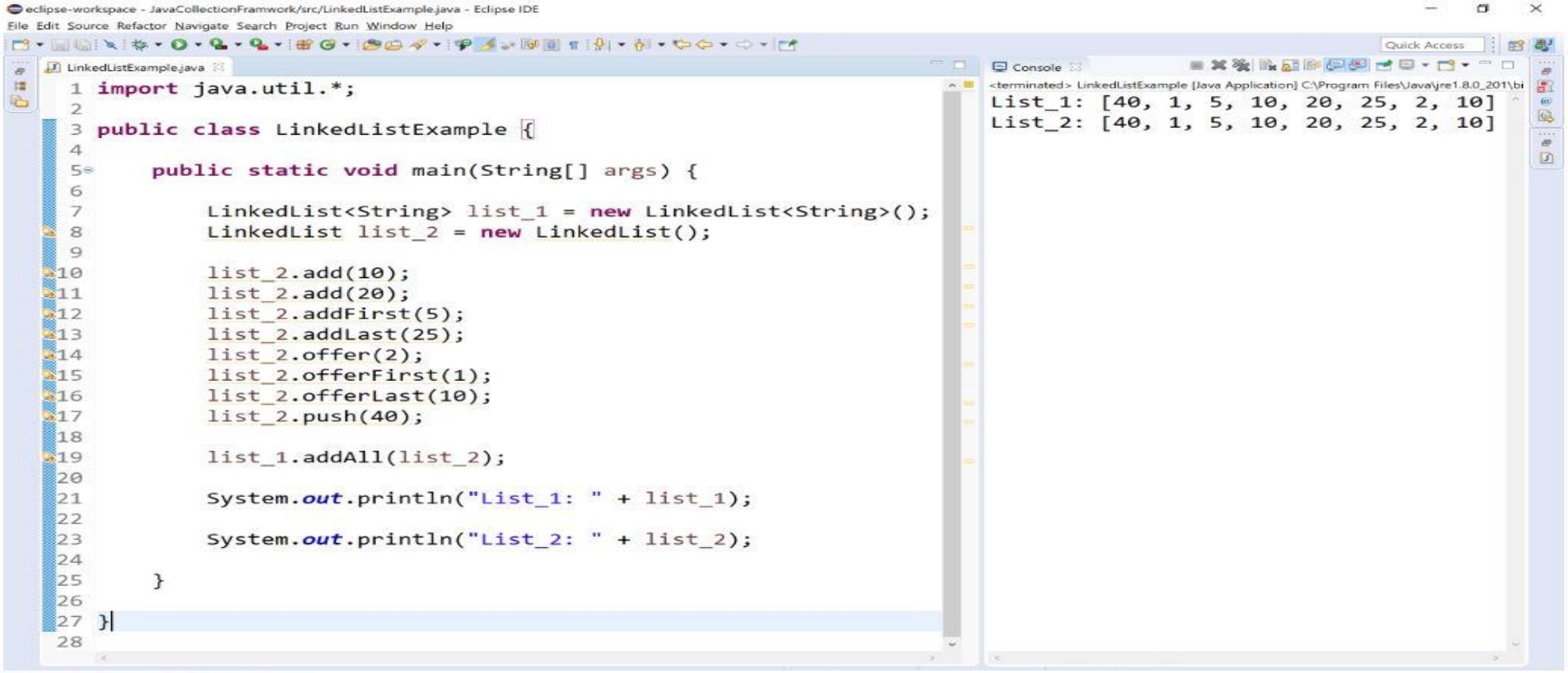
Operations on LinkedList

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

Adding Items

The LinkedList class has the following methods to add items.

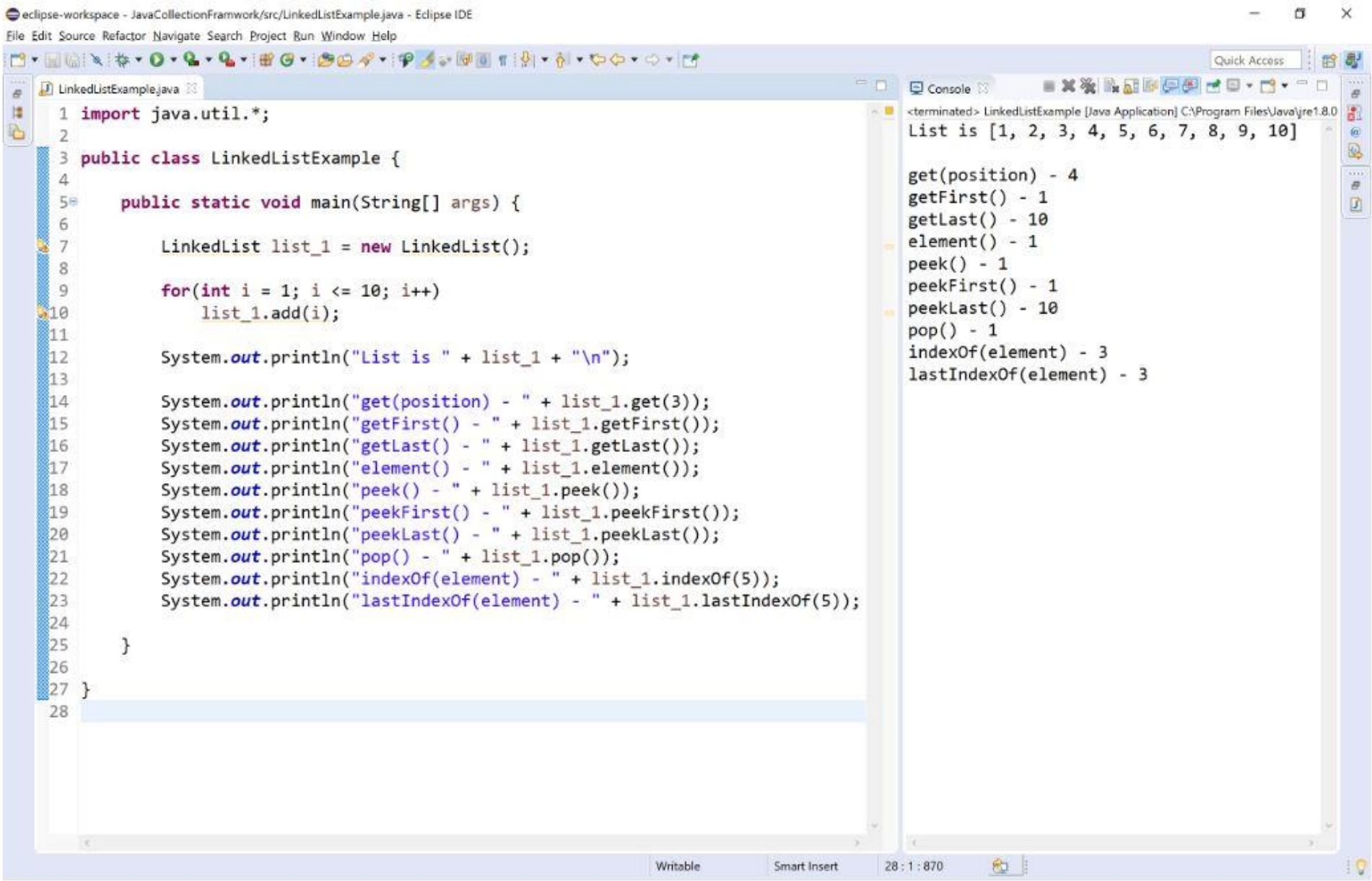
* `boolean add(E element)` - Appends given element to the List.
* `boolean addAll(Collection c)` - Appends given collection of elements to the List.
* `void add(int position, E element)` - Inserts the given element at specified position.
* `boolean addAll(int position, Collection c)` - Inserts the given collection of elements at specified position.
* `void addFirst(E element)` - Inserts the given element at beggining of the list.
* `void addLast(E element)` - Inserts the given element at end of the list.
* `boolean offer(E element)` - Inserts the given element at end of the list.
* `boolean offerFirst(E element)` - Inserts the given element at beggining of the list.
* `boolean offerLast(E element)` - Inserts the given element at end of the list.
* `void push(E element)` - Inserts the given element at beggining of the list.



Accessing Items

The LinkedList class has the following methods to access items.

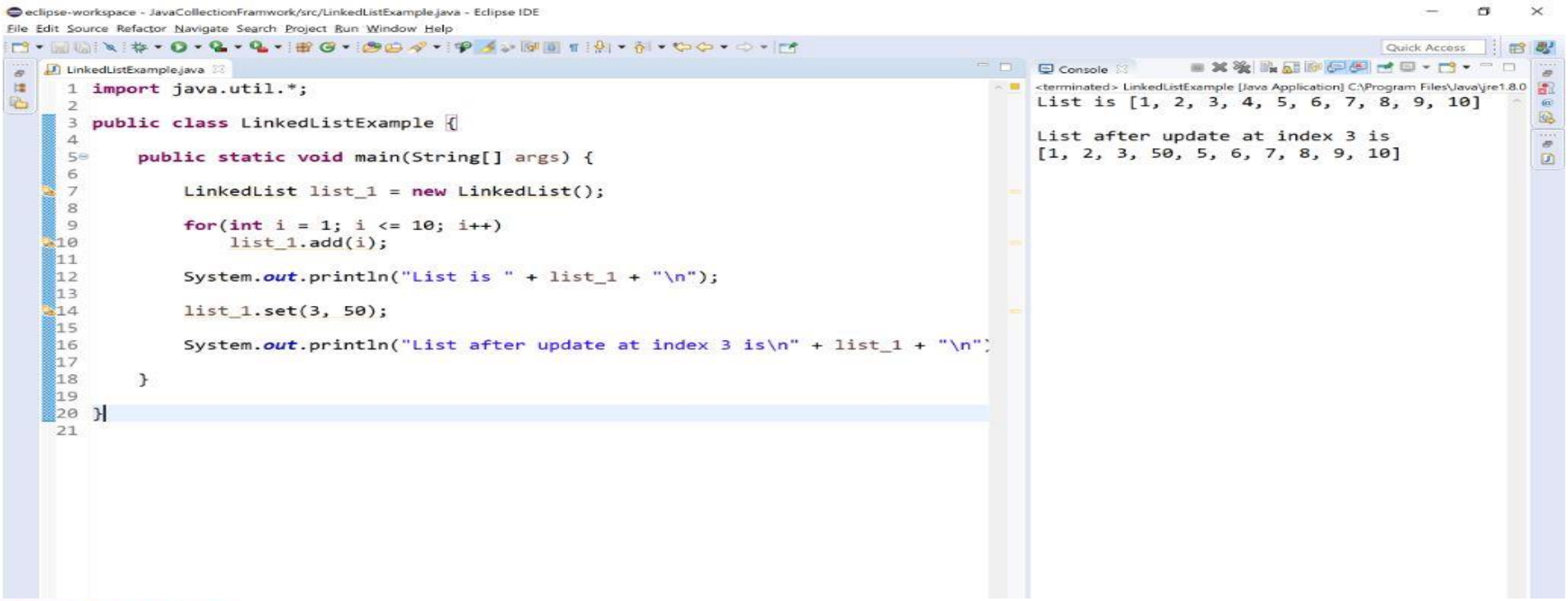
- `E get(int position)` - Returns element at specified position from the LinkedList.
- `E element()` - Returns the first element from the invoking LinkedList.
- `E getFirst()` - Returns the first element from the invoking LinkedList.
- `E getLast()` - Returns the last element from the invoking LinkedList.
- `E peek()` - Returns the first element from the invoking LinkedList.
- `E peekFirst()` - Returns the first element from the invoking LinkedList, and returns null if list is empty.
- `E peekLast()` - Returns the last element from the invoking LinkedList, and returns null if list is empty.
- `int indexOf(E element)` - Returns the index value of given element first occurence in the LinkedList.
- `int lastIndexOf(E element)` - Returns the index value of given element last occurence in the LinkedList.
- `E pop()` - Returns the first element from the invoking LinkedList.



Updating Items

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

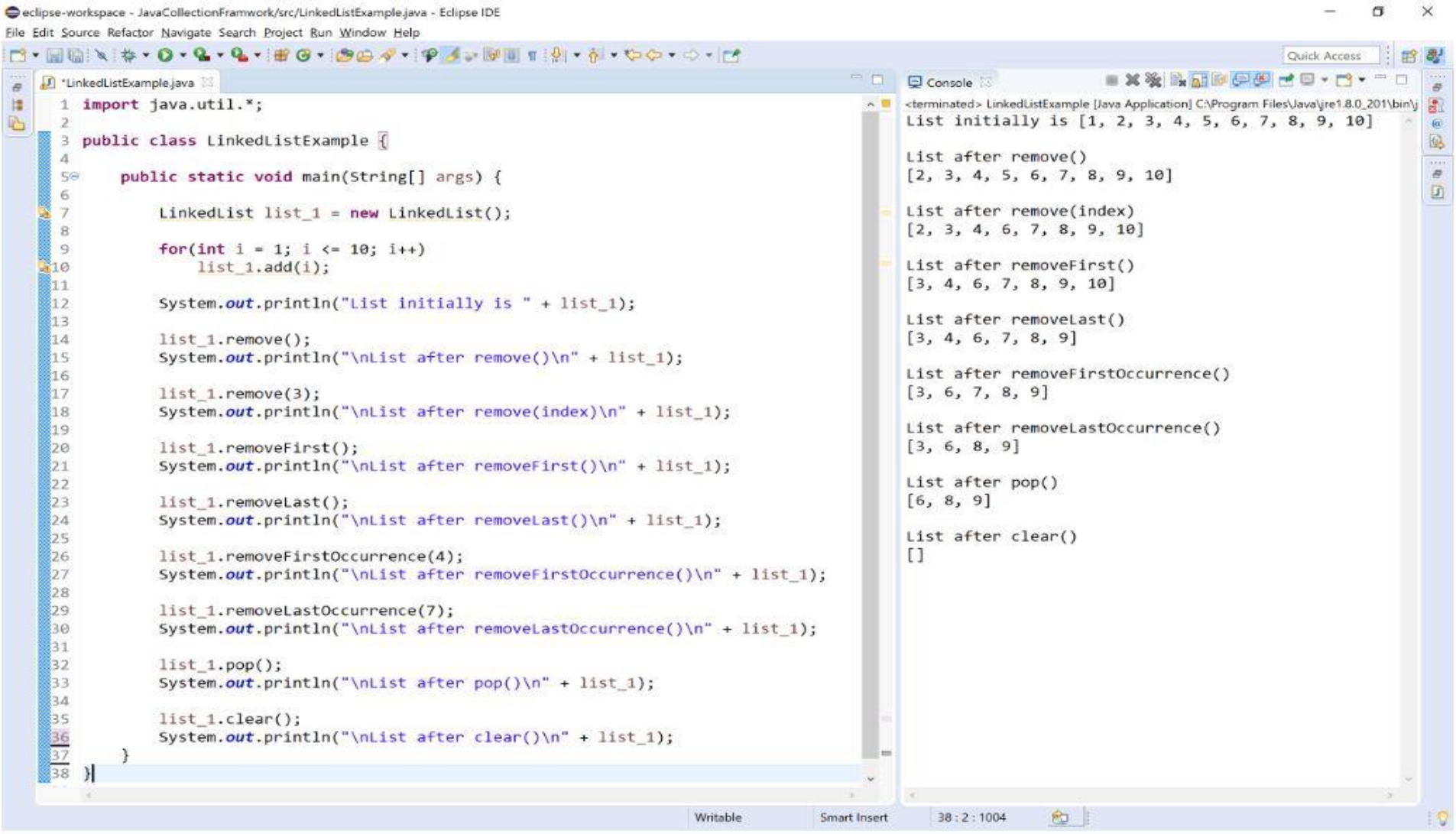
* `E set(int index, E newElement)` - Replace the element at specified index with newElement in the invoking LinkedList.



Removing Items

The LinkedList class has the following methods to remove items.

- `E remove( )` - Removes the first element from the invoking LinkedList.
- `E remove(int index)` - Removes the element at specified index in the invoking LinkedList.
- `boolean remove(Object element)` - Removes the first occurrence of the given element from the invoking LinkedList.
- `E removeFirst( )` - Removes the first element from the invoking LinkedList.
- `E removeLast( )` - Removes the last element from the invoking LinkedList.
- `boolean removeFirstOccurrence(Object element)` - Removes from the first occurrence of the given element from the invoking LinkedList.
- `boolean removeLastOccurrence(Object element)` - Removes from the last occurrence of the given element from the invoking LinkedList.
- `E poll( )` - Removes the first element from the LinkedList, and returns null if the list is empty.
- `E pollFirst( )` - Removes the first element from the LinkedList, and returns null if the list is empty.
- `E pollLast( )` - Removes the last element from the LinkedList, and returns null if the list is empty.
- `E pop( )` - Removes the first element from the LinkedList.
- `void clear( )` - Removes all the elements from the LinkedList.


eclipse-workspace - JavaCollectionFramwork/src/LinkedListExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*LinkedListExample.java
1 import java.util.*;
2
3 public class LinkedListExample {
4
5 public static void main(String[] args) {
6
7 LinkedList list_1 = new LinkedList();
8
9 for(int i = 1; i <= 10; i++)
10 list_1.add(i);
11
12 System.out.println("List initially is " + list_1);
13
14 list_1.remove();
15 System.out.println("\nList after remove()\n" + list_1);
16
17 list_1.remove(3);
18 System.out.println("\nList after remove(index)\n" + list_1);
19
20 list_1.removeFirst();
21 System.out.println("\nList after removeFirst()\n" + list_1);
22
23 list_1.removeLast();
24 System.out.println("\nList after removeLast()\n" + list_1);
25
26 list_1.removeFirstOccurrence(4);
27 System.out.println("\nList after removeFirstOccurrence()\n" + list_1);
28
29 list_1.removeLastOccurrence(7);
30 System.out.println("\nList after removeLastOccurrence()\n" + list_1);
31
32 list_1.pop();
33 System.out.println("\nList after pop()\n" + list_1);
34
35 list_1.clear();
36 System.out.println("\nList after clear()\n" + list_1);
37 }
38 }
Console
<terminated> LinkedListExample [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\j
List initially is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List after remove()
[2, 3, 4, 5, 6, 7, 8, 9, 10]
List after remove(index)
[2, 3, 4, 6, 7, 8, 9, 10]
List after removeFirst()
[3, 4, 6, 7, 8, 9, 10]
List after removeLast()
[3, 4, 6, 7, 8, 9]
List after removeFirstOccurrence()
[3, 6, 7, 8, 9]
List after removeLastOccurrence()
[3, 6, 8, 9]
List after pop()
[6, 8, 9]
List after clear()
[]
Writable
Smart Insert
38 : 2 : 1004

Vector class

In java, the package `java.util` contains a class called `Vector` which implements the `List` interface.

The Vector is similar to an ArrayList. Like ArrayList Vector also maintains the insertion order. But Vector is synchronized, due to this reason, it is rarly used in the non-thread application. It also lead to poor performance.

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

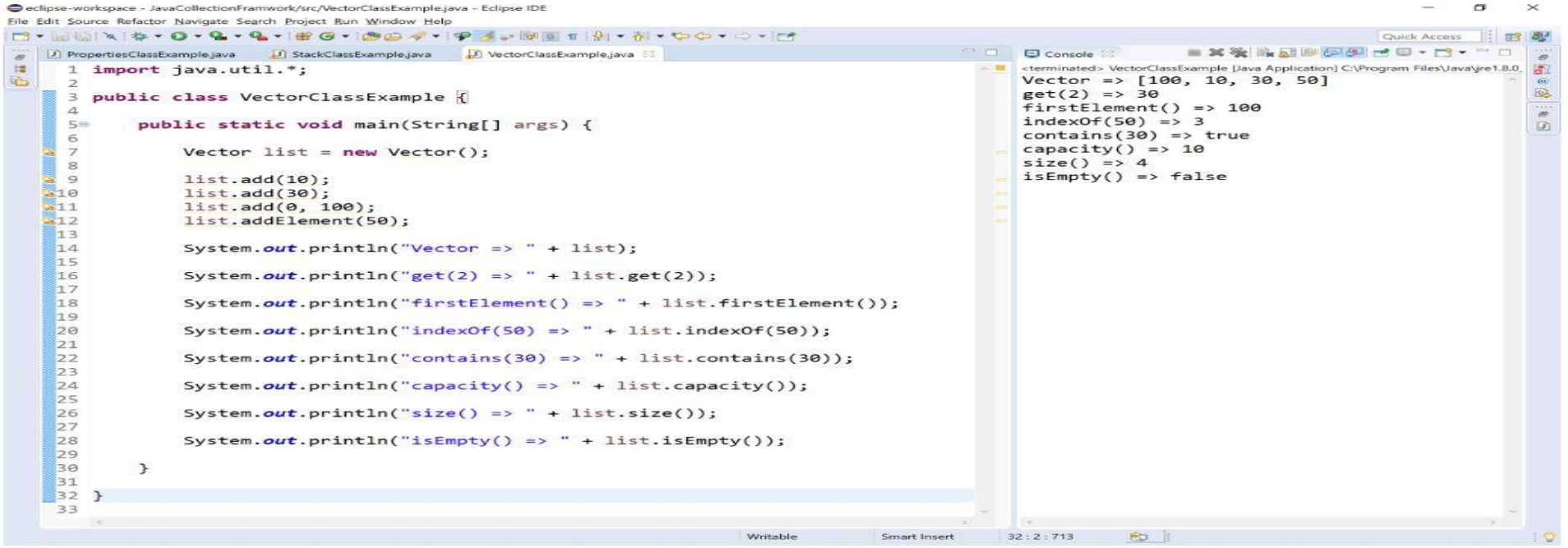
The Vector class in java has the following constructor.

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

The Vector class in java has the following methods.

S.No.	Methods with Description
1	**boolean add(Object o)** It appends the specified element to the end of this Vector.
2	**void add(int index, Object element)** It inserts the specified element at the specified position in this Vector.
3	**void addElement(Object obj)** Adds the specified object to the end of the vector, increasing its size by one.
4	**boolean addAll(Collection c)** It appends all of the elements in the specified Collection to the end of the Vector.
5	**boolean addAll(int index, Collection c)** It inserts all of the elements in in the specified Collection into the Vector at the specified position.
6	**Object set(int index, Object element)** It replaces the element at the specified position in the vector with the specified element.
7	**void setElementAt(Object obj, int index)** It sets the element at the specified index of the vector to be the specified object.
8	**Object remove(int index)** It removes the element at the specified position in the vector.
9	**boolean remove(Object o)** It removes the first occurrence of the specified element in the vector.
10	**boolean removeElement(Object obj)** It removes the first occurrence of the specified element in the vector.
11	**void removeElementAt(int index)** It removes the element at specified index in the vector.
12	**void removeRange(int fromIndex, int toIndex)** It removes from the Vector all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
13	**boolean removeAll(Collection c)** It removes from the vector all of its elements that are contained in the specified Collection.
14	**void removeAllElements()** It removes all the elements from the vector.
15	**boolean retainAll(Collection c)** It removes all the elements from the vector except elements those are in the given collection.

16	**Object elementAt(int index)** It returns the element at specified index in the Vector.
17	**Object get(int index)** It returns the element at specified index in the Vector.
18	**Enumeration elements()** It returns the Enumeration of all the elements of the Vector.
19	**Object firstElement()** It returns the first element of the Vector.
20	**Object lastElement()** It returns the last element of the Vector.
21	**int indexOf(Object element)** It returns the index value of the first occurence of the given element in the Vector.
22	**int indexOf(Object elem, int index)** It returns the index value of the first occurence of the given element, search beginning at specified index in the Vector.
23	**int lastIndexOf(Object elememnt)** It returns the index value of the last occurence of the given element, search beginning at specified index in the Vector.
24	**List subList(int fromIndex, int toIndex)** It returns a list containing elements fromIndex to toIndex in the Vector.
25	**int capacity()** It returns the current capacity of the Vector.
26	**void clear()** It removes all the elements from the Vector.
27	**Object clone()** It returns a clone of the Vector.
28	**boolean contains(Object element)** It returns true if element found in the Vector, otherwise returns false.




```java
import java.util.*;

public class VectorClassExample {

    public static void main(String[] args) {

        Vector list = new Vector();

        list.add(10);
        list.add(30);
        list.add(0, 100);
        list.addElement(50);

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

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

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

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

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

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

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

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

    }

}
```

Stack

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

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

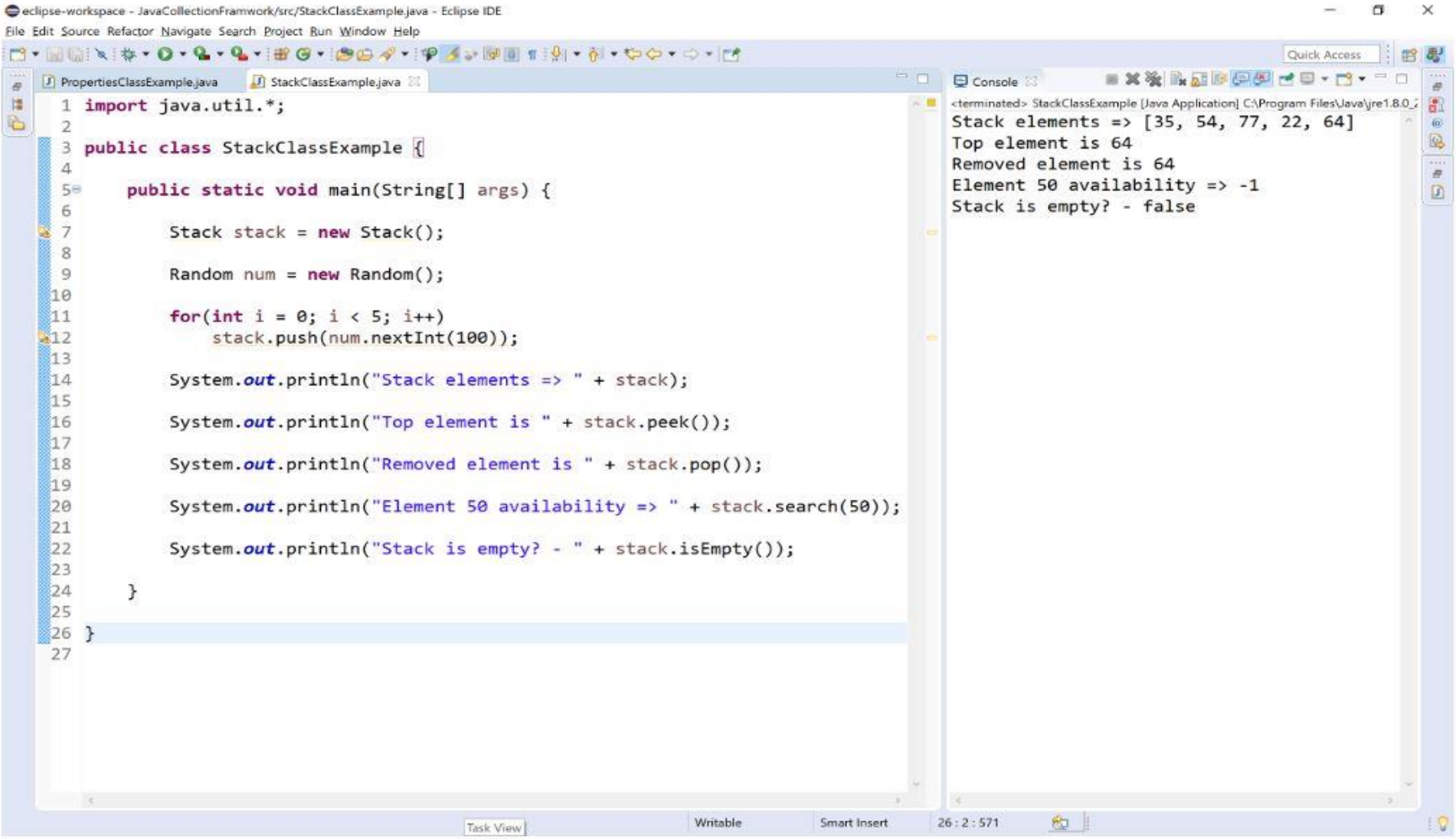
In Stack, the elements are added to the top of the stack and removed from the top of the stack.

The Stack class in java has the following constructor.

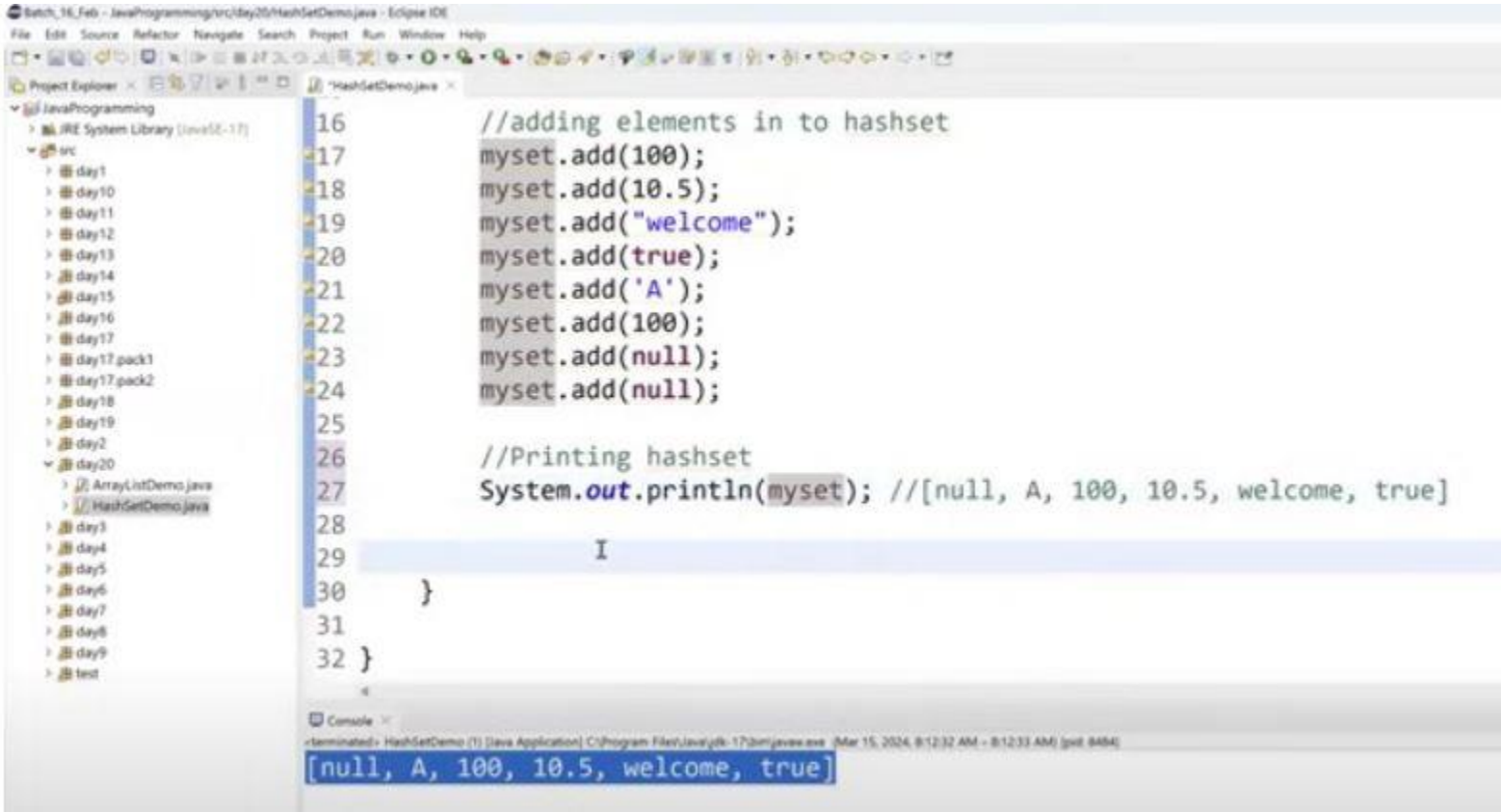
S. No.	Constructor with Description
1	**Stack( )** It creates an empty Stack.

The Stack class in java has the following methods.

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


eclipse-workspace - JavaCollectionFramwork/src/StackClassExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
PropertiesClassExample.java
StackClassExample.java
1 import java.util.*;
2
3 public class StackClassExample {
4
5 public static void main(String[] args) {
6
7 Stack stack = new Stack();
8
9 Random num = new Random();
10
11 for(int i = 0; i < 5; i++)
12 stack.push(num.nextInt(100));
13
14 System.out.println("Stack elements => " + stack);
15
16 System.out.println("Top element is " + stack.peek());
17
18 System.out.println("Removed element is " + stack.pop());
19
20 System.out.println("Element 50 availability => " + stack.search(50));
21
22 System.out.println("Stack is empty? - " + stack.isEmpty());
23
24 }
25
26 }
27
Console
<terminated> StackClassExample [Java Application] C:\Program Files\Java\jre1.8.0_
Stack elements => [35, 54, 77, 22, 64]
Top element is 64
Removed element is 64
Element 50 availability => -1
Stack is empty? - false
Task View
Writable
Smart Insert
26 : 2 : 571

SET (The Set is an non-indexed sequence)



Hash set

The `HashSet` class is a part of java collection framework. It is available inside the `java.util` package. The HashSet class extends `AbstractSet` class and implements `Set` interface.

The elements of HashSet are organized using a mechanism called hashing. The HashSet is used to create hash table for storing set of elements.

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

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

HashSet class declaration

The HashSet class has the following declaration.

Example

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

HashSet class constructors

The HashSet class has the following constructors.

* `HashSet( )` - Creates an empty HashSet with the default initial capacity (16).
* `HashSet(Collection c)` - Creates a HashSet with given collection of elements.
* `HashSet(int initialCapacity)` - Creates an empty HashSet with the specified initial capacity.
* `HashSet(int initialCapacity, float loadFactor)` - Creates an empty HashSet with the specified initial capacity and loadFactor.

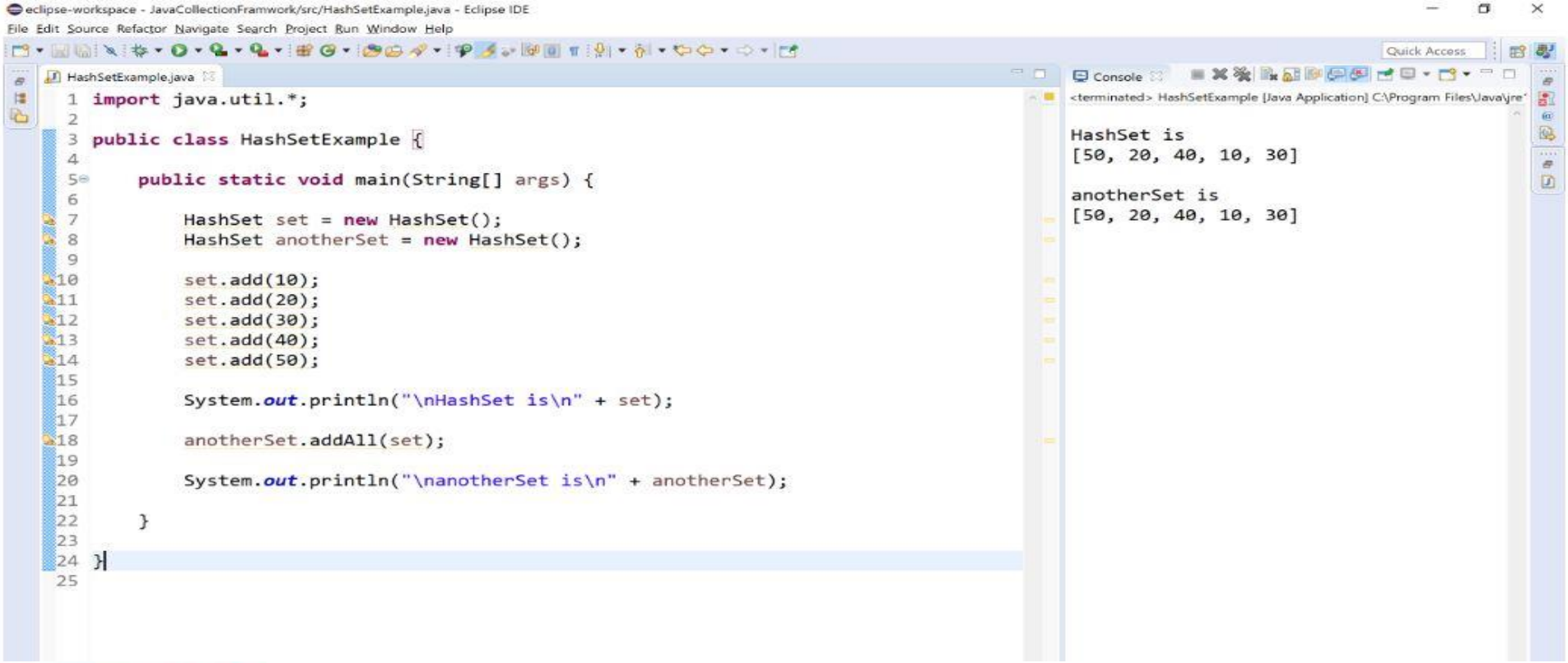
Operations on HashSet

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

Adding Items

The HashSet class has the following methods to add items.

* `boolean add(E element)` - Inserts given element to the HashSet.
* `boolean addAll(Collection c)` - Inserts given collection of elements to the HashSet.



Accessing Items

The HashSet class has no methods to access items, we can access whole set using its name.

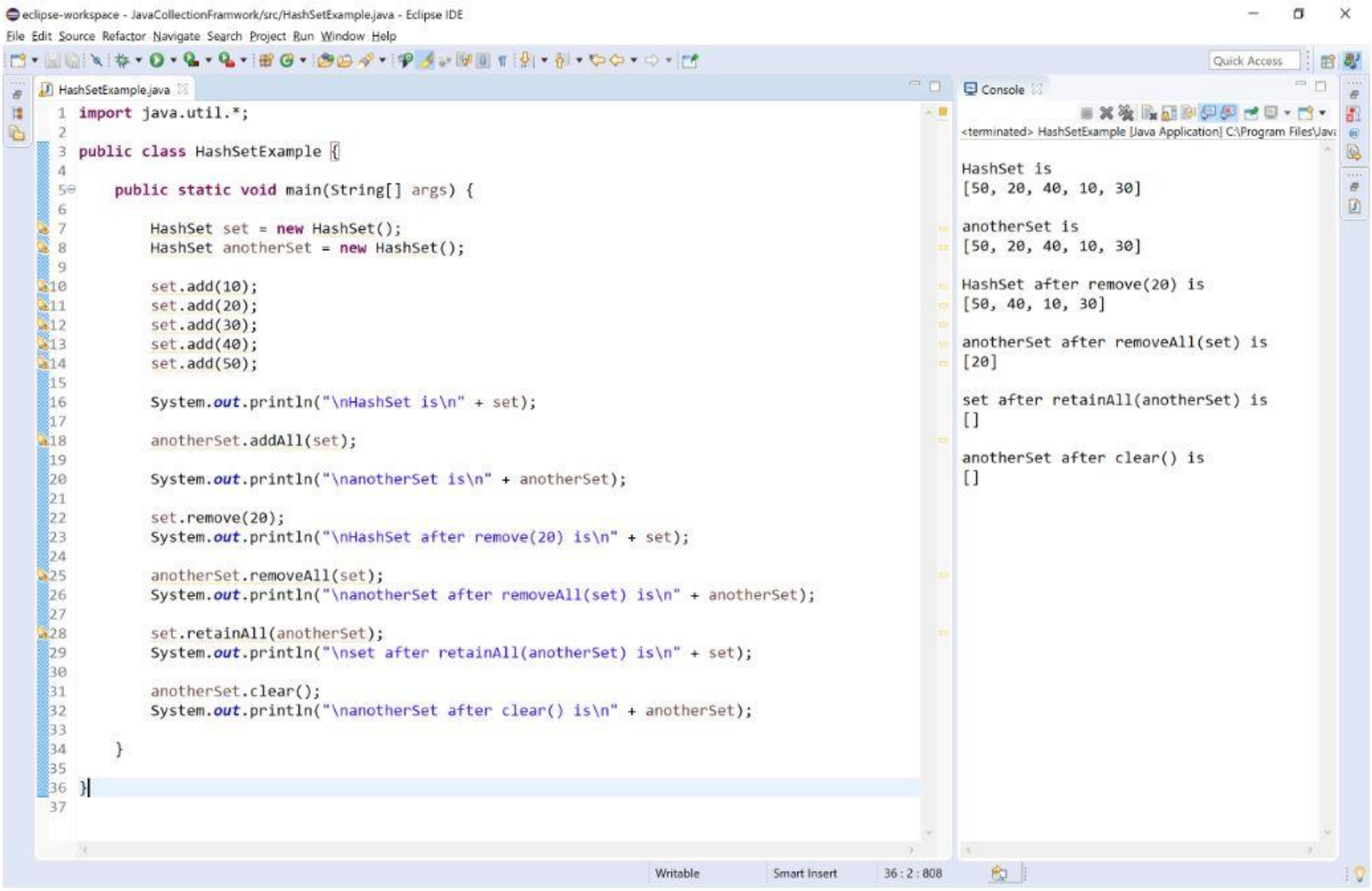
Updating Items

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

Removing Items

The HashSet class has the following methods to remove items.

- `boolean remove(Object o)` - Removes the specified element from the invoking HashSet.
- `boolean removeAll(Collection c)` - Removes all the elements of specified collection from the invoking HashSet.
- `boolean removeIf(Predicate p)` - Removes all of the elements of HashSet collection that satisfy the given predicate.
- `boolean retainAll(Collection c)` - Removes all of the elements of HashSet collection except specified collection of elements.
- `void clear( )` - Removes all the elements from the HashSet.


eclipse-workspace - JavaCollectionFramwork/src/HashSetExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
HashSetExample.java
1 import java.util.*;
2
3 public class HashSetExample {
4
5 public static void main(String[] args) {
6
7 HashSet set = new HashSet();
8 HashSet anotherSet = new HashSet();
9
10 set.add(10);
11 set.add(20);
12 set.add(30);
13 set.add(40);
14 set.add(50);
15
16 System.out.println("\nHashSet is\n" + set);
17
18 anotherSet.addAll(set);
19
20 System.out.println("\nanotherSet is\n" + anotherSet);
21
22 set.remove(20);
23 System.out.println("\nHashSet after remove(20) is\n" + set);
24
25 anotherSet.removeAll(set);
26 System.out.println("\nanotherSet after removeAll(set) is\n" + anotherSet);
27
28 set.retainAll(anotherSet);
29 System.out.println("\nset after retainAll(anotherSet) is\n" + set);
30
31 anotherSet.clear();
32 System.out.println("\nanotherSet after clear() is\n" + anotherSet);
33
34 }
35
36 }
37
Console
<terminated> HashSetExample [Java Application] C:\Program Files\Java
HashSet is
[50, 20, 40, 10, 30]
anotherSet is
[50, 20, 40, 10, 30]
HashSet after remove(20) is
[50, 40, 10, 30]
anotherSet after removeAll(set) is
[20]
set after retainAll(anotherSet) is
[]
anotherSet after clear() is
[]
Writable
Smart Insert
36 : 2 : 808

Tree Set

The `TreeSet` class is a part of java collection framework. It is available inside the `java.util` package. The TreeSet class extends `AbstractSet` class and implements `NavigableSet`, `Cloneable`, and `Serializable` interfaces.

The elements of TreeSet are organized using a mechanism called tree. The TreeSet class internally uses a TreeMap to store elements. The elements in a TreeSet are sorted according to their natural ordering.

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

TreeSet class declaration

The TreeSet class has the following declaration.

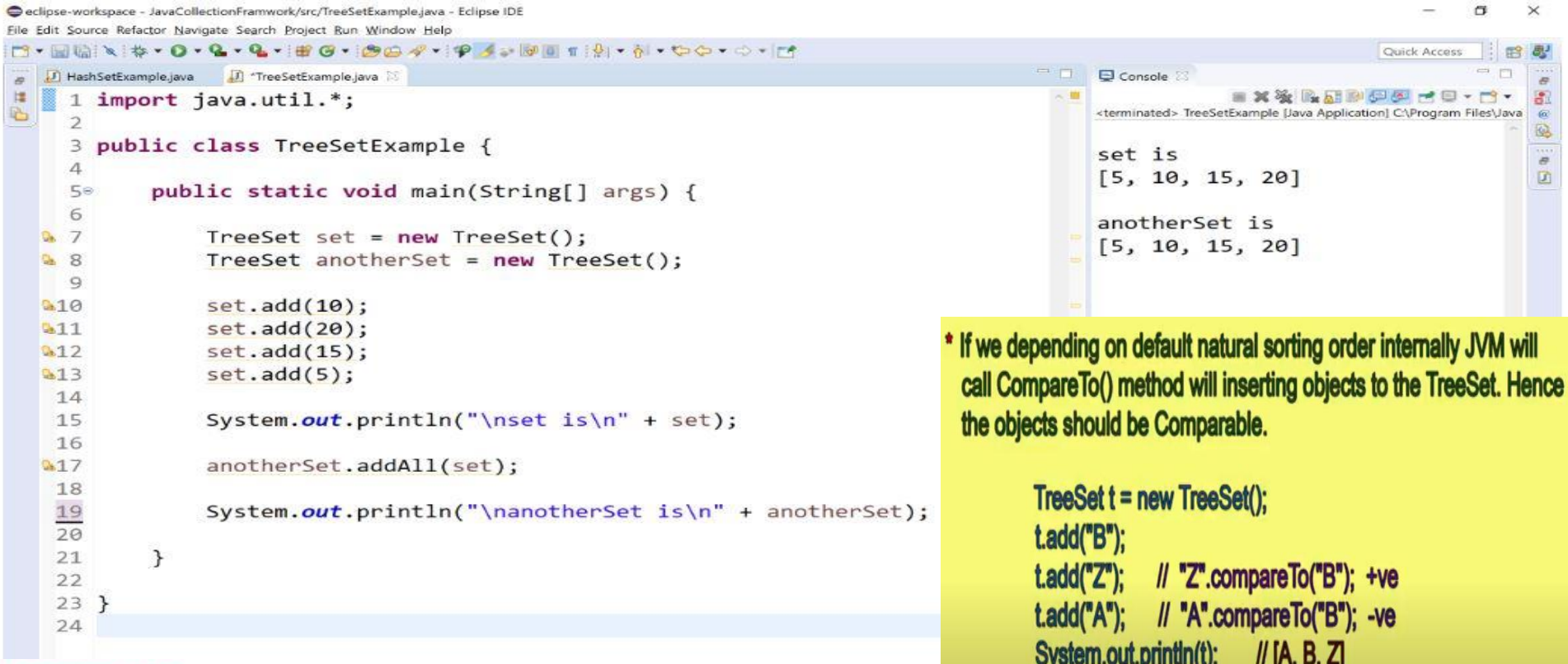
Example sorting order

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

TreeSet class constructors

The TreeSet class has the following constructors.

* `TreeSet( )` - Creates an empty TreeSet in which elements will get stored in default natural sorting order.
* `TreeSet(Collection c)` - Creates a TreeSet with given collection of elements.
* `TreeSet(Comparator c)` - Creates an empty TreeSet with the specified sorting order.
* `TreeSet(SortedSet s)` - This constructor is used to convert SortedSet to TreeSet.



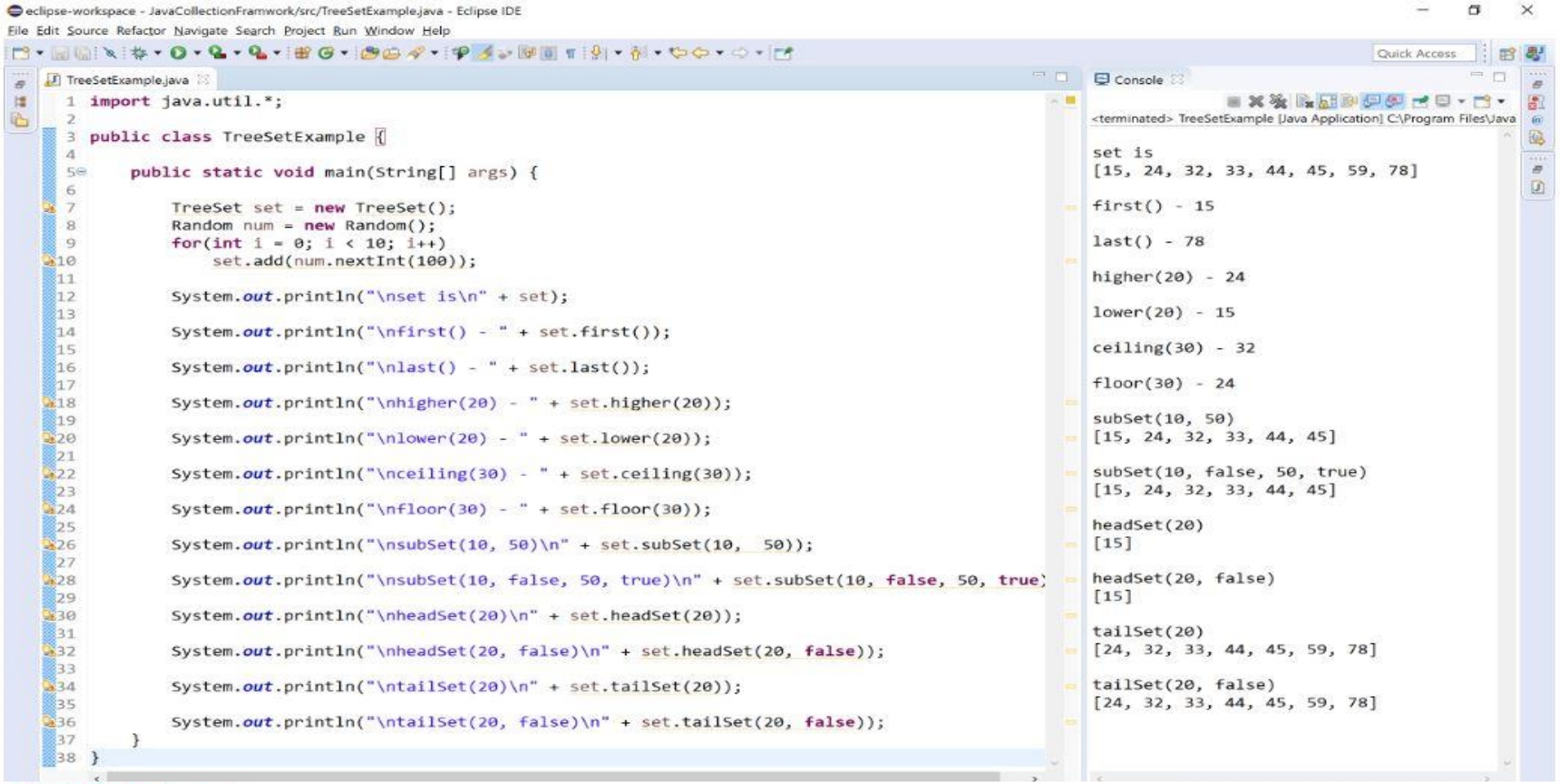
* If we depending on default natural sorting order internally JVM will call CompareTo() method will inserting objects to the TreeSet. Hence the objects should be Comparable.

```
TreeSet t = new TreeSet();
t.add("B");
t.add("Z");     // "Z".compareTo("B"); +ve
t.add("A");     // "A".compareTo("B"); -ve
System.out.println(t);      // [A, B, Z]
```

Accessing Items

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

* `E First( )` - Returns the first (smallest) element from the invoking TreeSet.
* `E last( )` - Returns the last (largest) element from the invoking TreeSet.
* `E higher(E obj)` - Returns the largest element e such that e>obj. If it does not found returns null.
* `E lower(E obj)` - Returns the largest element e such that e<obj. If it does not found returns null.
* `E ceiling(E obj)` - Returns the smallest element e such that e>=obj. If it does not found returns null.
* `E floor(E obj)` - Returns the largest element e such that e<=obj. If it does not found returns null.
* `SortedSet subSet(E fromElement, E toElement)` - Returns a set of elements that lie between the given range which includes fromElement and excludes toElement.
* `NavigableSet subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)` - Returns a set of elements that lie between the given range from the invoking TreeSet.
* `SortedSet tailSet(E fromElement)` - Returns a set of elements that are greater than or equal to the specified fromElement from the invoking TreeSet.
* `NavigableSet tailSet(E fromElement, boolean inclusive)` - Returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element from the invoking TreeSet.
* `SortedSet headSet(E fromElement)` - Returns a set of elements that are smaller than or equal to the specified fromElement from the invoking TreeSet.
* `NavigableSet headSet(E fromElement, boolean inclusive)` - Returns a set of elements that are smaller than or equal to (if, inclusive is true) the specified element from the invoking TreeSet.



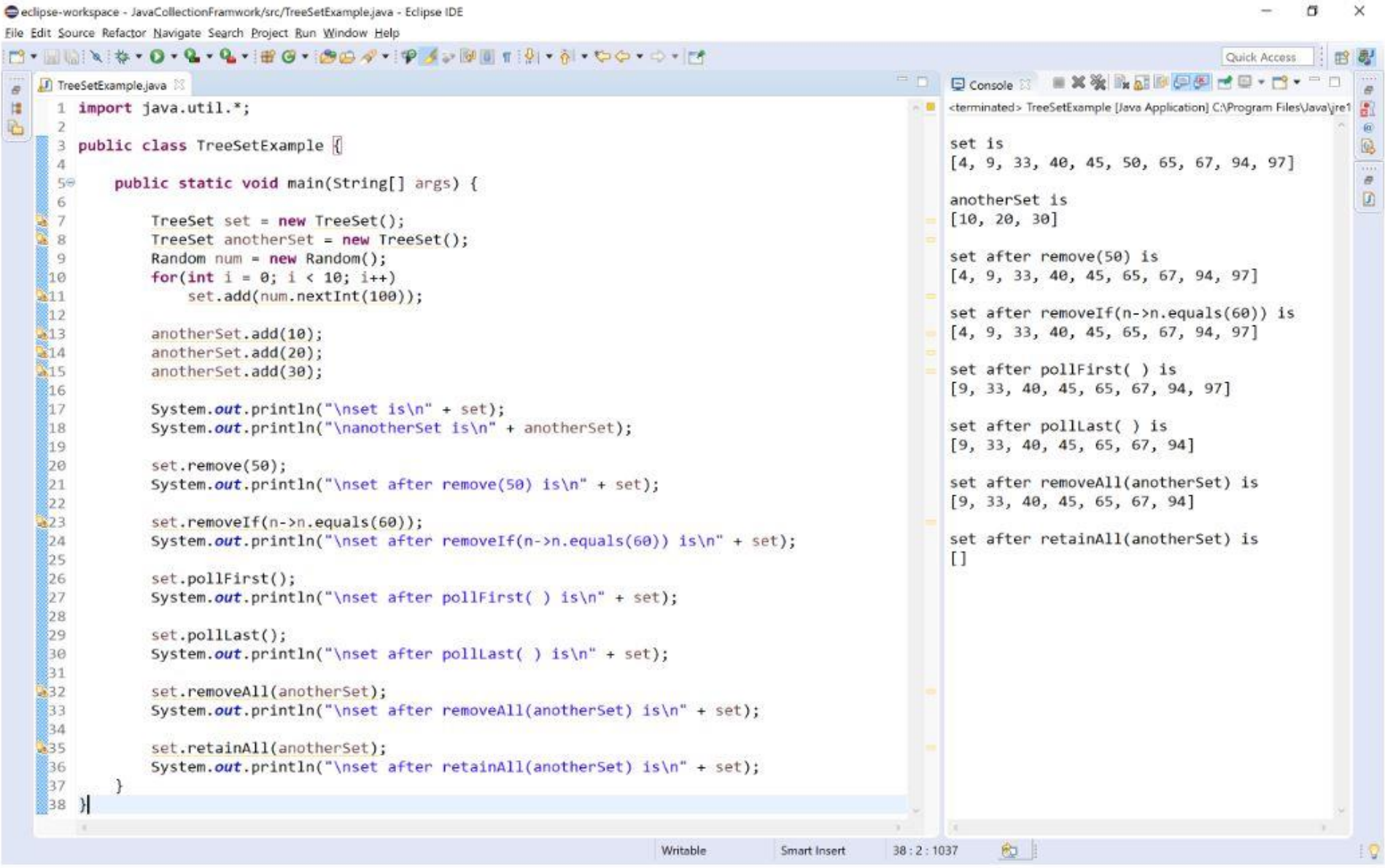
Updating Items

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

Removing Items

The TreeSet class has the following methods to remove items.

- `boolean remove(Object o)` - Removes the specified element from the invoking TreeSet.
- `boolean removeAll(Collection c)` - Removes all the elements those are in the specified collection from the invoking TreeSet.
- `boolean removeIf(Predicate p)` - Removes all of the elements of the TreeSet collection that satisfy the given predicate.
- `boolean retainAll(Collection c)` - Removes all the elements except those are in the specified collection from the invoking TreeSet.
- `E pollFirst( )` - Removes the first (smallest) element from the invoking TreeSet, and returns the same.
- `E pollLast( )` - Removes the last (largest) element from the invoking TreeSet, and returns the same.
- `void clear( )` - Removes all the elements from the TreeSet.


eclipse-workspace - JavaCollectionFramwork/src/TreeSetExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
TreeSetExample.java
1 import java.util.*;
2
3 public class TreeSetExample {
4
5 public static void main(String[] args) {
6
7 TreeSet set = new TreeSet();
8 TreeSet anotherSet = new TreeSet();
9 Random num = new Random();
10 for(int i = 0; i < 10; i++)
11 set.add(num.nextInt(100));
12
13 anotherSet.add(10);
14 anotherSet.add(20);
15 anotherSet.add(30);
16
17 System.out.println("\nset is\n" + set);
18 System.out.println("\nanotherSet is\n" + anotherSet);
19
20 set.remove(50);
21 System.out.println("\nset after remove(50) is\n" + set);
22
23 set.removeIf(n->n.equals(60));
24 System.out.println("\nset after removeIf(n->n.equals(60)) is\n" + set);
25
26 set.pollFirst();
27 System.out.println("\nset after pollFirst( ) is\n" + set);
28
29 set.pollLast();
30 System.out.println("\nset after pollLast( ) is\n" + set);
31
32 set.removeAll(anotherSet);
33 System.out.println("\nset after removeAll(anotherSet) is\n" + set);
34
35 set.retainAll(anotherSet);
36 System.out.println("\nset after retainAll(anotherSet) is\n" + set);
37 }
38 }
Console
<terminated> TreeSetExample [Java Application] C:\Program Files\Java\jre1
set is
[4, 9, 33, 40, 45, 50, 65, 67, 94, 97]
anotherSet is
[10, 20, 30]
set after remove(50) is
[4, 9, 33, 40, 45, 65, 67, 94, 97]
set after removeIf(n->n.equals(60)) is
[4, 9, 33, 40, 45, 65, 67, 94, 97]
set after pollFirst( ) is
[9, 33, 40, 45, 65, 67, 94, 97]
set after pollLast( ) is
[9, 33, 40, 45, 65, 67, 94]
set after removeAll(anotherSet) is
[9, 33, 40, 45, 65, 67, 94]
set after retainAll(anotherSet) is
[]
Writable
Smart Insert
38 : 2 : 1037

Map

The java collection framework has an interface `Map` that is available inside the `java.util` package. The Map interface is not a subtype of Collection interface.

- The `Map` stores the elements as a pair of key and value.
- The `Map` does not allows duplicate keys, but allows duplicate values.
- In a `Map`, each key can map to at most one value only.
- In a `Map`, the order of elements depends on specific implementations, e.g TreeMap and LinkedHashMap have predictable order, while HashMap does not.

The `Map` interface has the following child interfaces.

Interface	Description
Map	Maps unique key to value.
Map.Entry	Describe an element in key and value pair in a map. Entry is sub interface of Map.
SortedMap	It is a child of Map so that key are maintained in an ascending order.
NavigableMap	It is a child of SortedMap to handle the retrienal of entries based on closest match searches.

The `Map` interface has the following three classes.

Class	Description
HashMap	It implements the Map interface, but it doesn't maintain any order.
LinkedHashMap	It implements the Map interface, it also extends HashMap class. It maintains the insertion order.
TreeMap	It implements the Map and SortedMap interfaces. It maintains the ascending order.

Map Interface methods

The Map interface contains methods for handling elements of a map. It has the following methods.

The insertion of a key, value pair is said to be an entry in the map terminology.

Method	Description
V put(Object key, Object value)	Inserts an entry in the map.
void putAll(Map map)	Inserts the specified map into the invoking map.
V putIfAbsent(K key, V value)	Inserts the specified value with the specified key in the map only if that key does not exist.
Set keySet( )	Returns a Set that contains all the keys of invoking Map.
Collection values( )	Returns a collection that contains all the values of invoking Map.
Set<Map.Entry<K,V>> entrySet()	Returns a Set that contains all the keys and values of invoking Map.
V get(Object key)	Returns the value associated with the specified key.
V getOrDefault(Object key, V defaultValue)	Returns the value associated with the specified key, or defaultValue if the map does not contain the key.
boolean containsValue(Object value)	Returns true if specified value found in the map, else return false.
boolean containsKey(Object key)	Returns true if specified key found in the map, else return false.
V replace(K key, V value)	Used to replace the specified value for the specified key.
boolean replace(K key, V oldValue, V newValue)	Used to replaces the oldValue with the newValue for a specified key.
void replaceAll(BiFunction function)	Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
V merge(K key, V value, BiFunction remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V compute(K key, BiFunction remappingFunction)	Used to compute a mapping for the specified key and its current mapped value.
V computeIfAbsent(K key, Function mappingFunction)	Used to compute its value using the given mapping function, if the specified key is not already associated with a value, and enters it into this map unless null.
V computeIfPresent(K key, BiFunction remappingFunction)	Used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
void forEach(BiConsumer action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V remove(Object key)	Removes an entry for the specified key.
boolean remove(Object key, Object value)	Removes the specified values with the associated specified keys from the map.

The java collection framework has an interface `Map` that is available inside the `java.util` package. The Map interface is not a subtype of Collection interface.

The `Map` interface has the following three classes.

Class	Description
HashMap	It implements the Map interface, but it doesn't maintain any order.
LinkedHashMap	It implements the Map interface, it also extends HashMap class. It maintains the insertion order.
TreeMap	It implements the Map and SortedMap interfaces. It maintains the ascending order.

Commonly used methods defined by Map interface

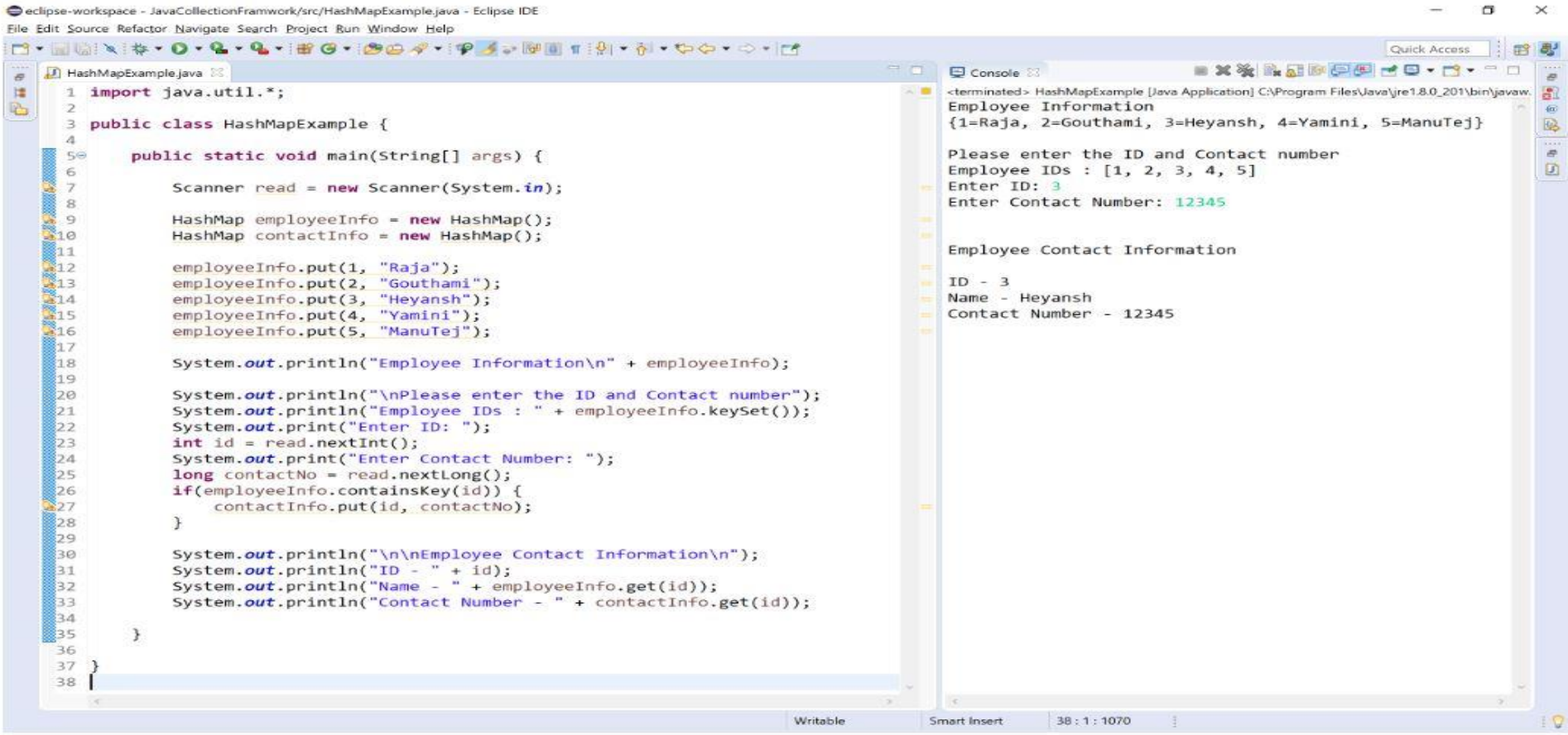
Method	Description
Object put(Object k, Object v)	It performs an entry into the Map.
Object putAll(Map m)	It inserts all the entries of m into invoking Map.
Object get(Object k)	It returns the value associated with given key.
boolean containsKey(Object k)	It returns true if map contain k as key. Otherwise false.
Set keySet()	It returns a set that contains all the keys from the invoking Map.
Set valueSet()	It returns a set that contains all the values from the invoking Map.
Set entrySet()	It returns a set that contains all the entries from the invoking Map.

HashMap Class

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

Key Properties of HashMap

* HashMap is a child class of AbstractMap class.
* HashMap implements the interfeaces Map, Cloneable, and Serializable.
* HashMap stores data as a pair of key and value.
* HashMap uses Hash table concept to store the data.
* HashMap does not allow duplicate keys, but values may be repeated.
* HashMap allows only one null key and multiple null values.
* HashMap does not follow any oreder.
* HashMap has the default capacity 16 entries.

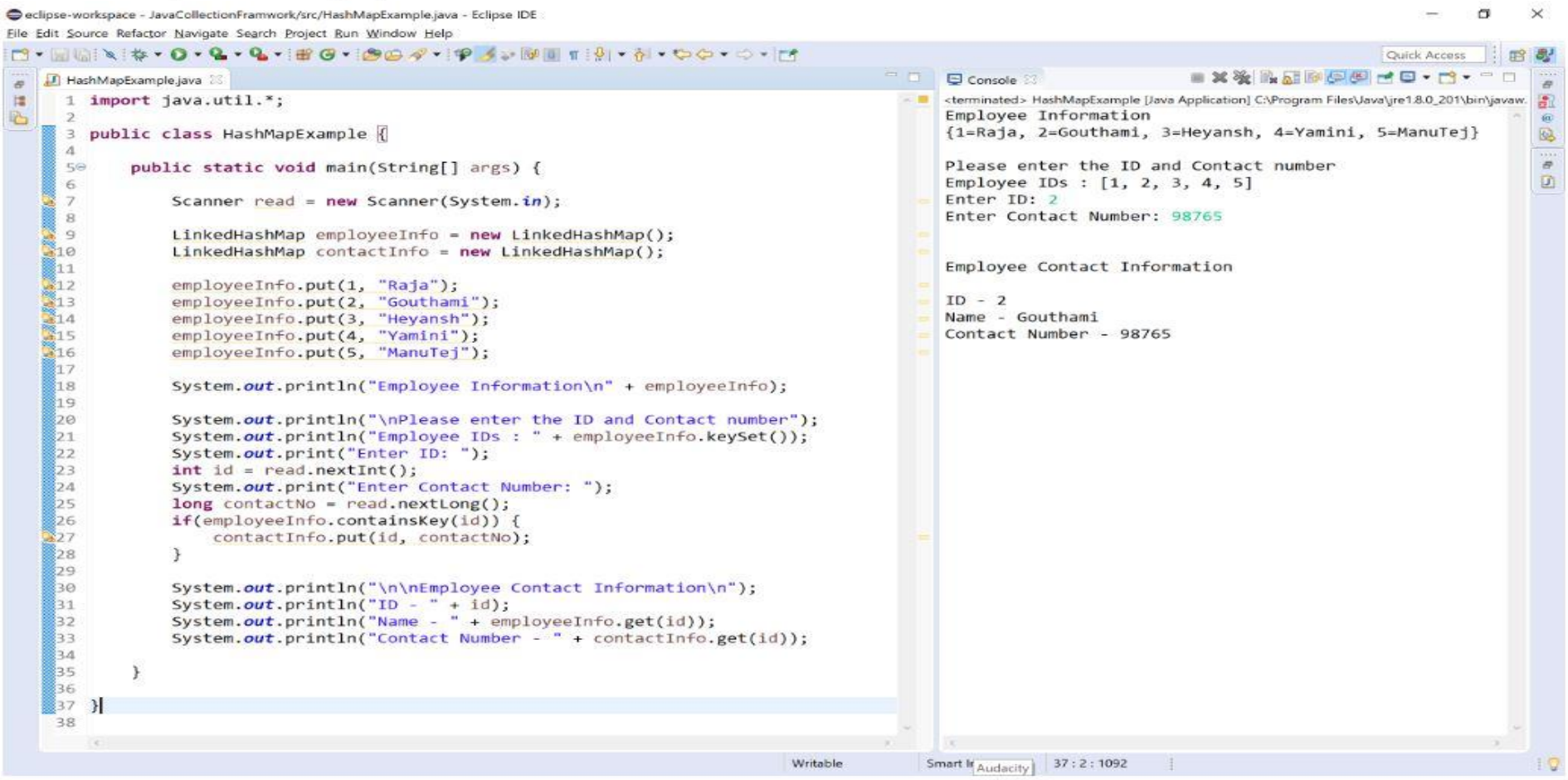


LinkedHashMap Class

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

Key Properties of LinkedHashMap

* LinkedHashMap is a child class of HashMap class.
* LinkedHashMap implements the Map interface.
* LinkedHashMap stores data as a pair of key and value.
* LinkedHashMap uses Hash table concept to store the data.
* LinkedHashMap does not allow duplicate keys, but values may be repeated.
* LinkedHashMap allows only one null key and multiple null values.
* LinkedHashMap follows the insertion oreder.
* LinkedHashMap has the default capacity 16 entries.

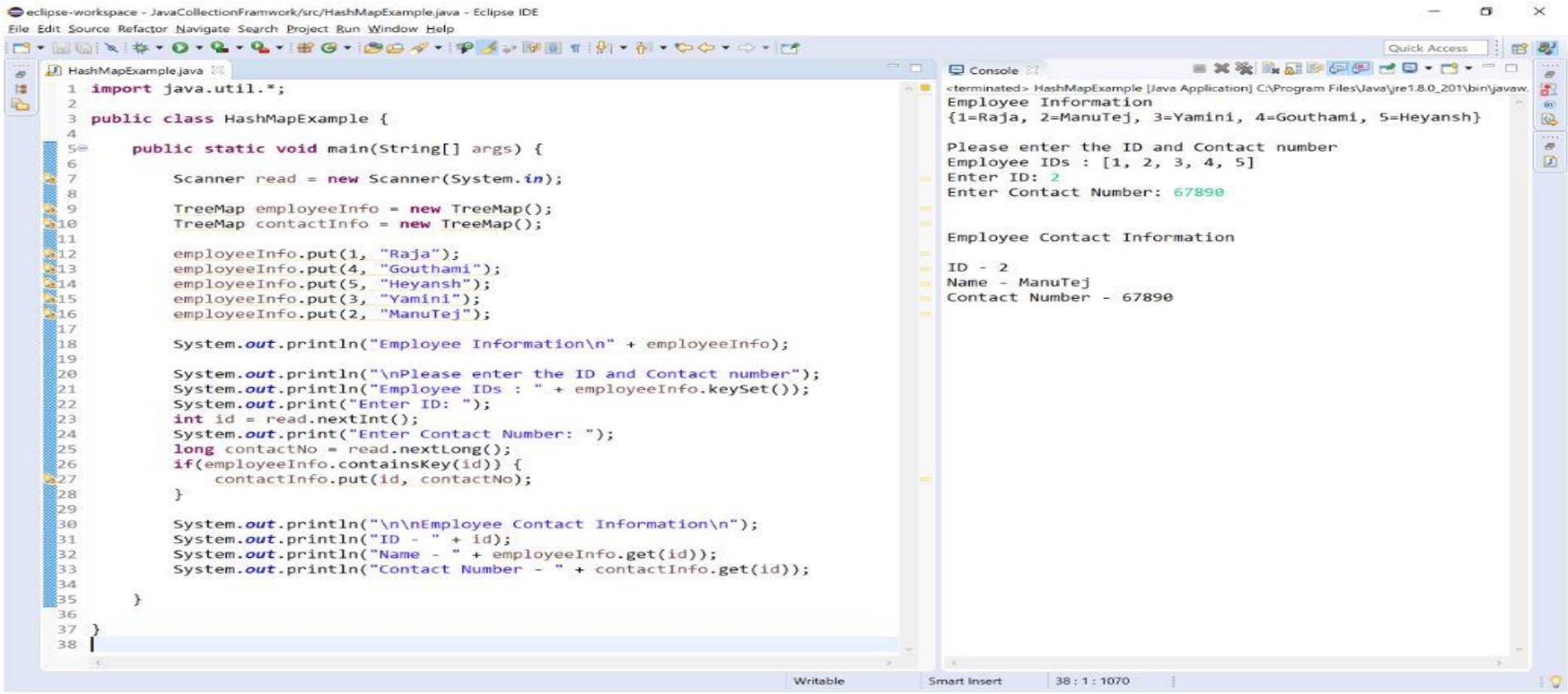


TreeMap Class

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

Key Properties of TreeMap

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



Enumeration

In java, an Enumeration is a list of named constants. The enum concept was introduced in Java SE 5 version. The enum in Java programming the concept of enumeration is greatly expanded with lot more new features compared to the other languages like C, and C++.

In java, the enumeration concept was defined based on the class concept. When we create an enum in java, it converts into a class type. This concept enables the java enum to have constructors, methods, and instance variables.

All the constants of an enum are *public*, *static*, and *final*. As they are static, we can access directly using enum name.

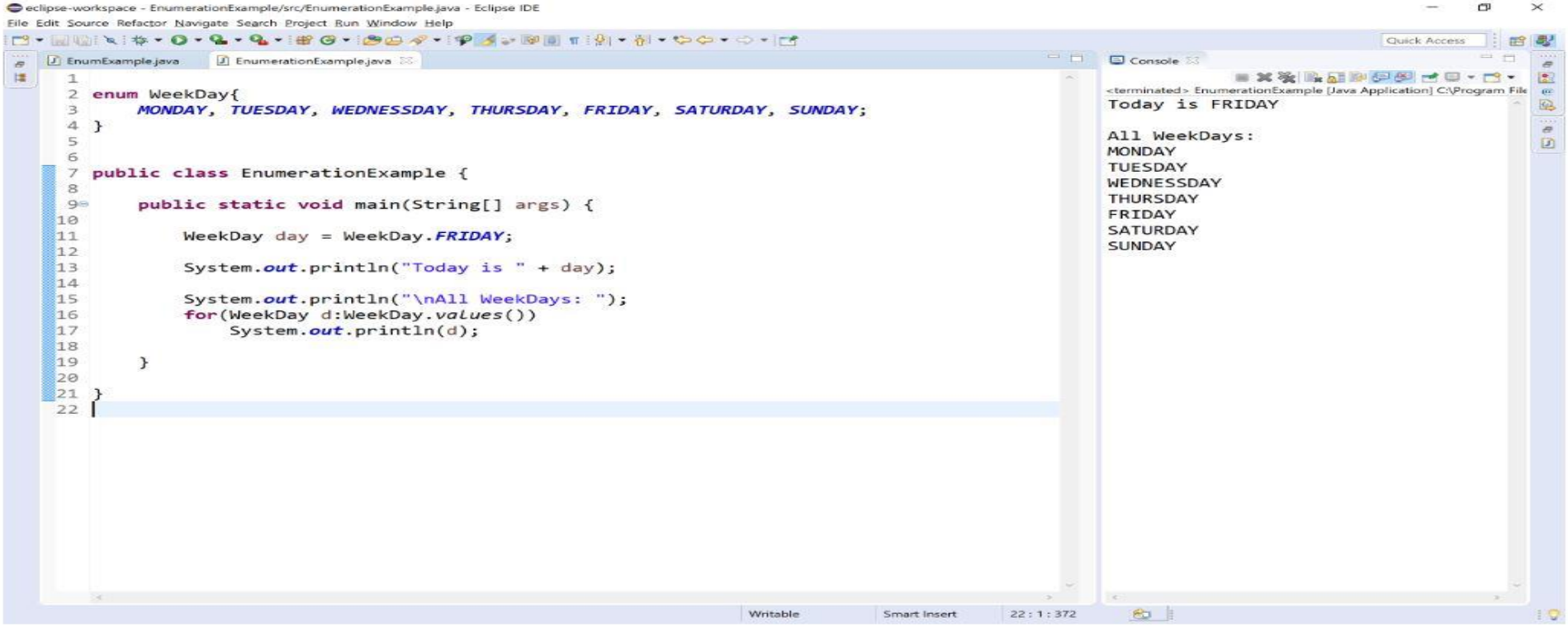
The main objective of enum is to define our own data types in Java, and they are said to be enumeration data types.

Creating enum in Java

To create enum in Java, we use the keyword *enum*. The syntax for creating enum is similar to that of class.

In java, an enum can be defined outside a class, inside a class, but not inside a method.

Let's look at the following example program for creating a basic enum.

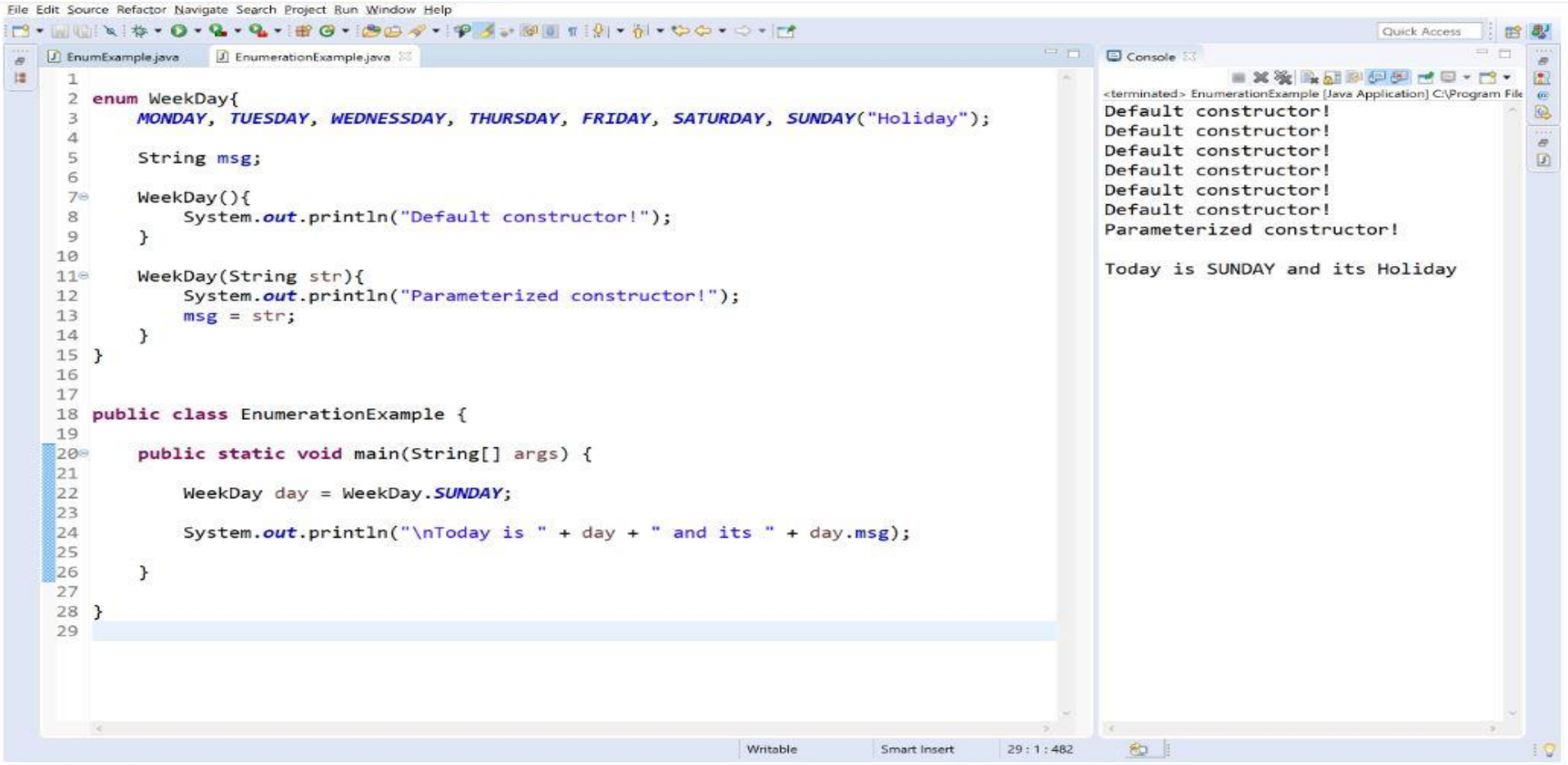


Constructors in Java enum

In a java programming language, an enum can have both the type of constructors default and parameterized. The execution of constructor depends on the constants we defined in the enum. For every constant in an enum, the constructor is executed.

If an enum has a parameterized constructor, the parameter value should be passed when we define constant.

Let's look at the following example program for illustrating constructors in enum.




```
1
2  enum WeekDay{
3      MONDAY, TUESDAY, WEDNESSDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY("Holiday");
4
5      String msg;
6
7      WeekDay(){
8          System.out.println("Default constructor!");
9      }
10
11     WeekDay(String str){
12         System.out.println("Parameterized constructor!");
13         msg = str;
14     }
15 }
16
17
18 public class EnumerationExample {
19
20     public static void main(String[] args) {
21
22         WeekDay day = WeekDay.SUNDAY;
23
24         System.out.println("\nToday is " + day + " and its " + day.msg);
25
26     }
27
28 }
29
```

In the above example, the constant `SUNDAY` is created by calling the parameterized constructor, other constants created by calling default constructor.

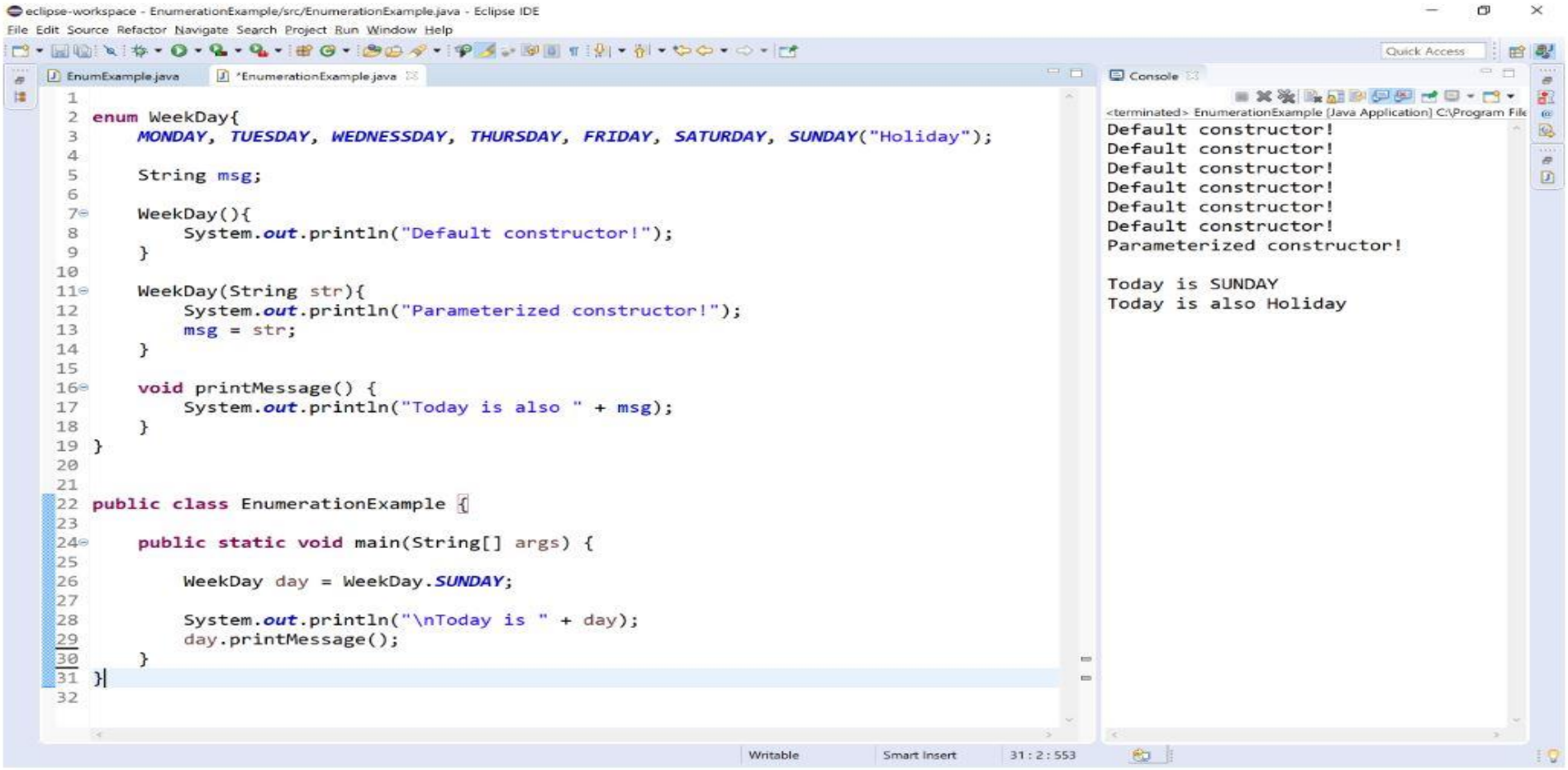
In the above example, the constant `SUNDAY` is created by calling the parameterized constructor, other constants created by calling default constructor.

Methods in Java enum

In a java programming language, an enum can have methods. These methods are similar to the methods of a class. All the methods defined in an enum can be used with all the constants defined by the same enum.

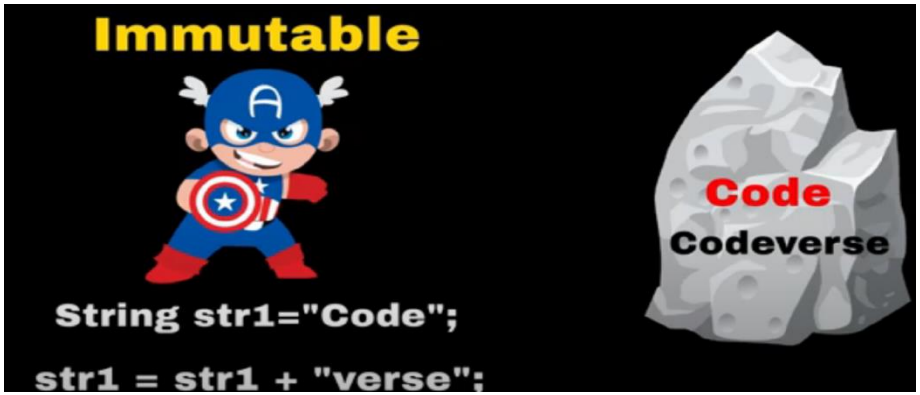
Let's look at the following example program for illustrating methods in enum.

When we run the above program, it produce the following output.



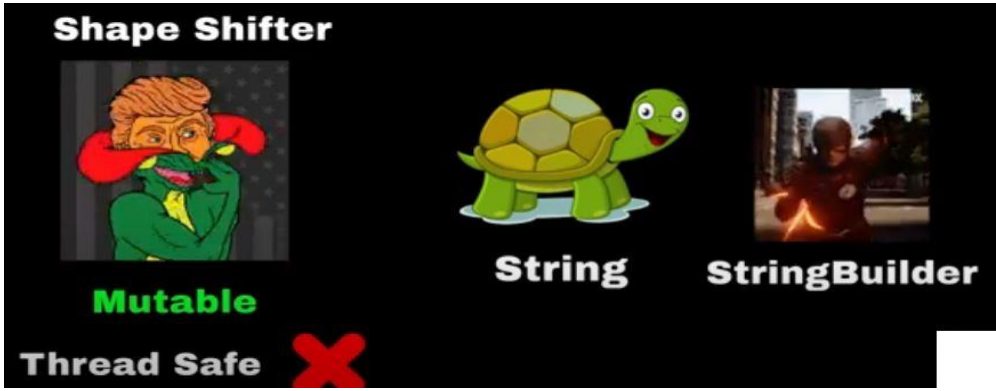
- In java, enum can not extend another enum and a class.
- In java, enum can implement interfaces.
- In java, enum does not allow to create an object of it.
- In java, every enum extends a built-in class `Enum` by default.

String Handling



String

String Buffer



String Builder

String

A string is a sequence of characters surrounded by double quotations. In a java programming language, a string is the object of a built-in class `String`.

In the background, the string values are organized as an array of a character data type.

The string created using a character array can not be extended. It does not allow to append more characters after its definition, but it can be modified.

Let's look at the following example java code.

Example

```
char[] name = {'J', 'a', 'v', 'a', ' ', 'T', 'u', 't', 'o', 'r', 'i', 'a', 'l', 's'};
//name[14] = '@';   //ArrayIndexOutOfBoundsException
name[5] = '-';
System.out.println(name);
```

The `String` class defined in the package `java.lang` package. The String class implements `Serializable`, `Comparable`, and `CharSequence` interfaces.

The string created using the `String` class can be extended. It allows us to add more characters after its definition, and also it can be modified.

Let's look at the following example java code.

Example

```
String siteName = "btechsmartclass.com";
siteName = "www.btechsmartclass.com";
```

Creating String object in java

In java, we can use the following two ways to create a string object.

- Using string literal
- Using String constructor

Let's look at the following example java code.

Example

```
String title = "Java Tutorials";   // Using literals

String siteName = new String("www.btechsmartclass.com"); // Using constructor
```

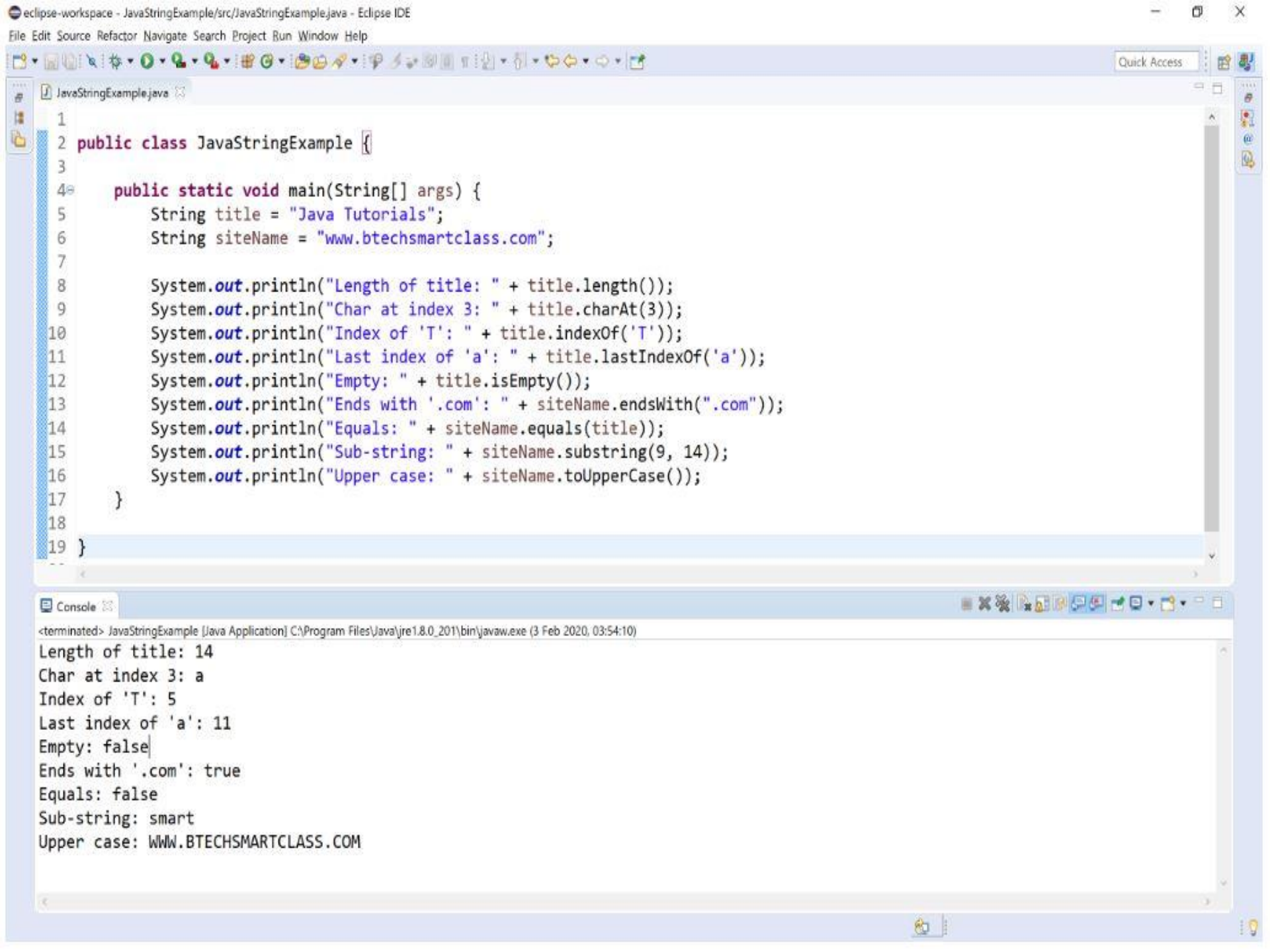
The String class constructor accepts both string and character array as an argument.

String handling methods

In java programming language, the String class contails various methods that can be used to handle string data values. It containg methods like concat( ), compareTo( ), split( ), join( ), replace( ), trim( ), length( ), intern( ), equals( ), comparison( ), substring( ), etc.

The following table depicts all built-in methods of String class in java.

Method	Description	Return Value
charAt(int)	Finds the character at given index	char
length()	Finds the length of given string	int
compareTo(String)	Compares two strings	int
compareToIgnoreCase(String)	Compares two strings, ignoring case	int
concat(String)	Concatenates the object string with argument string.	String
contains(String)	Checks whether a string contains sub-string	boolean
contentEquals(String)	Checks whether two strings are same	boolean
equals(String)	Checks whether two strings are same	boolean
equalsIgnoreCase(String)	Checks whether two strings are same, ignoring case	boolean
startsWith(String)	Checks whether a string starts with the specified string	boolean
endsWith(String)	Checks whether a string ends with the specified string	boolean
getBytes()	Converts string value to bytes	byte[]
hashCode()	Finds the hash code of a string	int


eclipse-workspace - JavaStringExample/src/JavaStringExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
JavaStringExample.java
1
2 public class JavaStringExample {
3
4 public static void main(String[] args) {
5 String title = "Java Tutorials";
6 String siteName = "www.btechsmartclass.com";
7
8 System.out.println("Length of title: " + title.length());
9 System.out.println("Char at index 3: " + title.charAt(3));
10 System.out.println("Index of 'T': " + title.indexOf('T'));
11 System.out.println("Last index of 'a': " + title.lastIndexOf('a'));
12 System.out.println("Empty: " + title.isEmpty());
13 System.out.println("Ends with '.com': " + siteName.endsWith(".com"));
14 System.out.println("Equals: " + siteName.equals(title));
15 System.out.println("Sub-string: " + siteName.substring(9, 14));
16 System.out.println("Upper case: " + siteName.toUpperCase());
17 }
18
19 }
Console
<terminated> JavaStringExample [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (3 Feb 2020, 03:54:10)
Length of title: 14
Char at index 3: a
Index of 'T': 5
Last index of 'a': 11
Empty: false
Ends with '.com': true
Equals: false
Sub-string: smart
Upper case: WWW.BTECHSMARTCLASS.COM

String Buffer

- **String Buffer** class represents a thread-safe, mutable sequence of characters.
- A string buffer is like a String but it can be modified. At any point in time, it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- The methods of String Buffer class are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.
- The principal operations on a String Buffer instance are the append and insert methods, which are overloaded so as to accept data of any type. Each converts a given data to a string and then appends or inserts the characters of that string to the string buffer.
- The append method always adds these characters at the end of the buffer and the insert method adds the characters at a specified point.

Below are String Buffer class methods and descriptions. They can be used as an individual method or in combination. These methods are synchronized and can be used for a multi-threaded environment.

1. **append():** Appends the string representation of the specified argument to the specified String Buffer instance sequence.
2. **insert():** Inserts the string representation of the specified argument to the specified String Buffer instance sequence.

```
public class Main {

    public static void main(String[] args) {

        StringBuffer stringBuffer = new StringBuffer("Vikram");
        System.out.println(stringBuffer);
        stringBuffer.append(" Gupta");
        System.out.println(stringBuffer);
        stringBuffer.insert(7, "Binod ");
        System.out.println(stringBuffer);
    }
}
```

OUTPUT:

```
Vikram
Vikram Gupta
Vikram Binod Gupta
```

String Builder

- **String Builder** represents a mutable sequence of characters.
- The instance of this class does not guarantee Synchronization and hence should not be used in multi-threaded environments.
- Wherever possible we should always try to use String Builder instead of String Buffer as String Builder is faster than String Buffer under most implementations.
- Below are String Builder class methods and descriptions. They can be used as an individual method or in combination.

1. **append():** Appends the string representation of the specified argument to the specified String Builder instance sequence.
2. **insert():** Inserts the string representation of the specified argument to the specified String Builder instance sequence.

```java
public class Main {

    public static void main(String[] args) {

        StringBuilder stringBuilder = new StringBuilder("Vikram");
        System.out.println(stringBuilder);
        stringBuilder.append(" Gupta");
        System.out.println(stringBuilder);
        stringBuilder.insert(7," Binod");
        System.out.println(stringBuilder);
    }
}
```

OUTPUT:

```
Vikram
Vikram Gupta
Vikram Binod Gupta
```

StringBuffer	StringBuilder
1. Every method present in StringBuffer is synchronized.	1.No method present in StringBuilder is synchronized .
2. At a time only one thread is allow to operate on StringBuffer object. Hence StringBuffer object is Thread safe	2. At a time multiple threads are allow to operate on StringBuilder object and hence StringBuilder object is not Thread Safe.
3. It Increases waiting time of threads and hence relatively performance is low.	3. Threads are not required to wait to operate on StringBuilder object and hence relatively performance is high.
4. Introduced in 1.0 version	4. Introduced in 1.5 version

String Tokenizer

The StringTokenizer is a built-in class in java used to break a string into tokens. The StringTokenizer class is available inside the java.util package.

The StringTokenizer class object internally maintains a current position within the string to be tokenized.

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

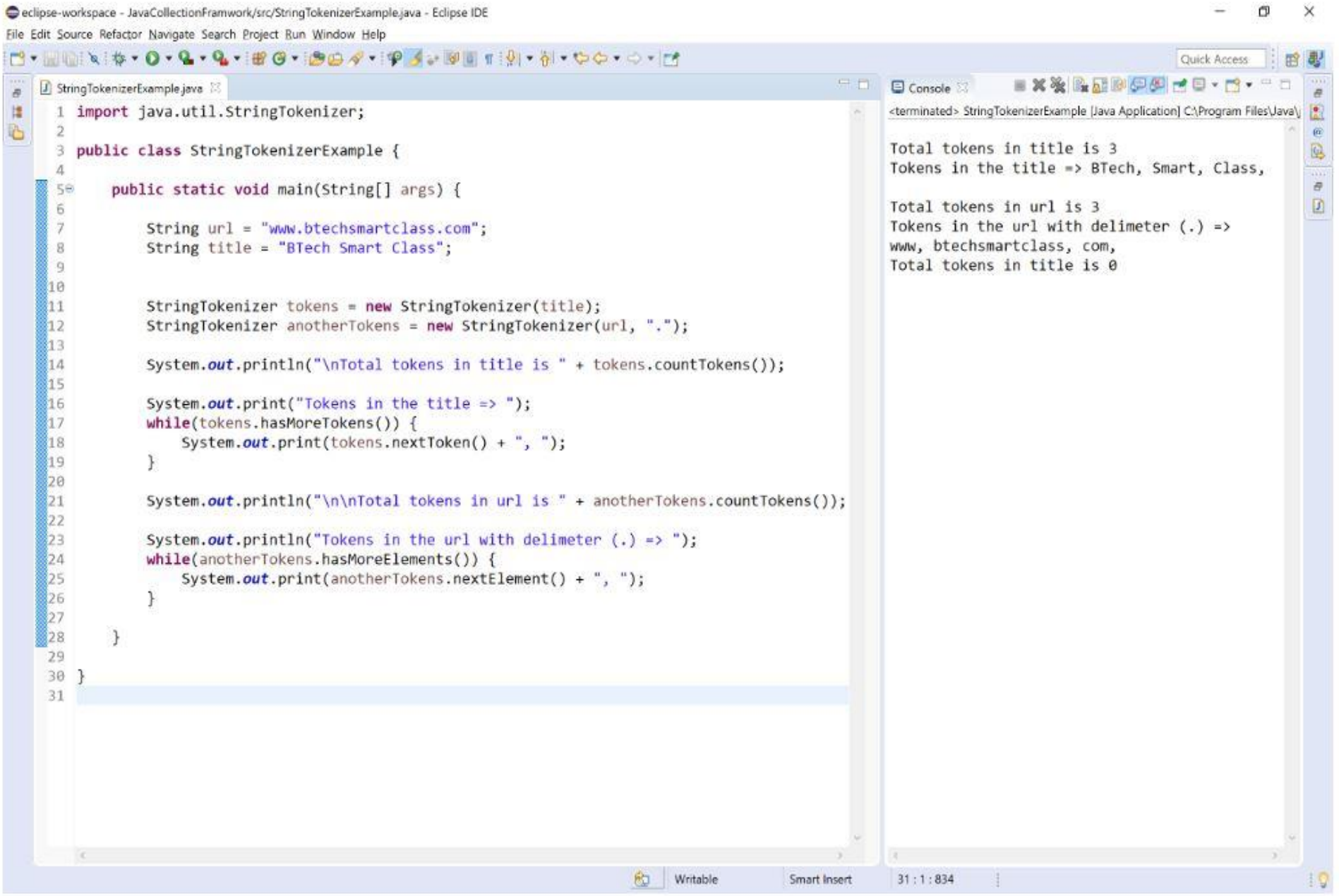
The StringTokenizer class in java has the following constructor.

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

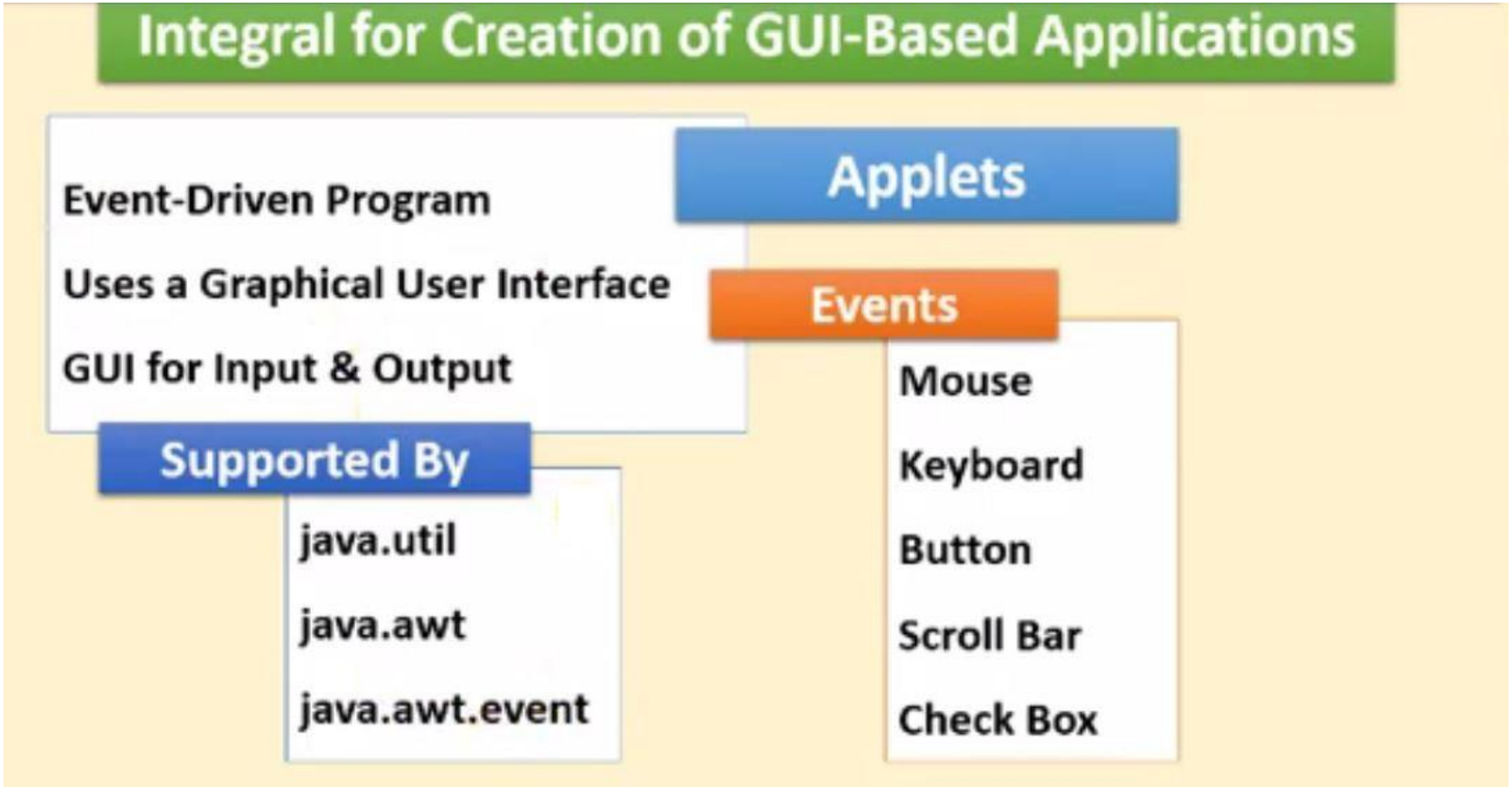
The StringTokenizer class in java has the following methods.

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

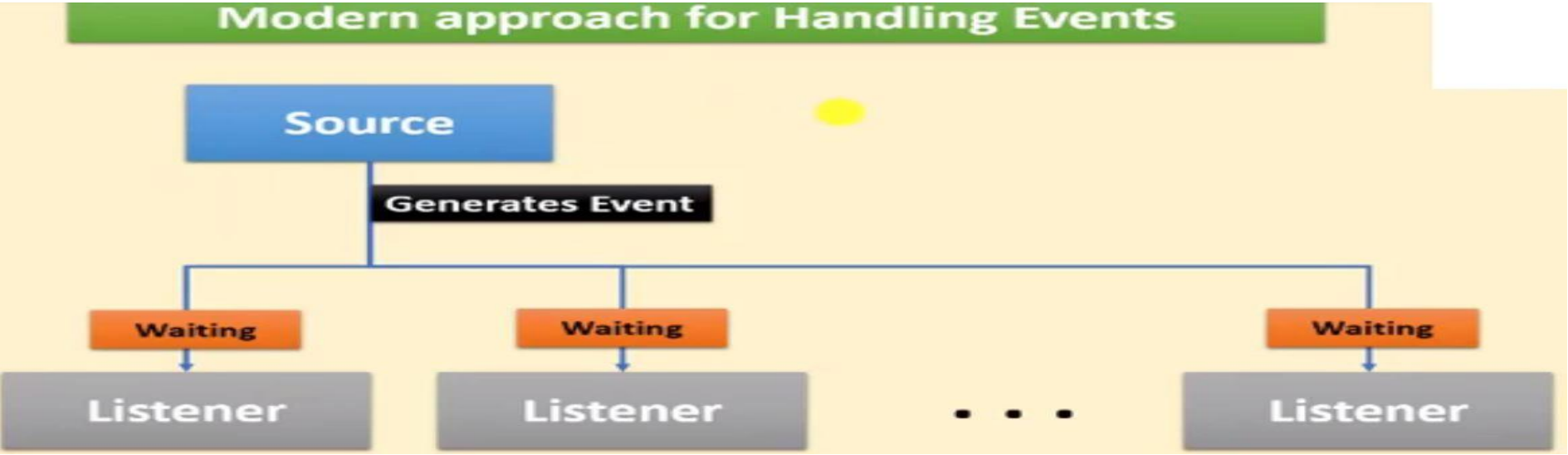
When we execute the above code, it produce the following output.

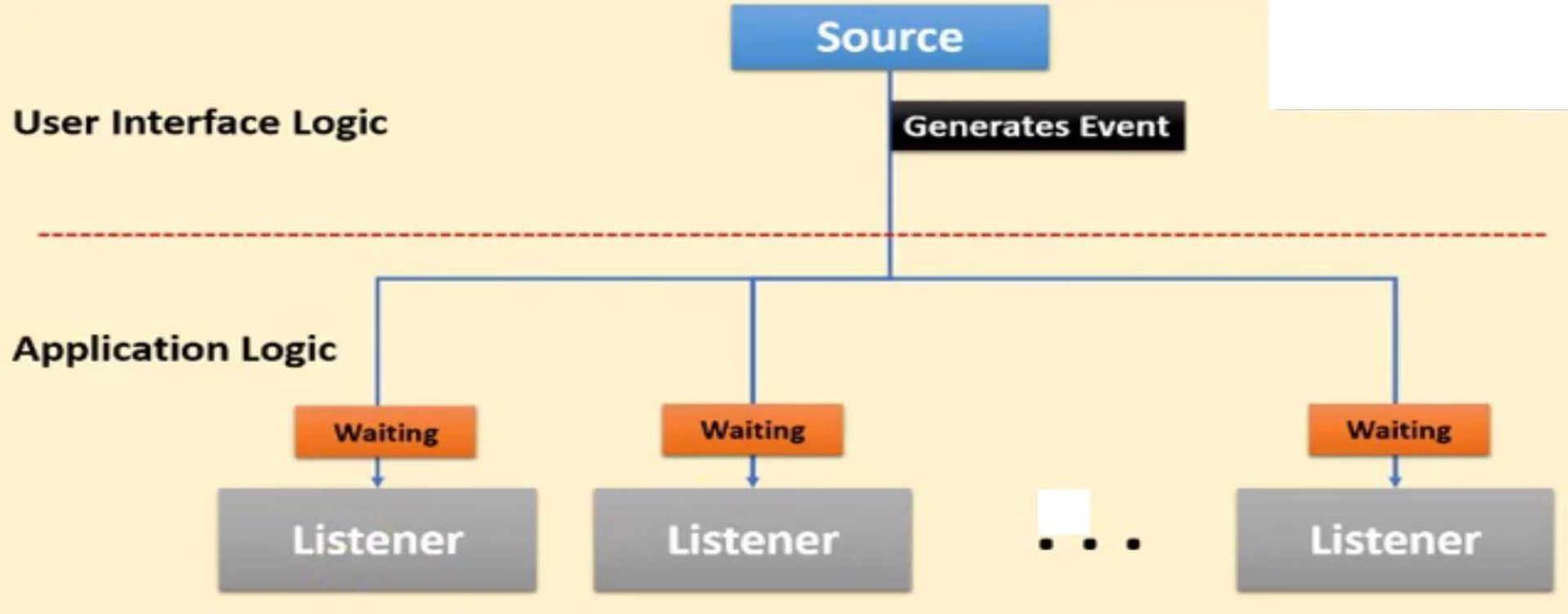


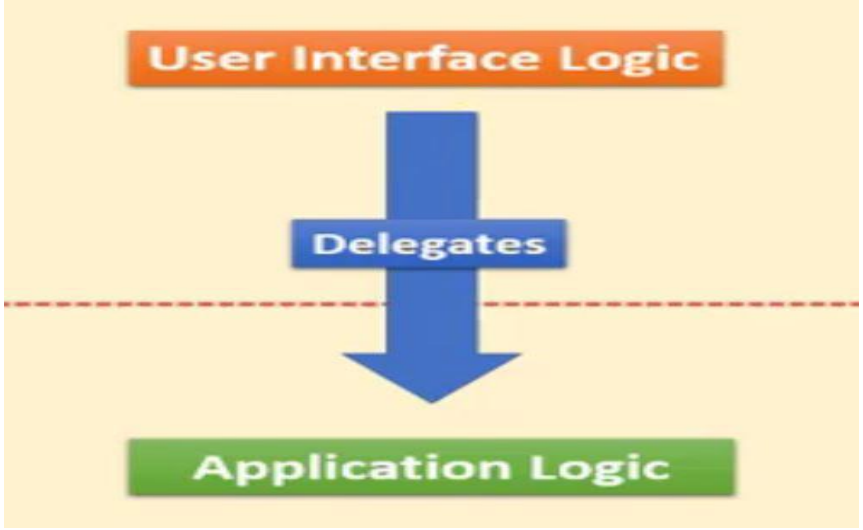
Event handling

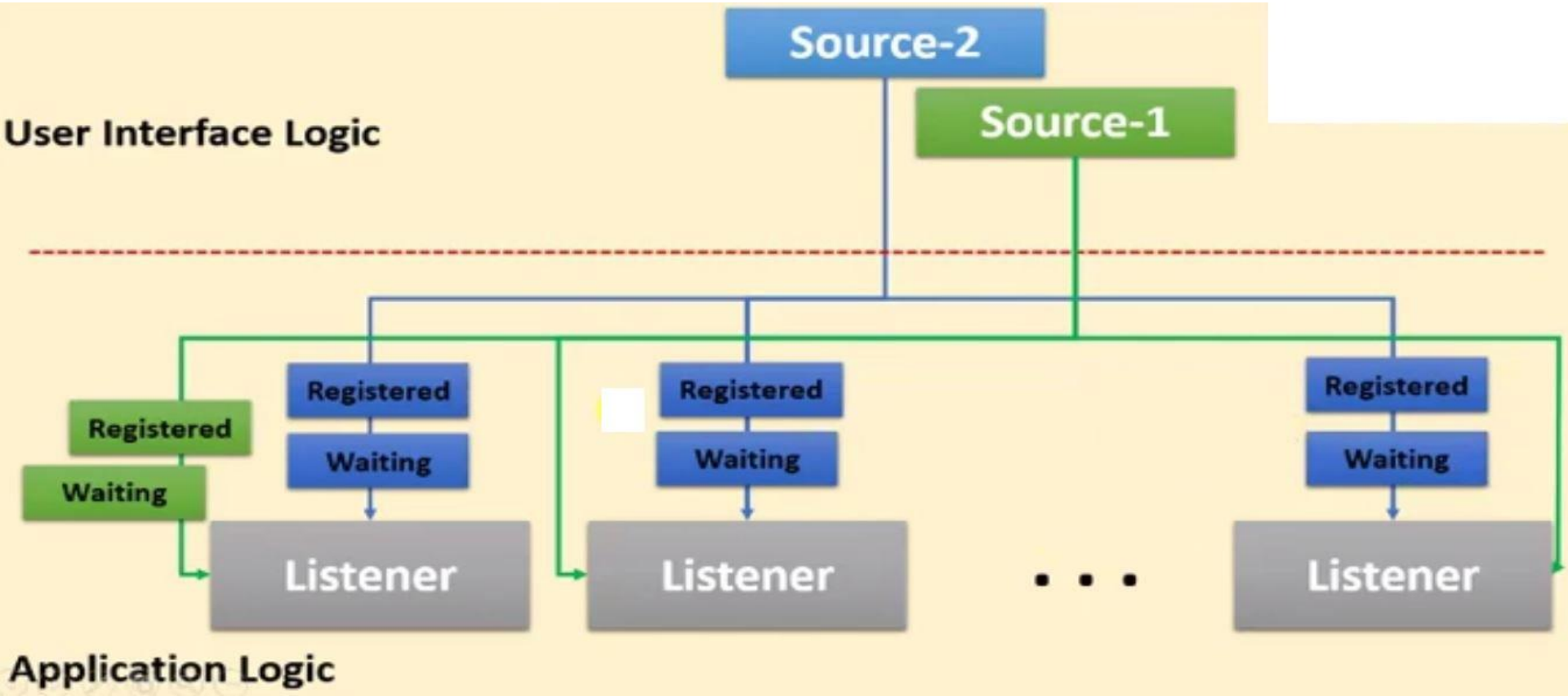


Delegation Event Model





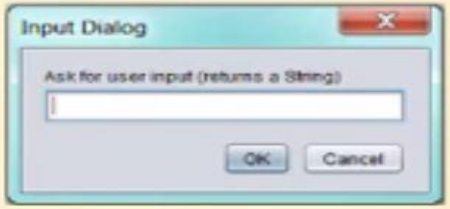

User Interface Logic
Delegates
Application Logic

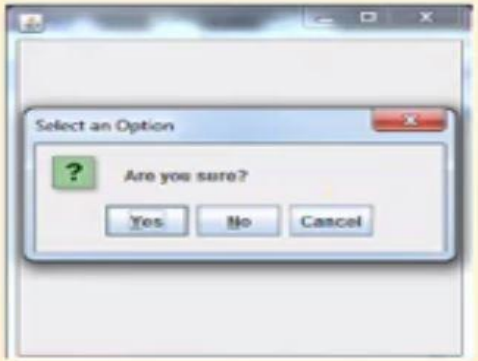

Source-2
Source-1
User Interface Logic
Registered
Waiting
Registered
Waiting
Registered
Waiting
Registered
Waiting
Listener
Listener
Listener
Application Logic

Events

Object that Describes a State Change in a Source

- Interaction by User
- Timer Expires
- Counter Exceeds
- Operation is Completed
- S/W | H/W Error





Event Sources

Object that Generates an Event

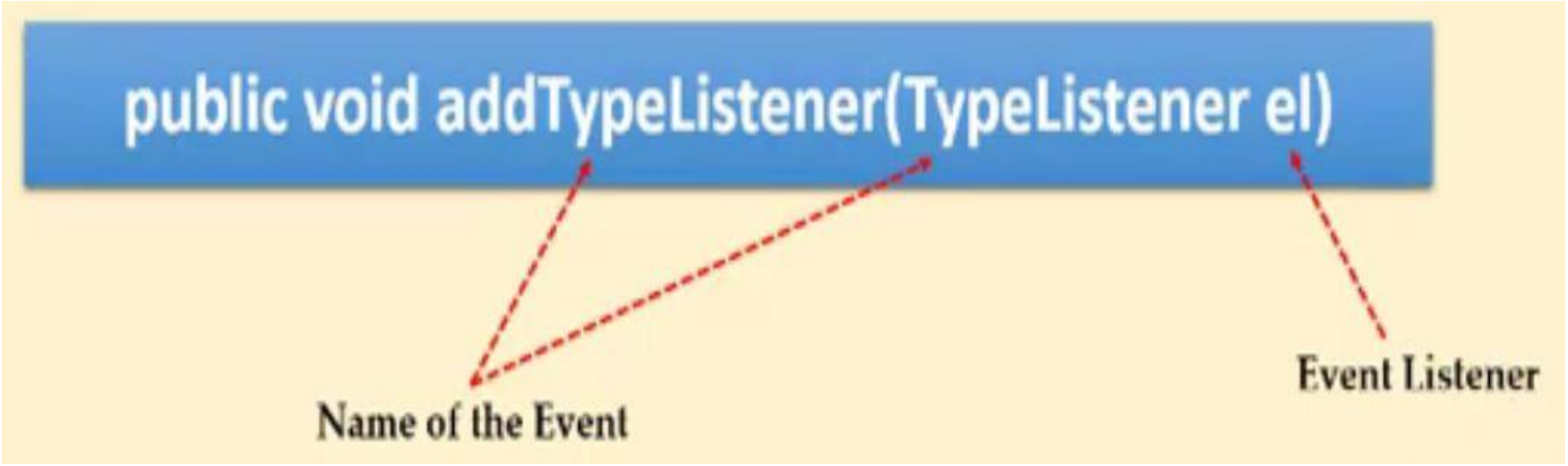
Occurs when Internal State of Object Changes

Sources May Generate more than One Type of Event

Source must Register Listeners

Registered Listeners will Receive Notifications of Specific Type of Event

Each Type of Event has its Own Registration Method



public void addTypeListener(TypeListener el)

addKeyListener( )

addMouseMotionListener( )

Events are Multicast

public void addTypeListener(TypeListener el)

Events are Multicast

public void addTypeListener(TypeListener el) throws java.util.TooManyListenersException

Event is now Unicast

public void removeTypeListener(TypeListener el)

Event Listeners

A Listener is an Object that is Notified when an Event Occurs

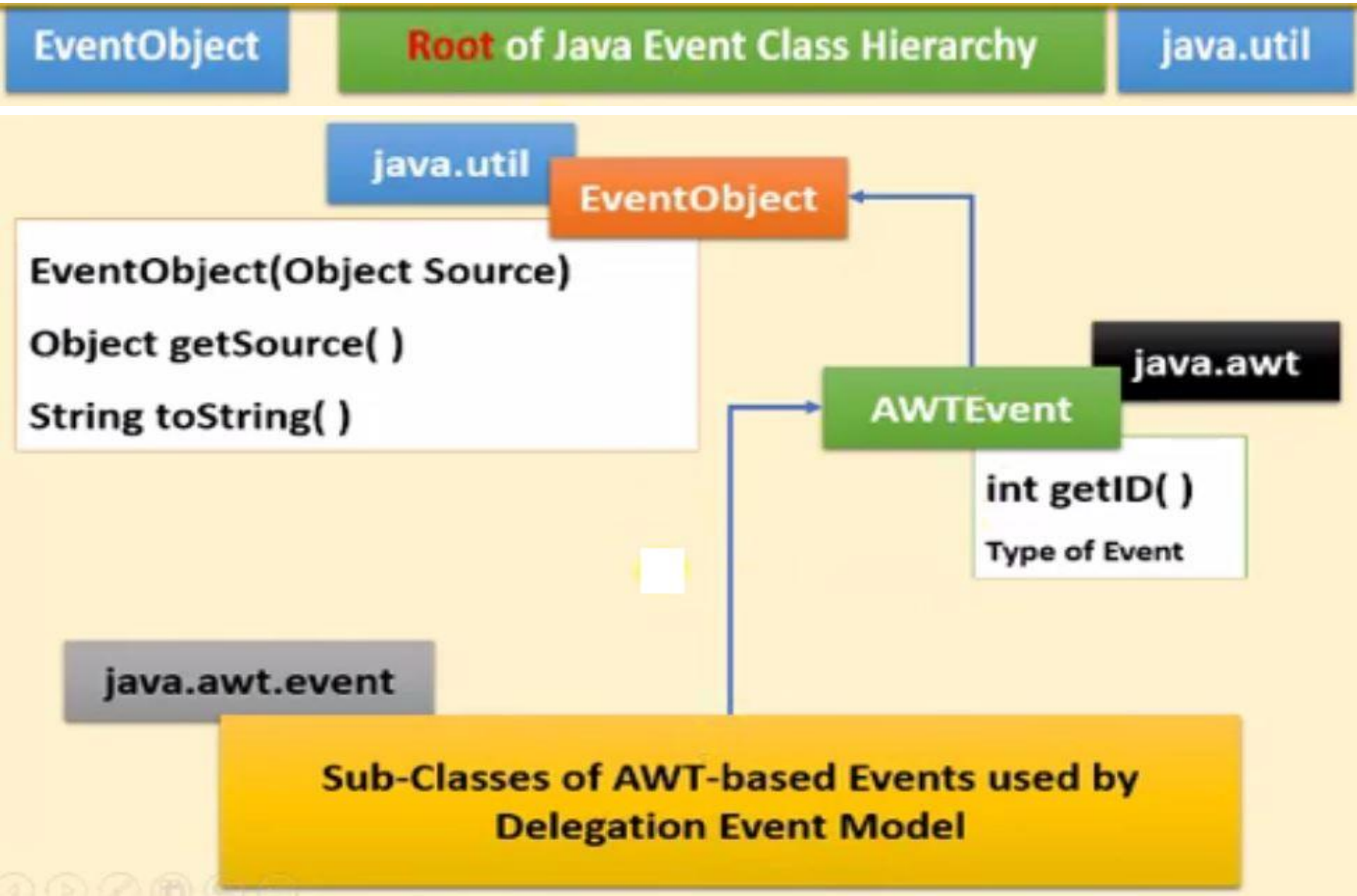
Should be Registered with One or More Sources

Must be Implemented to Receive & Process Notifications

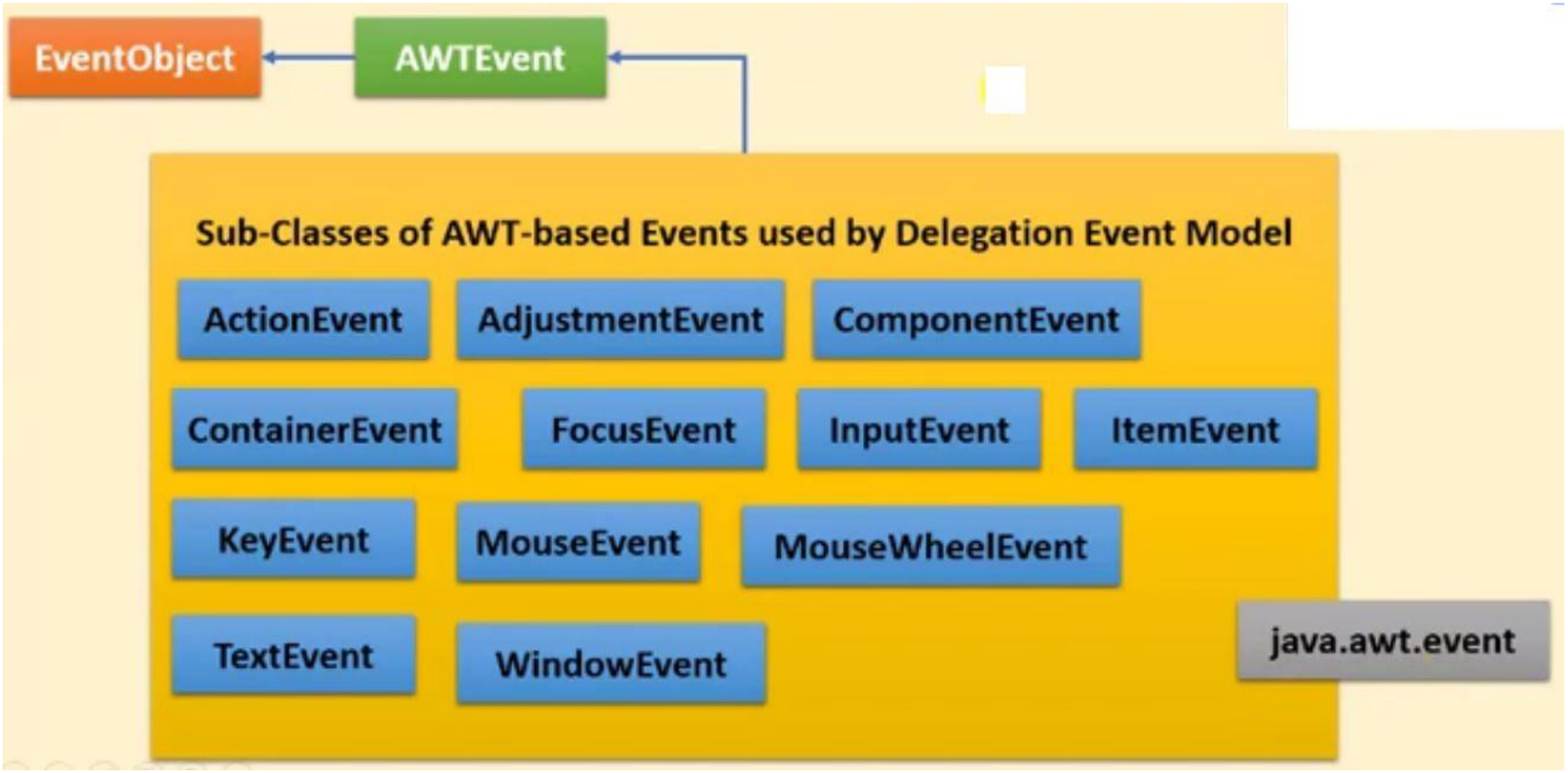
Methods that Receive & Process Events are defined in a Set of Interfaces

→ java.awt.event

Event Classes



Event Sub-Classes

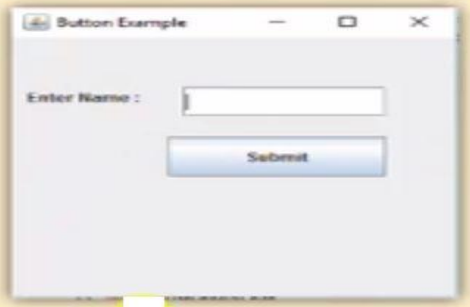


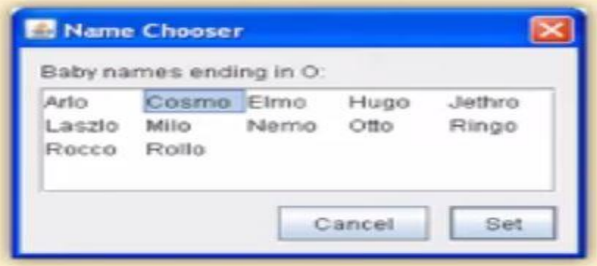
Action Event Class

Button is Pressed

List Item is Double Clicked

Menu Item is Selected





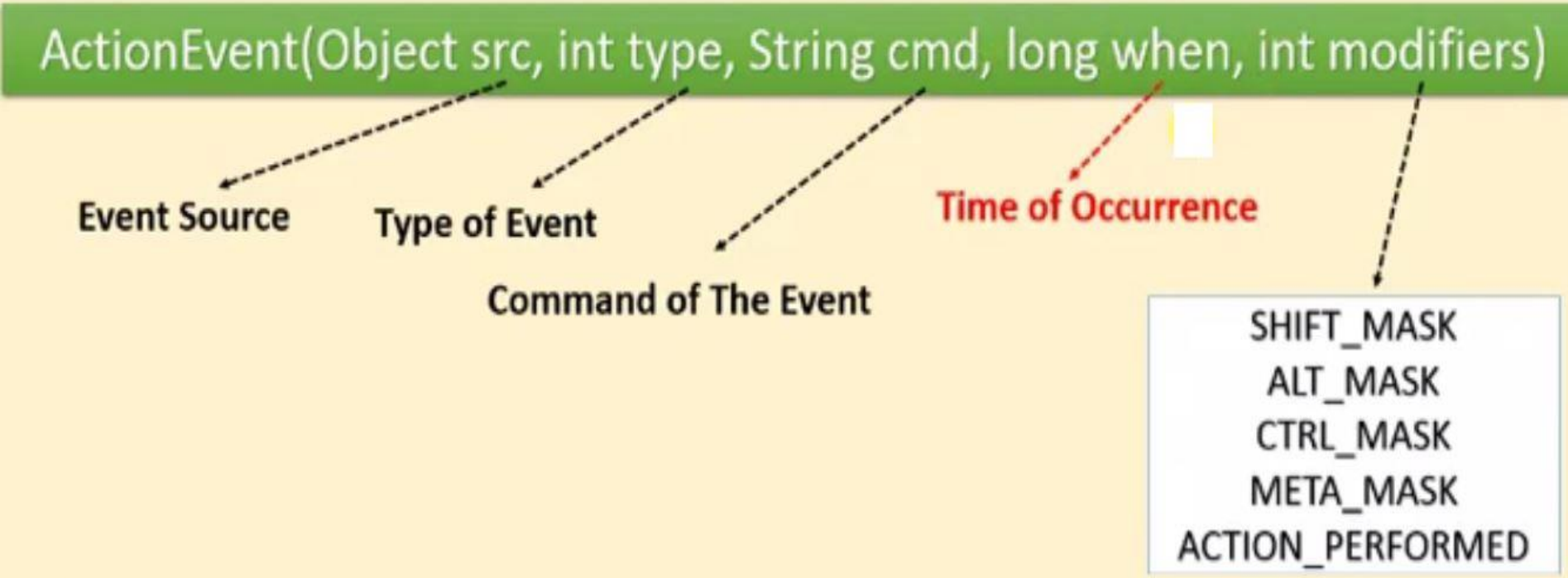
ActionEvent(Object src, int type, String cmd, int modifiers)

Event Source

Type of Event

Command of The Event

SHIFT_MASK
ALT_MASK
CTRL_MASK
META_MASK
ACTION_PERFORMED



ActionEvent(Object src, int type, String cmd, long when, int modifiers)

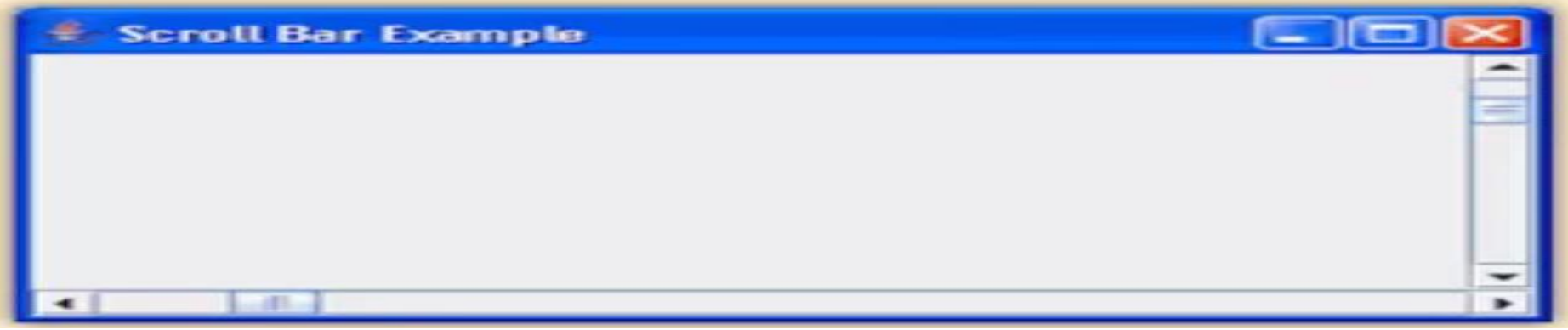
String getActionCommand( )

int getModifiers( )

long getWhen( )

Adjustment Event Class

Generated by a Scroll Bar



AdjustmentEvent(Adjustable src, int id, int type, int data)

- Event Source
- Event
- Type of Event
- Associated Data

Type of Event

BLOCK_DECREMENT
BLOCK_INCREMENT
TRACK
UNIT_DECREMENT
UNIT_INCREMENT

AdjustmentEvent(Adjustable src, int id, int type, int data)

Adjustable getAdjustable( )

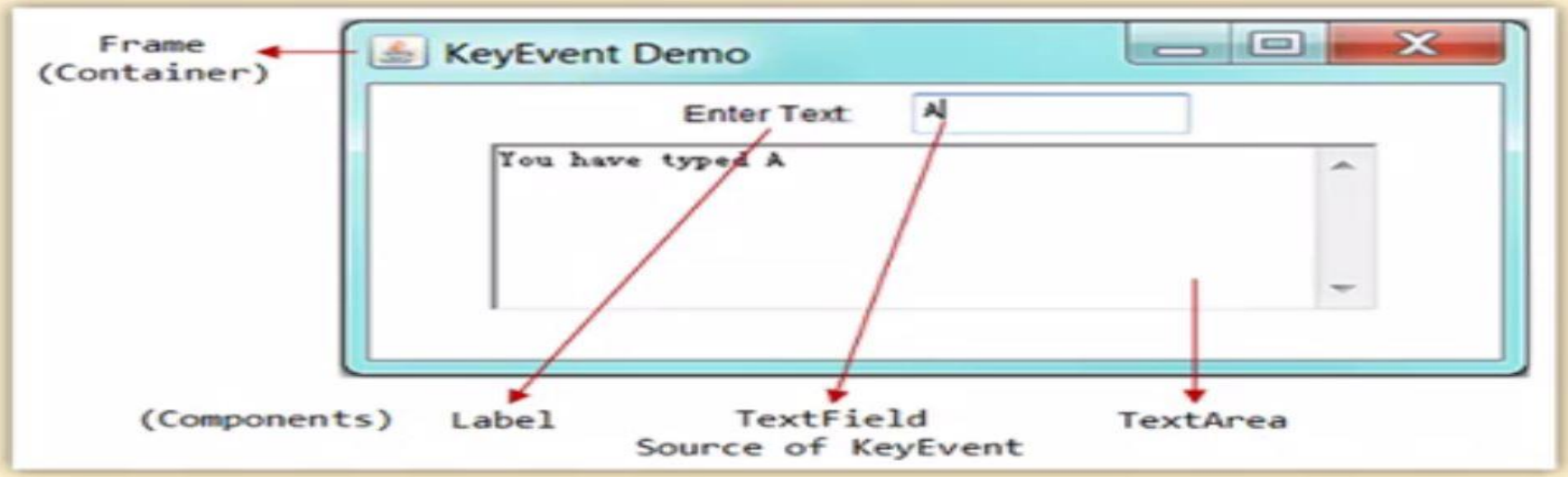
int getAdjustmentType( )

int getValue( )

Type of Event
BLOCK_DECREMENT
BLOCK_INCREMENT
TRACK
UNIT_DECREMENT
UNIT_INCREMENT

Component Event Class

When Size, Position, or Visibility of a Component is Changed

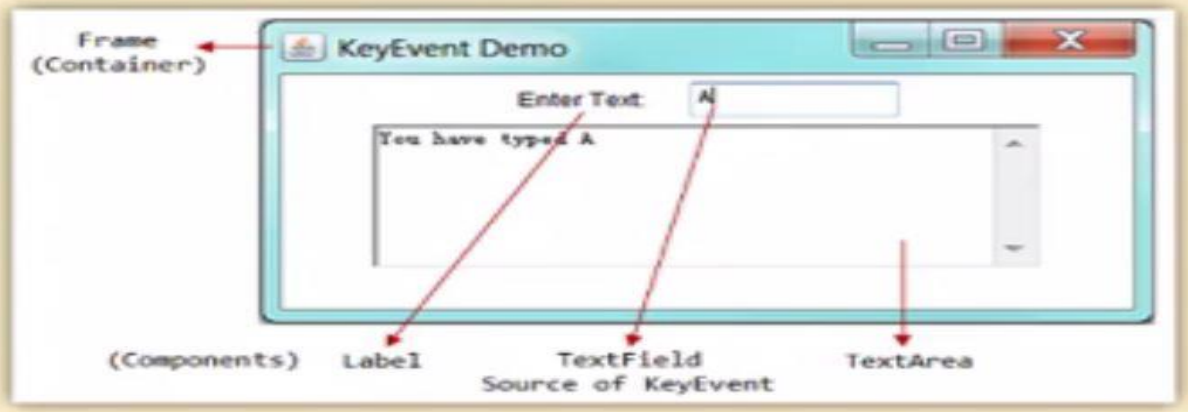


ComponentEvent(Component src, int type)

Event Source

Type of Event

COMPONENT_HIDDEN
COMPONENT_SHOWN
COMPONENT_MOVED
COMPONENT_RESIZED



Component getComponent( )

Super-Class for

ContainerEvent

FocusEvent

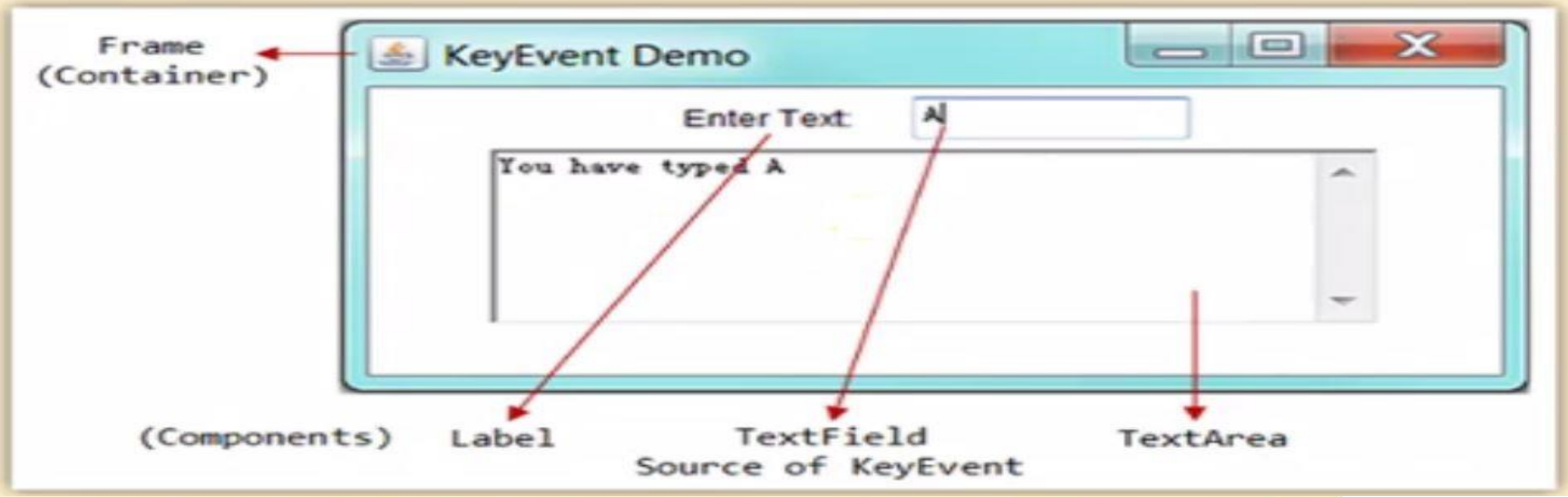
KeyEvent

MouseEvent

WindowEvent

Container Event Class

When a Component is Added to or Removed from a Container



ContainerEvent(Component src, int type, Component comp)

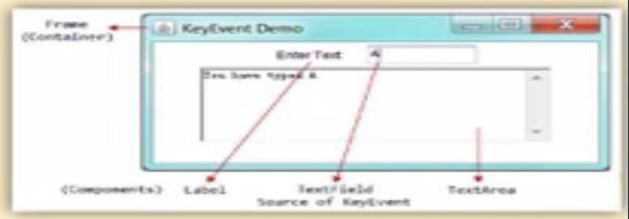
Event Source

Type of Event

COMPONENT_ADDED
COMPONENT_REMOVED

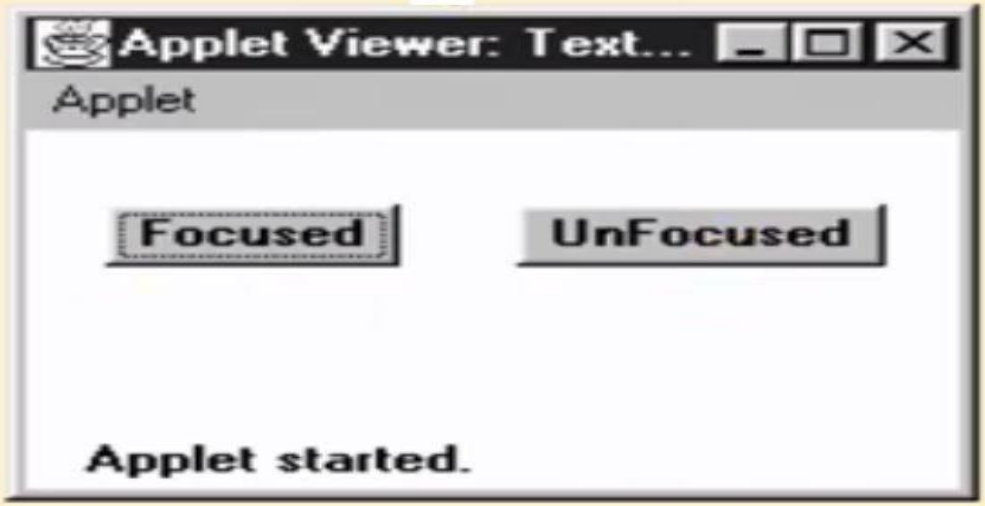
Container getContainer( )

Component getChild( )



Focus Event Class

When a Component Gains or Loses Input Focus

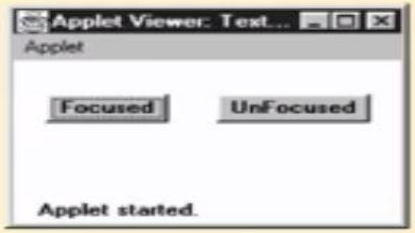


When a Component is Added to or Removed from a Container

ComponentEvent Class

FocusEvent Class

ContainerEvent Class



FocusEvent(Component src, int type, boolean temporaryFlag, Component other)

Event Source

Temporary Focus or Not

Opposite Component

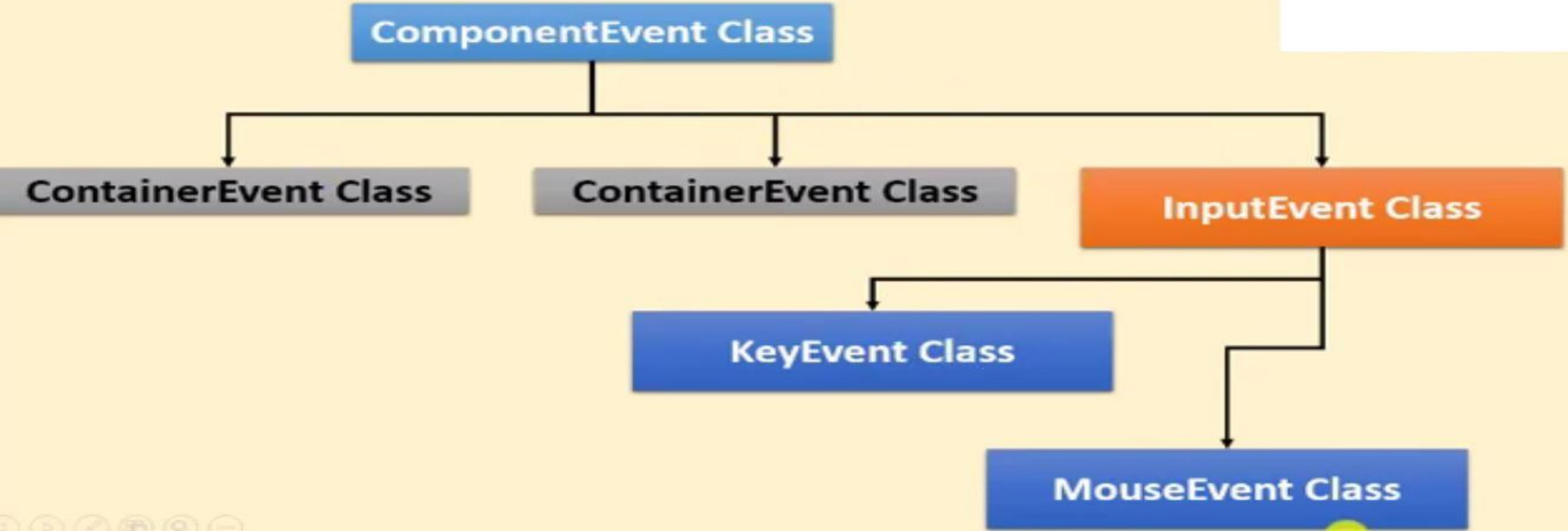
Type of Event

FOCUS_GAINED
FOCUS_LOST

Component getOppositeComponent( )

boolean isTemporary( )

Input Event Class



KeyEvent Class

MouseEvent Class

Modifiers

ALT_MASK
SHIFT_MASK
CTRL_MASK
META_MASK
ALT_GRAPH_MASK
BUTTON2_MASK
BUTTON3_MASK
BUTTON1_MASK

KeyEvent Class

MouseEvent Class

Original Modifiers

ALT_MASK
SHIFT_MASK
CTRL_MASK
META_MASK
ALT_GRAPH_MASK
BUTTON1_MASK
BUTTON2_MASK
BUTTON3_MASK

Extended Modifiers

ALT_DOWN_MASK
SHIFT_ DOWN_ MASK
CTRL_ DOWN_ MASK
META_ DOWN_ MASK
ALT_GRAPH_ DOWN_ MASK
BUTTON1_ DOWN_ MASK
BUTTON2_ DOWN_ MASK
BUTTON3_ DOWN_ MASK

Original Modifiers
ALT_MASK
SHIFT_MASK
CTRL_MASK
META_MASK
ALT_GRAPH_MASK
BUTTON1_MASK
BUTTON2_MASK
BUTTON3_MASK

Extended Modifiers
ALT_DOWN_MASK
SHIFT_ DOWN_ MASK
CTRL_ DOWN_ MASK
META_ DOWN_ MASK
ALT_GRAPH_ DOWN_ MASK
BUTTON1_ DOWN_ MASK
BUTTON2_ DOWN_ MASK
BUTTON3_ DOWN_ MASK

int getModifiers( )

int getModifiersEx( )

boolean isAltDown( )

boolean isAltGraphDown( )

boolean isControlDown( )

boolean isMetaDown( )

boolean isShiftDown( )

Item Event Class

Check Box | List Item

ItemEvent(ItemSelectable src, int type, Object entry, int state)

- Event Source
- Type of Event
  - SELECTED
  - DESELECTED
  - ITEM_STATE_CHANGED
- Specific Item Passes
- Current State of the Item

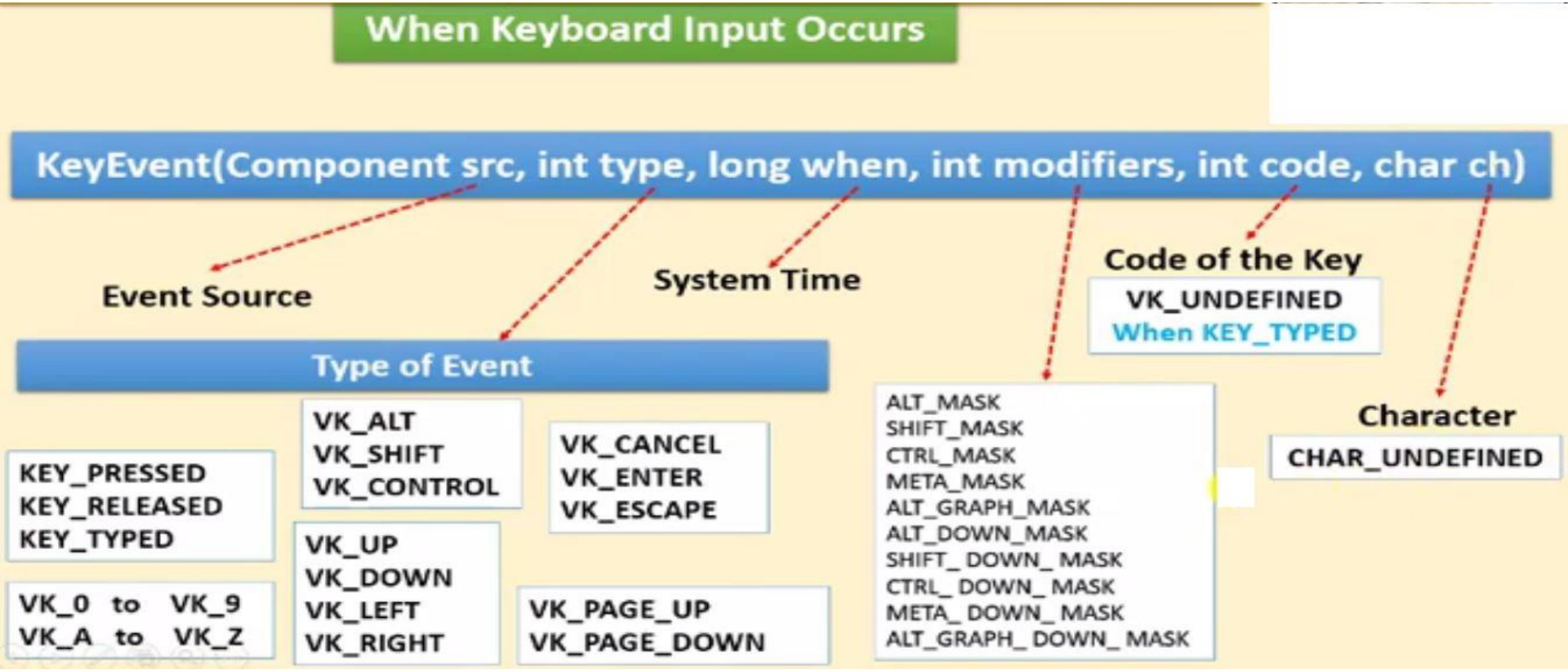
Object getItem( )

int getStateChange( )

ItemSelectable getItemSelectable( )

- Selectable Item
  - Lists
  - Choices

Key Event Class



VK_UNDEFINED

int getKeyCode( )

CHAR_UNDEFINED

char getKeyChar( )

Mouse Event Class

When Mouse Input Occurs

`MouseEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup)`

- Event Source
- Type of Event
  - MOUSE_PRESSED
  - MOUSE_RELEASED
  - MOUSE_CLICKED
  - MOUSE_MOVED
  - MOUSE_DRAGGED
  - MOUSE_WHEEL
  - MOUSE_ENTERED
  - MOUSE_EXITED
- System Time
- BUTTON1_MASK
  BUTTON2_MASK
  BUTTON3_MASK
  BUTTON1_DOWN_MASK
  BUTTON2_DOWN_MASK
  BUTTON3_DOWN_MASK
- Click Count
- If Popup Menu Appears

`int getX( )` `int getY( )`

`Point getPoint( )`

`void translatePoint(int x, int y)`

`int getClickCount( )`

`boolean isPopupTrigger( )`

`int getButton( )`

- NOBUTTON
- BUTTON1
- BUTTON2
- BUTTON3

`MouseEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup)`

int getX( )

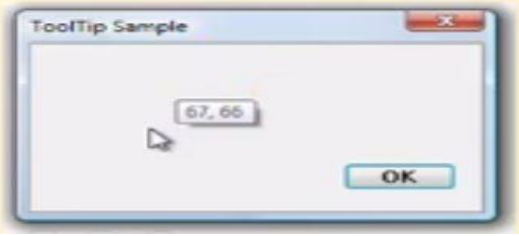
int getY( )

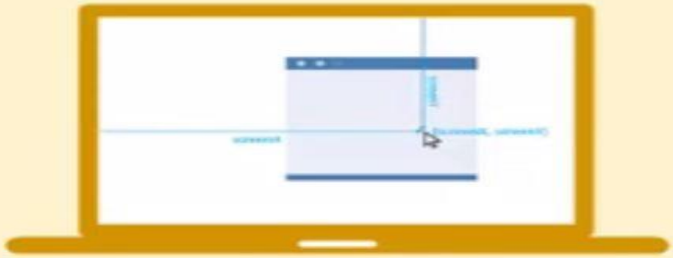
Point getPoint( )

int getXOnScreen( )

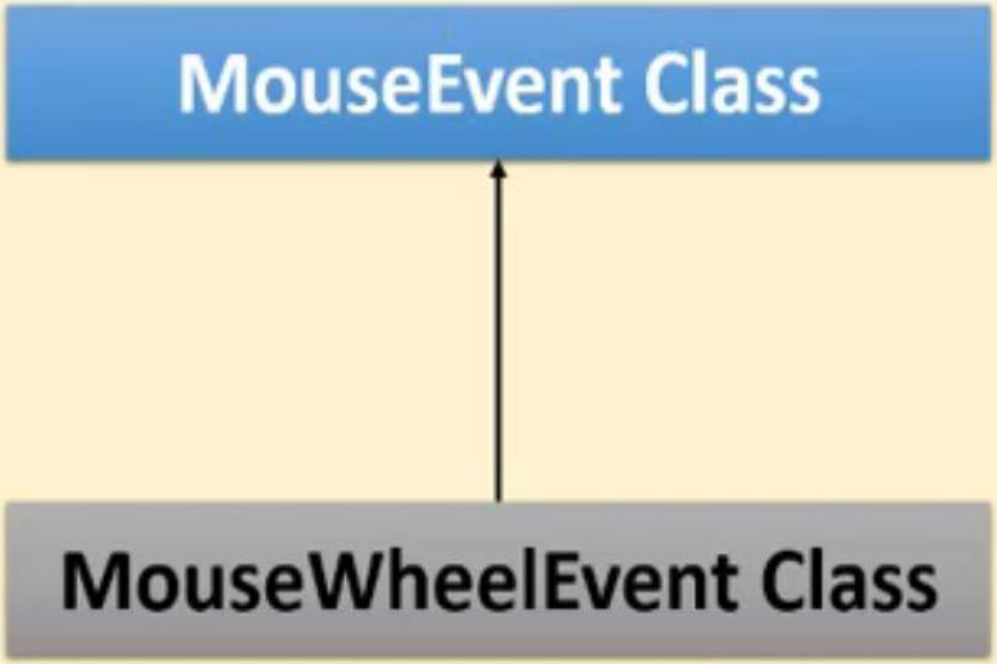
int getYOnScreen( )

Point getLocationOnScreen( )





Mouse Wheel Event Class



MouseWheelEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, Boolean triggersPopup, int scrollHow, int amount, int count)

Number of Rotation

Number of Units to Scroll

WHEEL_BLOCK_SCROLL
WHEEL_UNIT_SCROLL

MouseWheelEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, Boolean triggersPopup, int scrollHow, int amount, int count)

WHEEL_BLOCK_SCROLL
WHEEL_UNIT_SCROLL

Number of Units to Scroll

Number of Rotation

int getScrollType( )

int getWheelRotation( )

+ve

-ve

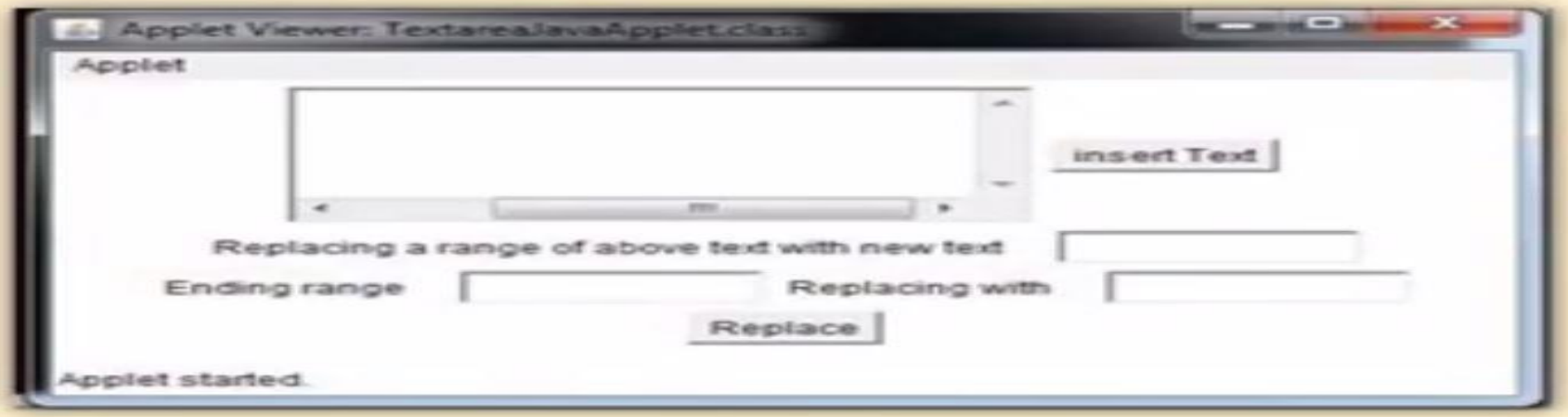
Clockwise

CounterClockwise

int getScrollAmount( )

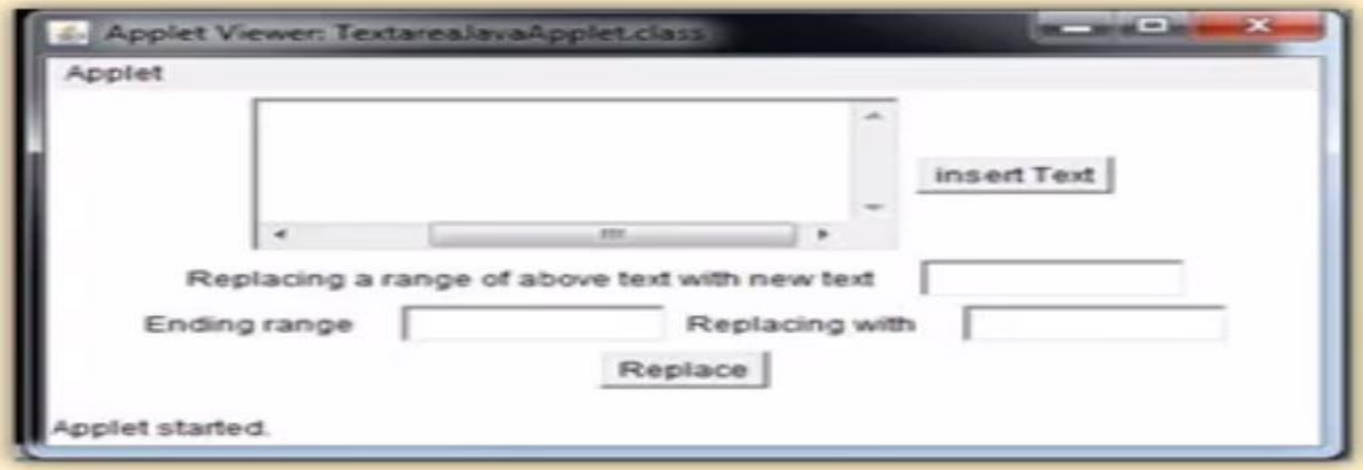
Text Event Class

Generated by Text Fields & Text Areas



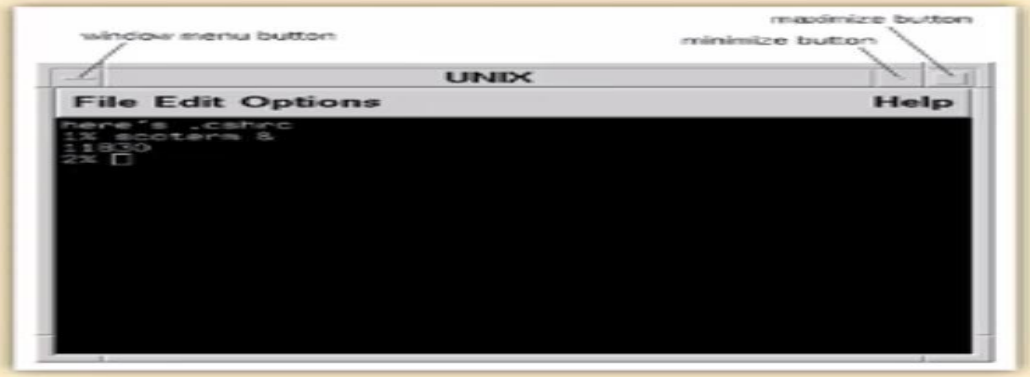
TextEvent(Object src, int type)

TEXT_VALUE_CHANGED



Window Event Class

WINDOW_OPENED
WINDOW_CLOSING
WINDOW_CLOSED
WINDOW_ACTIVATED
WINDOW_DEACTIVATED
WINDOW_GAINED_FOCUS
WINDOW_LOST_FOCUS
WINDOW_ICONIFIED
WINDOW_DEICONIFIED
WINDOW_STATE_CHANGED



WindowEvent(Window src, int type)

Event Source

Type of Event

WINDOW_OPENED
WINDOW_CLOSING
WINDOW_CLOSED
WINDOW_ACTIVATED
WINDOW_DEACTIVATED
WINDOW_GAINED_FOCUS
WINDOW_LOST_FOCUS
WINDOW_ICONIFIED
WINDOW_DEICONIFIED
WINDOW_STATE_CHANGED

WindowEvent(Window src, int type, Window other)

Opposite Window

WindowEvent(Window src, int type, int fromState, int toState)

Prior State

New State

WindowEvent(Window src, int type, Window other, int fromState, int toState)

Window getWindow( )

Window getOppositeWindow( )

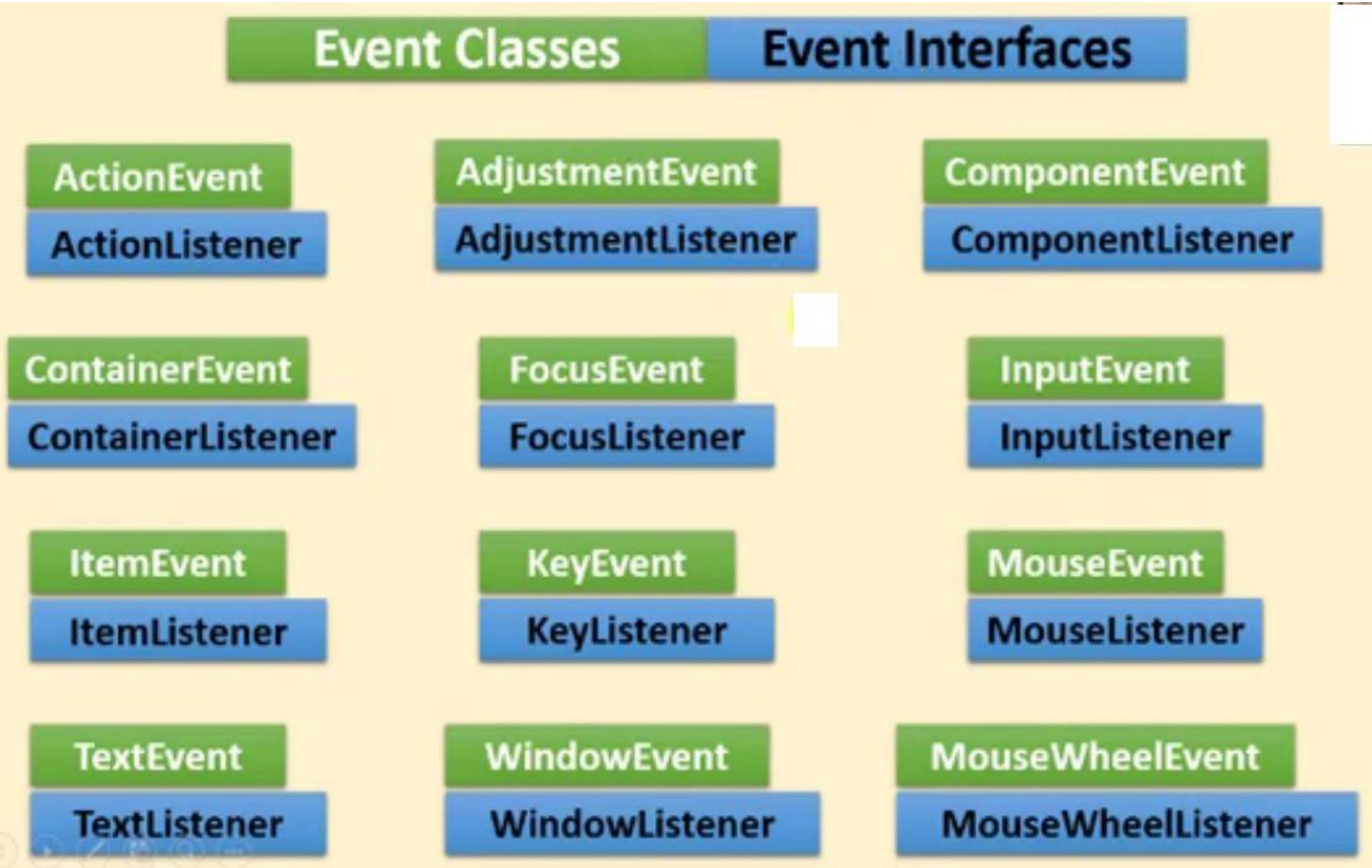
int getOldState( )

int getNewState( )

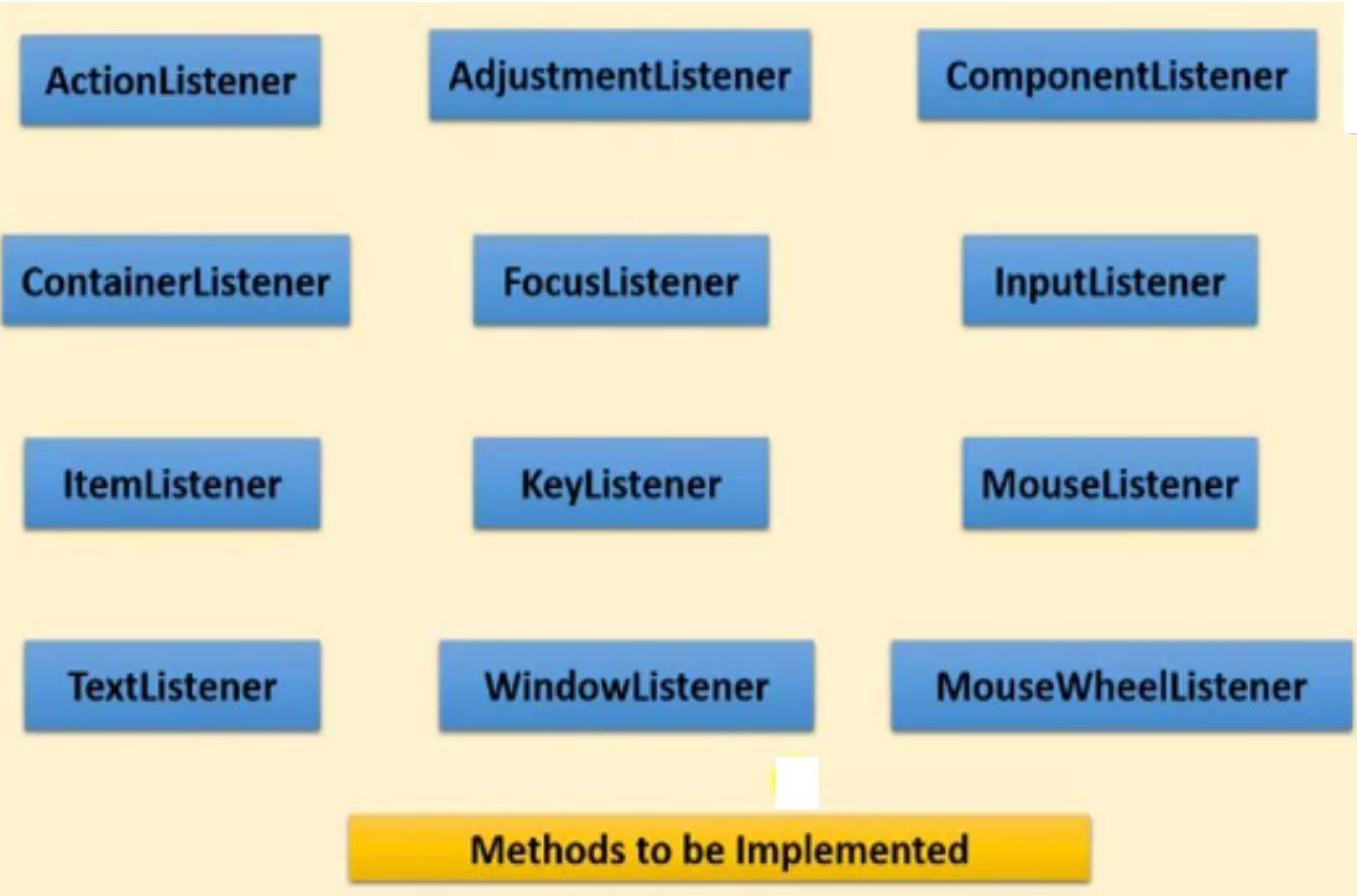
Delegation Event Model (2 Part)

Event Listener Interfaces

Event Listener Interfaces



Event Listener Interfaces



Action Listener Interface

```
void actionPerformed(ActionEvent ae)
```

Adjustment Listener Interface

```
void adjustmentValueChanged(AdjustmentEvent ae)
```

Component Listener Interface

```
void componentResized(ComponentEvent ce)
void componentMoved(ComponentEvent ce)
void componentShown(ComponentEvent ce)
void componentHidden(ComponentEvent ce)
```

Container Listener Interface

```
void componentAdded(ContainerEvent ce)
void componentRemoved(ContainerEvent ce)
```

Focus Listener Interface

```
void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)
```

Item Listener Interface

```
void itemStateChanged(ItemEvent ie)
```

Key Listener Interface

```
void keyPressed(KeyEvent ke)
void keyReleased(KeyEvent ke)
void keyTyped(KeyEvent ke)
```

Mouse Listener Interface

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

Mouse Wheel Listener Interface

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

Text Listener Interface

```
void textChanged(TextEvent te)
```

Window Listener Interface

```
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)
```

Window Focus Listener Interface

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

Adapter Classes

Implementing only those Events in which you are Interested

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

- Java adapter classes *provide the default implementation of listener interfaces*.
- If we want to handle few events, then we use adapter classes.
  Ex: Handling only mouse pressed and mouse released events.
- The adapter classes are found in **java.awt.event** package.
- The Adapter classes with their corresponding listener interfaces are given below.

Listener interface	Adapter class
MouseListener	Mouse Adapter
KeyListener	Key Adapter

Note: An Applet runs inside a web browser. Frame is an application program which don't run inside a browser.

Program to handle only mouse pressed and mouse released events using Adapter classes

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class madapter extends MouseAdapter
{
   Frame f;
   madapter()
   {
        f=new Frame();
        f.addMouseListener(this);
        f.setSize(400,400);
        f.setVisible(true);
   }
   public  void mouseClicked(MouseEvent e)
   {
        f.setBackground(Color.red);
   }
   public static void main(String args[ ])
   {
        madapter m1=new madapter();
   }
}
```

Inner Class

- **inner class** is a class which is declared inside the class.
- **Syntax**

```
class outer
{
        class inner
        {
           ..............
        }
}
```

Ex: Program to handle mouse events using inner class

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class innerclass extends Applet
{
    public void init()
    {
        addMouseListener(new madapter());
    }
    public class madapter extends MouseAdapter
    {
        public  void mouseClicked(MouseEvent e)
        {
            setBackground(Color.red);
        }
    }
}
/*
<html>
    <applet code="innerclass.class" width="300" height="300"></applet>
</html>
*/
```

Anonymous Inner Classes

- An inner class without name is called Anonymous Inner Class

Ex:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ainnerclass extends Applet
{
    public void init()
    {
        addMouseListener(new MouseAdapter()
                {
                    public  void mouseClicked(MouseEvent e)
                            {
                        setBackground(Color.red);
                            }
                });
    }
}
/*
<html>
    <applet code="ainnerclass.class" width="300" height="300"></applet>
</html>
*/
```

AWT-(Abstract Window Toolkit)

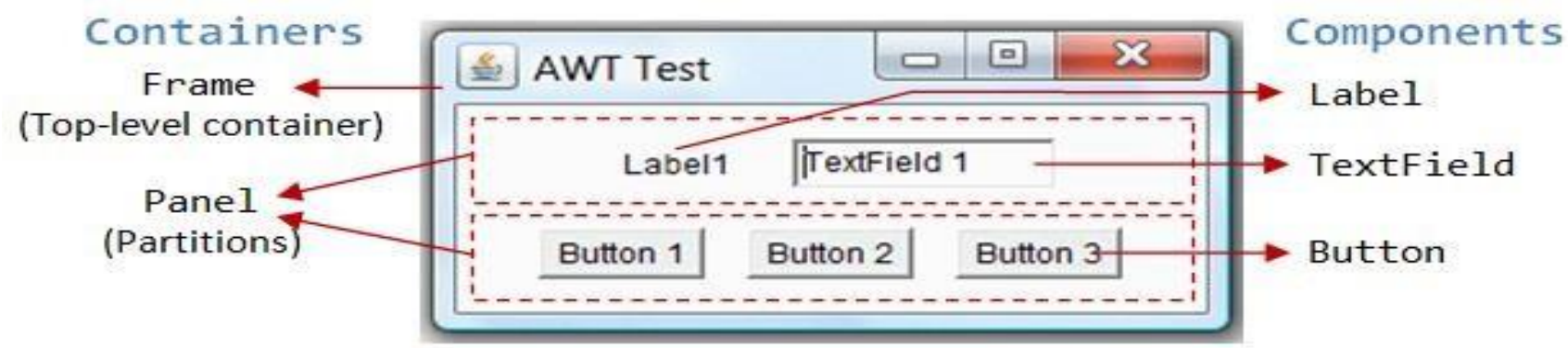
- **AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

- **AWT** (Abstract Window Toolkit) API consists 12 packages along with 370 classes. Fortunately, only 2 packages - **java.awt** and **java.awt.event** - are commonly-used.

- The **java.awt** package provides various classes such as TextField, Label, TextArea, CheckBox, Choice, List etc.

- The **java.awt.event** package provides Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent.

- **AWT Containers and Components**



Container:

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.
- The Container class extends Frame and Panel.

Frame:

- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Panel:

- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Component:

- Component is an abstract class that contains various
- classes such as Button, Label, Checkbox, TextField, Menu and etc.

Useful Methods of Component class

Method	Description
add(Component c)	inserts a component on this component.
setSize(int width,int height)	sets the size (width and height) of the component.
setLayout(LayoutManager m)	defines the layout manager for the component.
setVisible(boolean status)	changes the visibility of the component, by default false.
setTitle(String text)	Sets the title for component

→ **A Simple AWT Application:**

- You need a **frame**. There are two ways to create a **frame** in AWT.
- **By extending Frame class** (inheritance)

Ex:
```
class Example extends Frame
{
  ……..
}
```

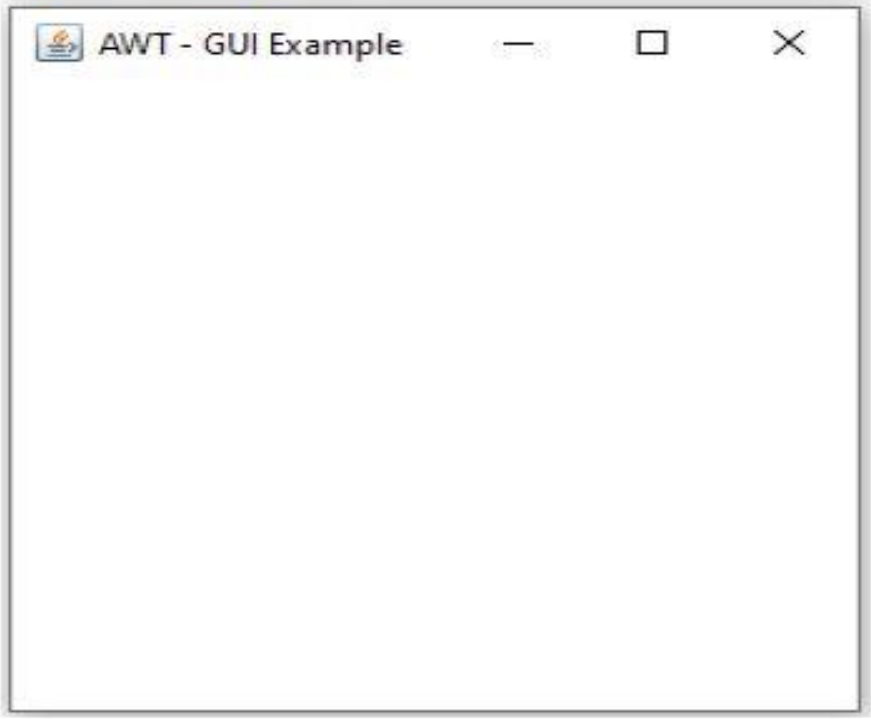
- **By creating the object of Frame class** (association)

Ex:
```
class Example
{
Frame obj=new Frame();
  ……..
}
```

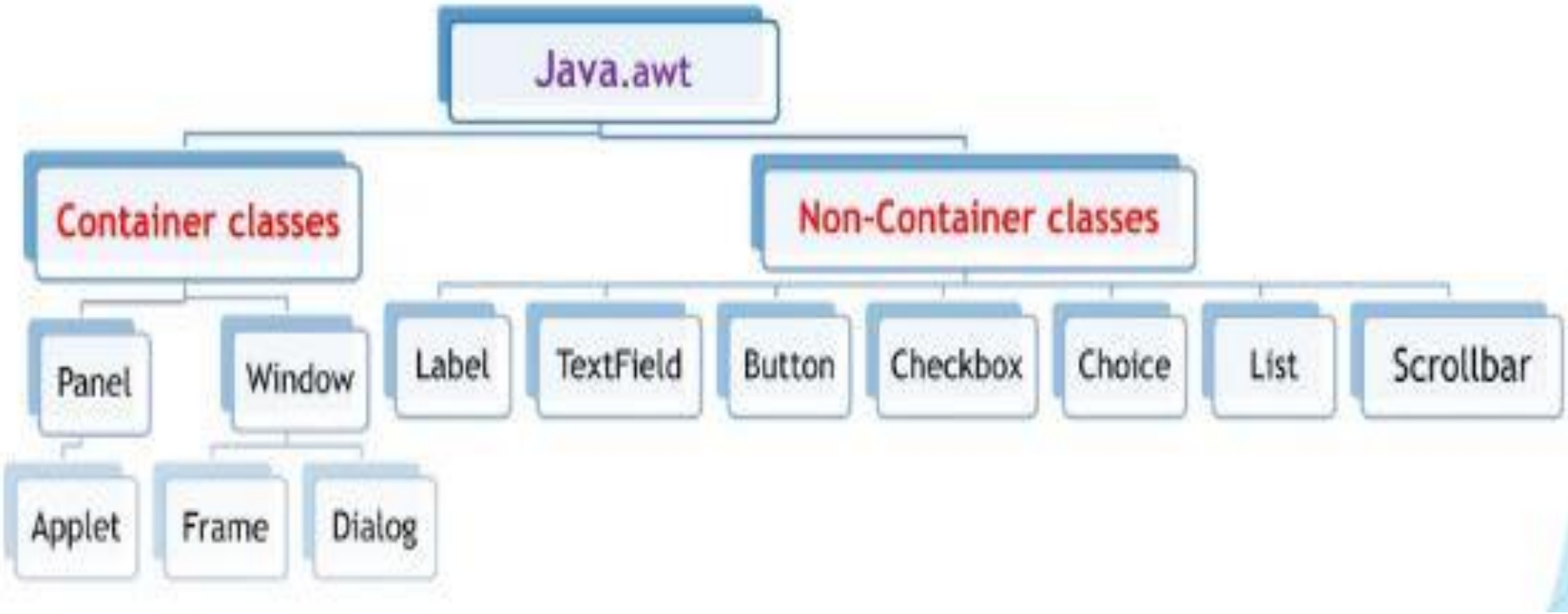
Example:

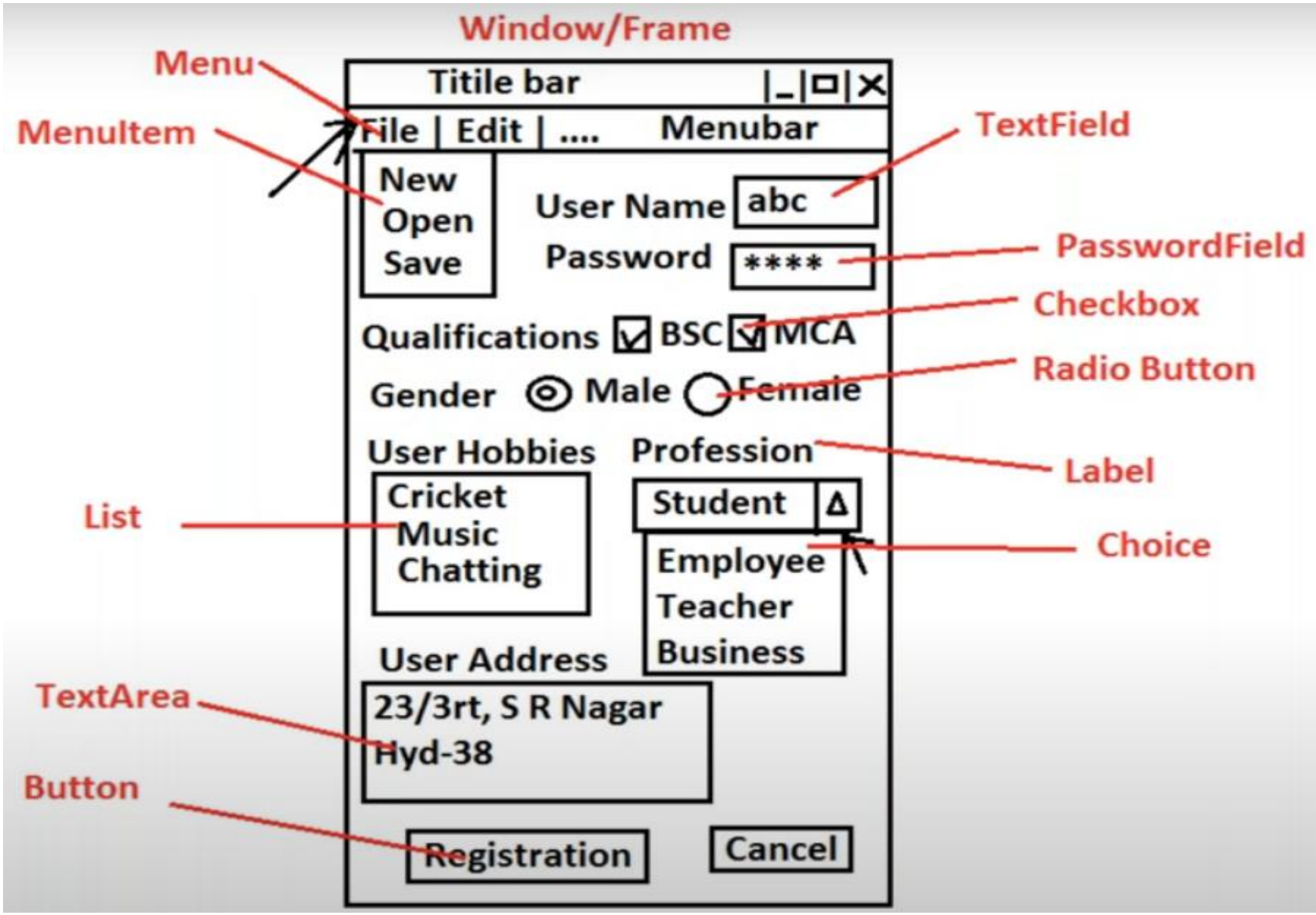
SimpleGUIExample.java

```
import java.awt.*;
public class SimpleGUIExample {
public static void main(String[] args) {
      // To create frame
      Frame f=new Frame("AWT - GUI Example");
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
      }
}
```



AWT Components or Elements

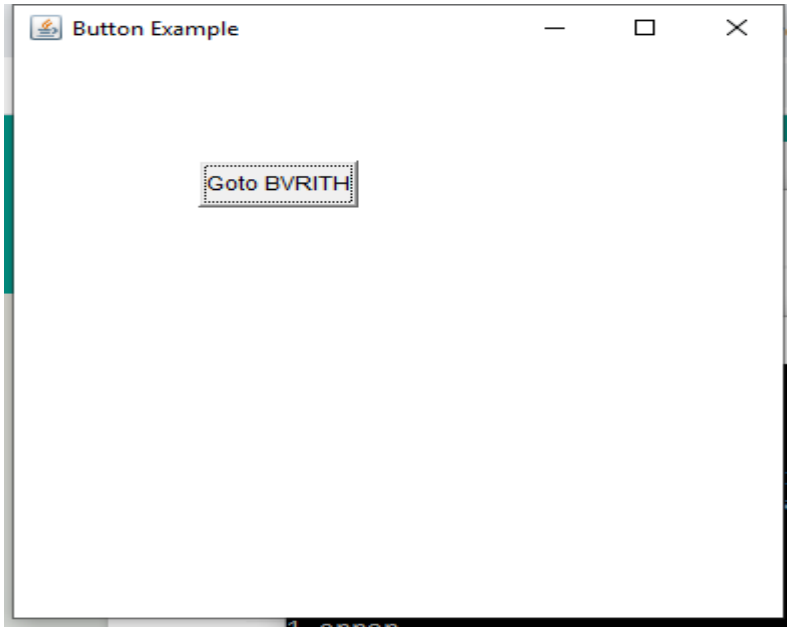



Window/Frame
Menu
MenuItem
Titile bar
File | Edit | .... Menubar
New
Open
Save
User Name
abc
TextField
Password
****
PasswordField
Qualifications
BSC
MCA
Checkbox
Gender
Male
Female
Radio Button
User Hobbies
Profession
Label
List
Cricket
Music
Chatting
Student
Employee
Teacher
Business
Choice
User Address
TextArea
23/3rt, S R Nagar
Hyd-38
Button
Registration
Cancel

JAVA AWT BUTTON

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

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

public class ButtonExample {
   public static void main(String[] args) {
      Frame f = new Frame("Button Example");
      Button b = new Button("Goto BVRITH");
      b.setBounds(100, 100, 80, 30);
      f.add(b);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);
      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }
}
```



setBounds(int x,int y,int width,int height)

This method is used to declare location ,width & height of all components of AWT.

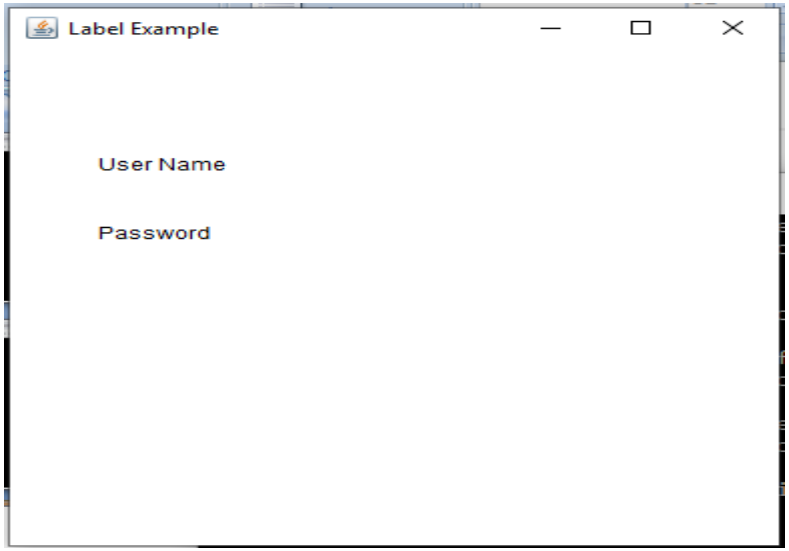
Example: setBounds(50,100,80,30);

X, Y, width, Height

JAVA AWT LABEL

The object of Label 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.

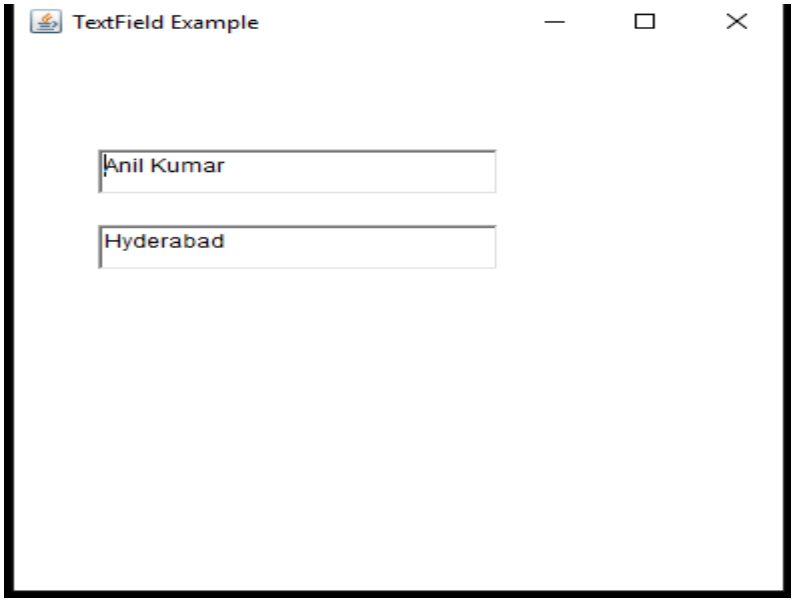
```
import java.awt.*;
import java.awt.event.*;

class LabelExample {

   public static void main(String args[]) {
      Frame f = new Frame("Label Example");
      Label l1, l2;
      l1 = new Label("User Name");
      l1.setBounds(50, 100, 100, 30);
      l2 = new Label("Password");
      l2.setBounds(50, 150, 100, 30);
      f.add(l1);
      f.add(l2);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });

   }
}
```



JAVA AWT TEXTFIELD

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

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

class TextFieldExample {

   public static void main(String args[]) {
      Frame f = new Frame("TextField Example");
      TextField t1, t2;
      t1 = new TextField("Anil Kumar");
      t1.setBounds(50, 100, 200, 30);
      t2 = new TextField("Hyderabad");
      t2.setBounds(50, 150, 200, 30);
      f.add(t1);
      f.add(t2);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });

   }
}
```



JAVA AWT TEXTAREA

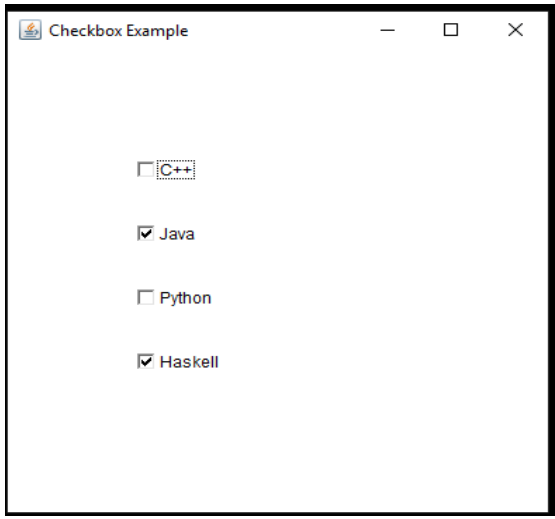
The object of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

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

public class TextAreaExample {

   TextAreaExample() {
      Frame f = new Frame();
      TextArea area = new TextArea("Welcome to BVRIT HYDERABAD");
      area.setBounds(50, 100, 300, 100);
      f.add(area);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);
      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      new TextAreaExample();
   }
}
```

JAVA AWT CHECKBOX

The Checkbox 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".

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

public class CheckboxExample {

   CheckboxExample() {
      Frame f = new Frame("Checkbox Example");
      Checkbox checkbox1 = new Checkbox("C++");
      checkbox1.setBounds(100, 100, 50, 50);
      Checkbox checkbox2 = new Checkbox("Java", true);
      checkbox2.setBounds(100, 150, 50, 50);
      Checkbox checkbox3 = new Checkbox("Python");
      checkbox3.setBounds(100, 200, 70, 50);
      Checkbox checkbox4 = new Checkbox("Haskell", true);
      checkbox4.setBounds(100, 250, 70, 50);
      f.add(checkbox1);
      f.add(checkbox2);
      f.add(checkbox3);
      f.add(checkbox4);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      new CheckboxExample();
   }
}
```

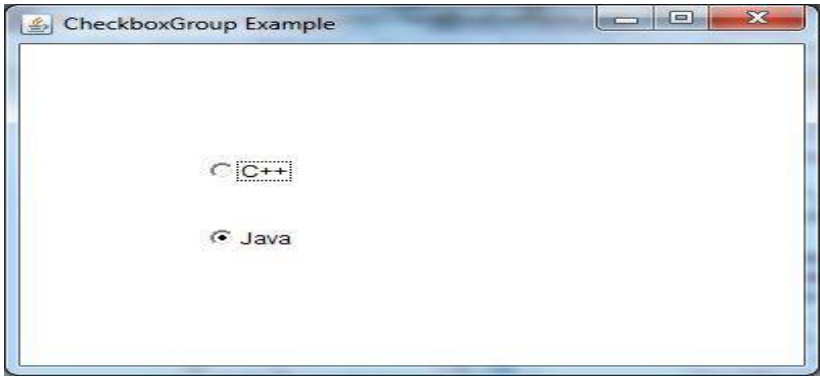


JAVA AWT CHECKBOXGROUP

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

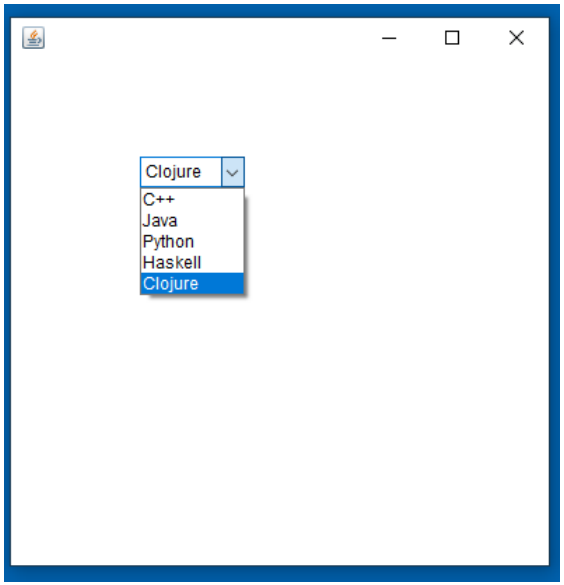
```
import java.awt.*;
import java.awt.event.*;

public class CheckboxGroupExample {

   CheckboxGroupExample() {
      Frame f = new Frame("CheckboxGroup Example");
      CheckboxGroup cbg = new CheckboxGroup();
      Checkbox checkBox1 = new Checkbox("C++", cbg, false);
      checkBox1.setBounds(100, 100, 50, 50);
      Checkbox checkBox2 = new Checkbox("Java", cbg, true);
      checkBox2.setBounds(100, 150, 50, 50);
      f.add(checkBox1);
      f.add(checkBox2);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      new CheckboxGroupExample();
   }
}
```



JAVA AWT CHOICE

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 Component class.

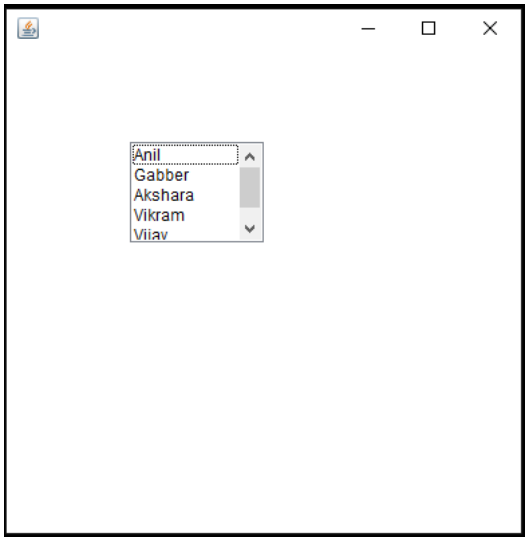
```
import java.awt.*;
import java.awt.event.*;

public class ChoiceExample {

   ChoiceExample() {
      Frame f = new Frame();
      Choice c = new Choice();
      c.setBounds(100, 100, 75, 75);
      c.add("C++");
      c.add("Java");
      c.add("Python");
      c.add("Haskell");
      c.add("Clojure");
      f.add(c);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      new ChoiceExample();
   }
}
```



JAVA AWT LIST

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

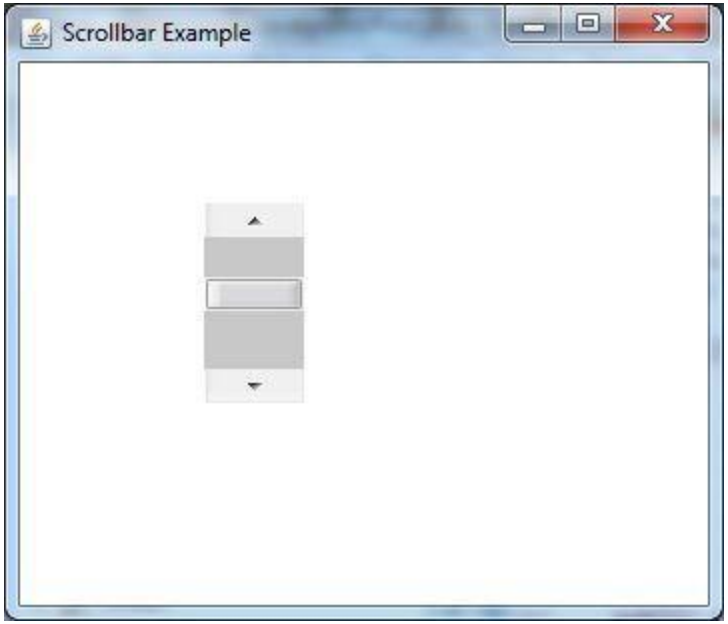
```
import java.awt.*;
import java.awt.event.*;

public class ListExample {

   ListExample() {
      Frame f = new Frame();
      List l1 = new List(5);
      l1.setBounds(100, 100, 100, 75);
      l1.add("Anil");
      l1.add("Gabber");
      l1.add("Akshara");
      l1.add("Vikram");
      l1.add("Vijay");
      f.add(l1);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      new ListExample();
   }
}
```



JAVA AWT SCROLLBAR

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

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

class ScrollbarExample {

    ScrollbarExample() {
        Frame f = new Frame("Scrollbar Example");
        Scrollbar s = new Scrollbar();
        s.setBounds(100, 100, 50, 100);
        f.add(s);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String args[]) {
        new ScrollbarExample();

    }
}
```

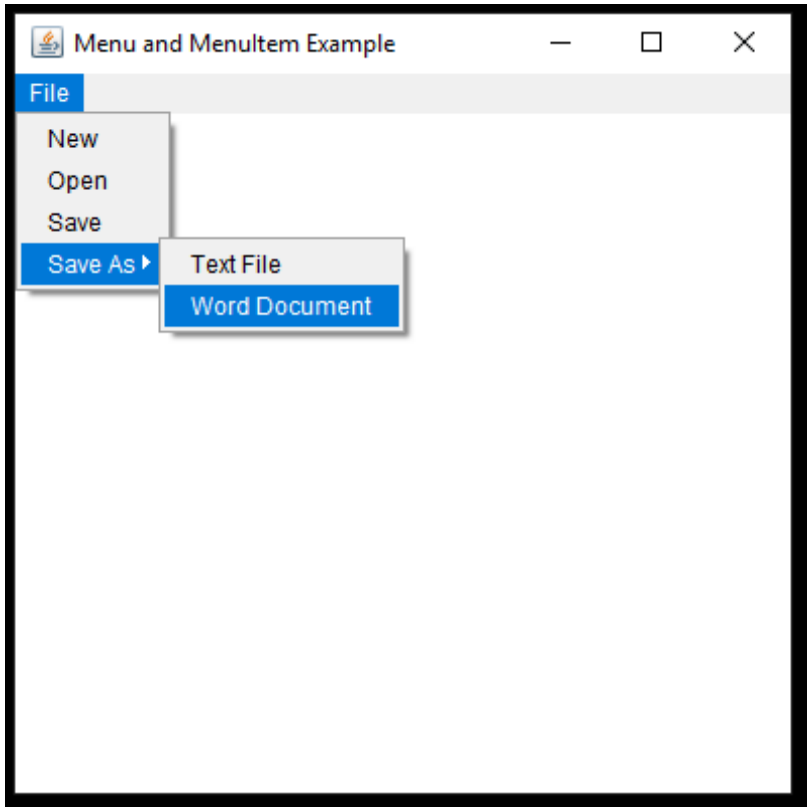


JAVA AWT MENUITEM AND MENU

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

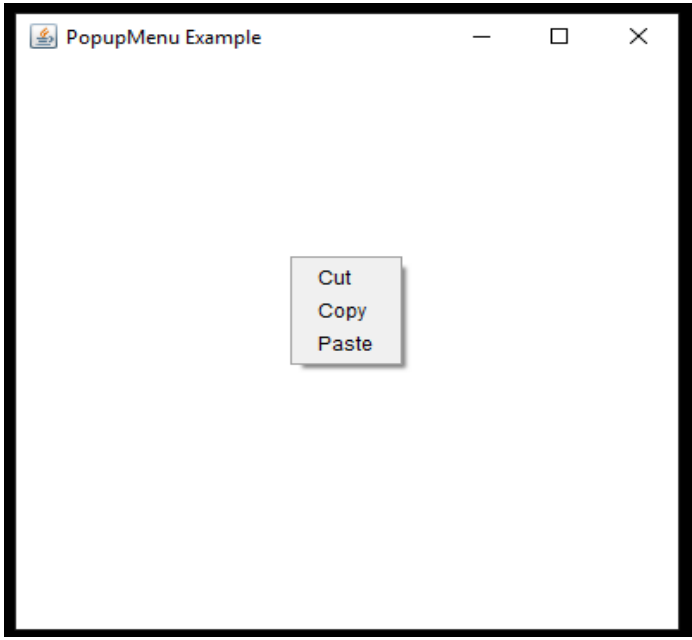
```
import java.awt.*;
import java.awt.event.*;

class MenuExample {
   MenuExample() {
      Frame f = new Frame("Menu and MenuItem Example");
      MenuBar mb = new MenuBar();
      Menu menu = new Menu("File");
      Menu submenu = new Menu("Save As");
      MenuItem i1 = new MenuItem("New");
      MenuItem i2 = new MenuItem("Open");
      MenuItem i3 = new MenuItem("Save");
      MenuItem i4 = new MenuItem("Text File");
      MenuItem i5 = new MenuItem("Word Document");
      menu.add(i1);
      menu.add(i2);
      menu.add(i3);
      submenu.add(i4);
      submenu.add(i5);
      menu.add(submenu);
      mb.add(menu);
      f.setMenuBar(mb);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }
   public static void main(String args[]) {
      new MenuExample();
   }
}
```



JAVA AWT POPUPMENU

PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.

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

class PopupMenuExample {

   PopupMenuExample() {
      final Frame f = new Frame("PopupMenu Example");
      final PopupMenu popupmenu = new PopupMenu("Edit");
      MenuItem cut = new MenuItem("Cut");
      cut.setActionCommand("Cut");
      MenuItem copy = new MenuItem("Copy");
      copy.setActionCommand("Copy");
      MenuItem paste = new MenuItem("Paste");
      paste.setActionCommand("Paste");
      popupmenu.add(cut);
      popupmenu.add(copy);
      popupmenu.add(paste);
      f.addMouseListener(new MouseAdapter() {
         public void mouseClicked(MouseEvent e) {
            popupmenu.show(f, e.getX(), e.getY());
         }
      });
      f.add(popupmenu);
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);

      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      new PopupMenuExample();
   }
}
```

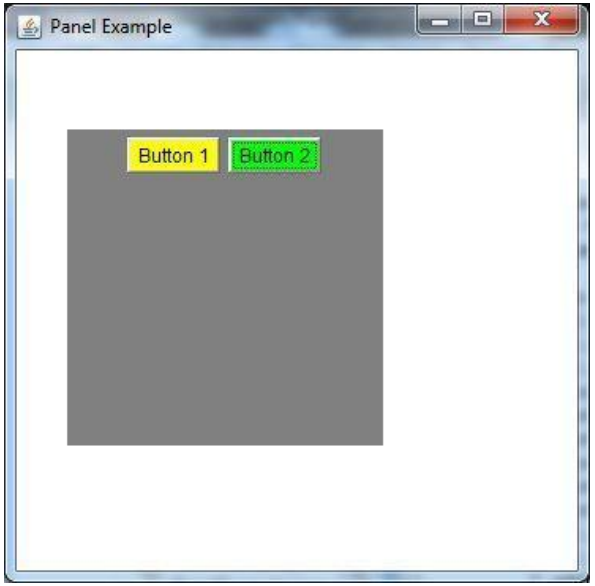


JAVA AWT PANEL

The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.

It doesn't have title bar.

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

public class PanelExample {

    PanelExample() {
        Frame f = new Frame("Panel Example");
        Panel panel = new Panel();
        panel.setBounds(40, 80, 200, 200);
        panel.setBackground(Color.gray);
        Button b1 = new Button("Button 1");
        b1.setBounds(50, 100, 80, 30);
        b1.setBackground(Color.yellow);
        Button b2 = new Button("Button 2");
        b2.setBounds(100, 100, 80, 30);

        b2.setBackground(Color.green);
        panel.add(b1);
        panel.add(b2);
        f.add(panel);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String args[]) {
        new PanelExample();
    }
}
```



LAYOUT MANAGERS

We Create several components like push buttons, checkboxes, radio buttons etc. in GUI. After creating these components, they should be placed in the fram (in AWT) or container (in Swing). While arranging them in the frame or container, they can be arranged in a perticular manner by using layout mangers. We have LayoutManger interface in java.awt package which is implemented in various classes which provides various layouts to arrange the components.

The following classes represents the layout managers in java :

- **Flow Layout**
- **Border Layout**
- **Card Layout**
- **Grid Layout**
- **Grid Bag layout**
- **Box Layout**

To set a perticular layout, we should first create an object to the layout class and pass the object to setLayout() method. For example to set FlowLayout to the container that holds the components, we can write:

```
FlowLayout obj = new FlowLayout();
c.setFlowLayout(obj);
```

FlowLayout

FLOWLAYOUT

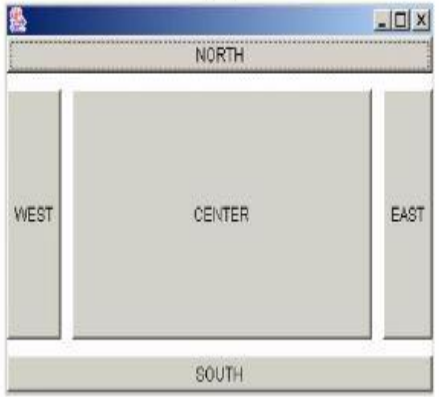
```
import java.awt.*;
class FlowLayoutDemo extends Frame {
   public static void main(String args[]) {
      FlowLayoutDemo fld = new FlowLayoutDemo();
      fld.setLayout(new FlowLayout(FlowLayout.RIGHT,
                                   10, 10));
      fld.add(new Button("ONE"));
      fld.add(new Button("TWO"));
      fld.add(new Button("THREE"));
      fld.setSize(100, 100);
      fld.setVisible(true);
   }
}
```

BORDER LAYER

```
import java.awt.*;
class BorderLayoutDemo extends Frame {
   public static void main(String args[]) {
     BorderLayoutDemo bld = new BorderLayoutDemo();
     bld.setLayout(new BorderLayout(10, 10));
     bld.add(new Button("NORTH"), BorderLayout.NORTH);
     bld.add(new Button("SOUTH"), BorderLayout.SOUTH);
     bld.add(new Button("EAST"), BorderLayout.EAST);
     bld.add(new Button("WEST"), BorderLayout.WEST);
     bld.add(new Button("CENTER"), BorderLayout.CENTER);
     bld.setSize(200, 200);
     bld.setVisible(true);
   }
}
```

- Sample output:

- After resizing:



GRIDLAYOUT

```
import java.awt.*;
class GridLayoutDemo extends Frame {
   public static void main(String args[]) {
      GridLayoutDemo gld = new GridLayoutDemo();
      gld.setLayout(new GridLayout(2, 3, 4, 4));
      gld.add(new Button("ONE"));
      gld.add(new Button("TWO"));
      gld.add(new Button("THREE"));
      gld.add(new Button("FOUR"));
      gld.add(new Button("FIVE"));
      gld.setSize(200, 200);
      gld.setVisible(true);
   }
}
```

- Sample

- After