Vector

Overview

1. The **Underlying Data Structure** is **Resizable Array OR Growable Array**.
2. **Insertion Order** is **Preserved**.
3. **Duplicate** Objects are **allowed**.
4. **Heterogeneous** Objects are **allowed.**
5. **Null Insertion** is **Possible**.
6. **Extends AbstractList** class and Implements **List**, **Serializable**, **Cloneable** and **RandomAccess** interfaces.

   ```
   public class Vector<E>
   extends AbstractList<E>
   implements List<E>, RandomAccess, Cloneable, java.io.Serializable
   ```

7. **Every Method** Present Inside Vector **is Synchronized** and **Hence Vector** Object is **Thread Safe**
8. It's in the **java.util package**.
9. The **iterators** returned by this class's iterator() and listIterator(int) **methods are fail-fast.**
10. **Vectors** are known to **give ConcurrentModificationException when accessed concurrently at the time of modification**.
11. Vectors are **slower in performance** as they acquire a lock on a thread.

Note:
● Fail-Fast **iterators** immediately **throw ConcurrentModificationException if** there is **structural modification** of the collection.
● Structural modification means adding, removing any element from a collection while a thread is iterating over that collection.

Important points regarding the Increment of vector capacity

1. The **vector** will **expand** in accordance **with the increment if** one is **supplied**.
2. **if** the **increment** is **not mentioned**, then each allocation cycle **doubles the vector's capacity**.
3. **Three protected data members** are defined by Vector
    a. **int capacityIncreament:** Contains the value of the increment

   b. **int elementCount**: Number of elements that are currently stored in the vector.
   c. **Object elementData[]**: The vector is kept in an array that is stored in it.

Constructors in Vectors

1. Vector():
   a. A default vector of capacity 10 gets created while calling this constructor.
   b. eg.,
      Vector<E> v = new Vector<E>();

2. Vector(int size):
   a. A vector is created with the given size as its capacity.
   b. eg.,
      Vector<E> v = new Vector<E>(10);

3. Vector(int size, int increment):
   a. A vector is created with the given size as its initial capacity, and whenever the capacity needs to be increased, it is increased by the given increment count.
   b. eg.,
      Vector<E> v = new Vector<E>(10,5);
   c. Initial size is 10 and increment size is 5.

4. Vector(Collection c):
   a. A Java vector is constructed from the given collection with the same order of elements as in the collection.
   b. eg.,
      Vector<E> v = new Vector<E>(Collection c);

Increment of Vector Capacity

- **By default**, the vector **increases** its **capacity by double**.
- **if** an **increment** is **specified** in its constructor, **Vector** will **grow in accordance** with it in each allocation cycle.
  e.g., Vector<E> v = new Vector<E>(20,5);
  - Vector **initial capacity** is **20** and **capacity increment** is **5**.
  - If **Vector gets full** and we try to **add a new element**, the **vector size** will **grow by 5**.
  - So the **new size** of the vector becomes **25**.

Vector Methods

Sr. No.	Method	Description
1	add(E e)	For **appending** the given **element e** in the given vector.
2	add(int index, E e)	For **inserting** the given **element e**, at the given index.
3	addAll(Collection c)	For **appending all the elements** from **collection c** to the java vector.
4	addAll(int index, Collection c)	For **inserting all the elements** present in the given **collection c** to the given java vector **at the given index**.
5	addElement(E e)	For **appending** the **element to** the **last** of the vector. Keep in mind, this method increases the size of the vector by one.
6	capacity()	For **getting the length** of the actual array inside the vector.
7	clear()	For **removing all of the elements** from the given vector in java.
8	clone()	For **making a clone** of the given vector in java.
9	contains(Object o)	For **finding** if the given vector contains the **specified element**. It return true if element is found.
10	copyInto(Object[] objArray)	For **copying the elements** of the given vector **into the array** passed in.
11	elementAt(int index)	For **accessing** the **element at the given index.**
12	elements()	For **getting an enumeration of the components** of the given vector in java.
13	equals(Object obj)	For **comparing** and telling if the **given object and the Vector** are equal or not.
14	firstElement()	For **getting the first object** of the vector present at index 0.
15	get(int index)	For **getting the element at the given index** from the Vector in java.
16	hashCode()	For **calculating the hash code** value for the Vector in java.
17	indexOf(Object o)	For **finding** out the **index** of the **first occurrence of** the **specified element** in this java vector. Keep in mind, if the element is not present, it will return -1.
18	insertElementAt(E element, int index)	For **inserting** the given element **at the index** in this java vector.
19	isEmpty()	For **finding** if the **vector is empty or not**.
20	lastElement()	For **getting the last element.**

21	lastIndexOf(Object o, int index)	For **finding the last occurrence** of the given element and its corresponding index. It **searches in reverse order** and returns -1 if the element is not found in the vector.
22	remove(int index)	For **removing** the **element at** the given **position**.
23	remove(Object o)	For **removing the first occurrence** of the given **element** in this vector.
24	removeAll(Collection<?> c)	For **removing all the elements** from the Vector **that are present in the given Collection.**
25	removeAllElements()	For **removing all the elements from** this **vector** in java and set its size to zero.
26	removeElement(Object obj)	For **removing** the **first occurrence** (going from the index 0) of the object from the vector.
27	removeElementAt(int index)	For **deleting** the **element at the given index** from the vector in java.
28	removeIf(Predicate<? super E> filter)	For **removing all of the elements of** this **collection** that **satisfy** the given **predicate**.
29	removeRange(int fromIndex,int toIndex)	For **removing all the elements from** this vector whose **index is between** **fromIndex**, (inclusive), and **toIndex**, (exclusive).
30	replaceAll(UnaryOperator<E> operator)	For **replacing each element** of this list with the result of applying the operator to that element.
31	retainAll(Collection<?> c)	For **deleting every element** of vector **except** the ones **that are contained** **in** the **given Collection**.
32	set(int index, E element)	For **replacing** the **element at the given index with the given element**.
33	setSize(int newSize)	For **setting** the **size** of the given java vector to the size given.
34	size()	For **getting** the **number of elements** in the java vector.
35	sort(Comparator<? super E> c)	For **sorting** the vector **according to the order** induced **by the Comparator**.
36	subList(int fromIndex, int toIndex)	This method **returns** a view of the **portion** of the vector **between fromIndex and toIndex -1** (both inclusive).
37	toArray()	For **getting an array** containing all of the elements in this java vector.
38	toArray(T[] a)	For **getting an array** with all of the elements in this vector in the correct order. Here the **runtime type of the returned array** is that of the specified array.
39	toString()	For **getting a string representation of Vector** in Java, containing the String representation of each element.

40	trimToSize()	For **making the capacity** of the vector in java to be equal **to the vector's current size**.

Iterating over the elements

- Can iterate over elements using two ways
    - Use get() method
      ```
      for (int index = 0; index < ourVector.size(); index++)
      {
          System.out.print(ourVector.get(index) + " ");
      }
      ```
    - Use foreach loop
      ```
      for (String str : ourVector)
      {
          System.out.print(str + " ");
      }
      ```

Replacing elements

- We can replace element using **set()** method
  e.g.,
  ```
  public static void main(String args[])
  {
          Vector<String> ourVector = new Vector<>();
          ourVector.add("happy");
          ourVector.add("crying");
          System.out.println("Vector before update: " + ourVector);

     // Using set() method to replace "crying" with "laughing"
          ourVector.set(1, "laughing");
  }
  ```

Removing Elements

- Using remove(int index)
  removes the element present at that specific index

- Using remove(Object o)
  only the first occurrence of the object is removed.

Stack

Overview

- It is a child class of Vector.
- Designed for LIFO (Last In First Out) operation.

Constructor

Stack s = new Stack<>();

Methods of Stack

1. Object push(Object o); To Insert an Object into the Stack.
2. Object pop(); To Remove and Return Top of the Stack.
3. Object peek(); Ro Return Top of the Stack without Removal.
4. boolean empty(); Returns true if Stack is Empty
5. int search(Object o);Returns Offset if the Element is Available Otherwise Returns -1.