Vector (Since Java 1.0)

1. All **Methods** are **synchronized.**
2. **Thread safe:** Only **one thread** is allowed to operate on vector object at a time.
3. It increases the **waiting time** of threads (since all the methods are synchronized) and hence performance is low
4. Not recommended to use in **performance point** of view.
5. Vector is introduced in Java 1.0. Hence Vector class is **legacy.**
6. Vector data structure is **growable array or resizable array**
7. Vector is maintained index based
8. Vector can store duplicate elements
9. Vector can store multiple null values
10. Vector follows insertion order and no sorting order in vector

Collection(I) → List(I)→ ArrayList(C), LinkedList(C) and Vector(C)→ Stack(C)

```
//Vector Constructor
Vector<String> vector = new Vector<String>();
//public synchronized boolean add(E e) {}
vector.add("One");
vector.add("Two");
vector.add("Three");
vector.add("Four");
System.out.println(vector); // [One, Two, Three, Four]

Vector<String> v = new Vector<String>(vector);
v.add("NameOne");
v.add("NameTwo");
v.add("NameThree");
v.add("NameFour");
System.out.println(v); // [One, Two, Three, Four, NameOne, NameTwo, NameThree, NameFour]
```

```
//Vector Constructor can be used for conversion from HashSet to Vector
Set<String> hashSet = new HashSet<String>();
hashSet.add("NameOne");
hashSet.add("NameTwo");
hashSet.add("NameThree");
hashSet.add("NameFour");
System.out.println(hashSet); // [NameOne, NameFour, NameTwo, NameThree]

Vector<String> v = new Vector<String>(hashSet);
System.out.println(v); // [NameOne, NameFour, NameTwo, NameThree]
```

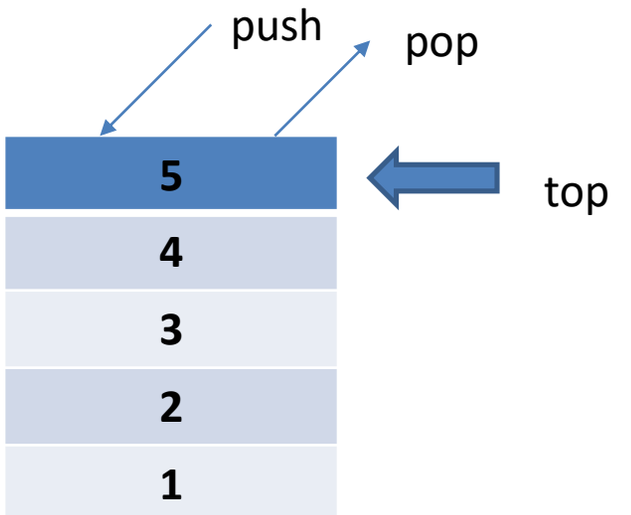
```
//Vector Methods
Vector<String> vector = new Vector<String>();
// public synchronized void addElement(E obj) {}
vector.addElement("One");
vector.addElement("Two");
vector.addElement("Three");
vector.addElement("Four");
System.out.println(vector); // [One, Two, Three, Four]

// public synchronized E firstElement() {}
System.out.println(vector.firstElement()); // One

// public synchronized E lastElement() {}
System.out.println(vector.lastElement()); // Four
```

ArrayList	Vector
JDK 1.2 Version	JDK 1.0 Version
Non Legacy Class	Legacy Class
Not Synchronized	Synchronized
Fast because it not synchronized	Slow because it is synchronized

Stack (Since Java 1.0)

1. **Stack** is child class of **Vector**
2. Stack class in java represents **LIFO** (Last in First Out) stack of objects.
3. There are only **5 methods** in Stack
4. Stack class is **legacy**
5. Stack is a **linear data structure**
6. Stack has only **one end (top)**

Note: LIFO : The element which we added last it remove first from data



1)public E push(E item);

Pushes the item on **top** of the stack

2)public synchronized E pop();

Removes the item at the **top** of the stack and returns that item

3) public synchronized E peek();

Returns the item at the **top** of the stack

4) public boolean empty();

Checks whether **stack** is empty or not

5) public synchronized int search(Object o);

Returns the **position** of an object in the stack.

```
//add
Stack<String> stack = new Stack<String>();
// public synchronized boolean add(E e) //Vector.class
stack.add("NameOne");
stack.add("NameTwo");
stack.add("NameThree");
stack.add("NameFour");
System.out.println(stack); // [NameOne, NameTwo, NameThree, NameFour]

// public void add(int index, E element) //Vector.class
stack.add(0, "NameZero");
System.out.println(stack); // [NameZero, NameOne, NameTwo, NameThree, NameFour]

stack.add(5, "NameFive");
System.out.println(stack); // [NameZero, NameOne, NameTwo, NameThree, NameFour, NameFive]
```

```
//push and pop //  Pushes the item on top of the stack
Stack<Integer> stack = new Stack<Integer>();
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);
System.out.println(stack); // [1, 2, 3, 4]

stack.pop();
System.out.println(stack); // [1, 2, 3]

stack.pop();
System.out.println(stack); // [1, 2]

stack.pop();
System.out.println(stack); // [1]

stack.pop();
System.out.println(stack); // []
```

```
//search // Returns the position of an object in the stack.
Stack<String> stack = new Stack<String>();
stack.add("A");
stack.add("B");
stack.add("C");
stack.add("D");
stack.add("E");
System.out.println(stack); // [A, B, C, D, E]
System.out.println(stack.search("A")); // 5
System.out.println(stack.search("E")); // 1
System.out.println(stack.search("F")); // -1
```

```
//peek() //  Returns the item at the top of the statck
Stack<String> stack = new Stack<String>();
stack.add("A");
stack.add("B");
stack.add("C");
stack.add("D");
stack.add("E");
System.out.println(stack.peek()); // E
System.out.println(stack.peek()); // E
System.out.println(stack.peek()); // E
```

```
//empty // Checks whether stack is empty or not
Stack<String> stack = new Stack<String>();
stack.add("A");
stack.add("B");
stack.add("C");
stack.add("D");
stack.add("E");
System.out.println(stack.empty()); // false
stack.clear();
System.out.println(stack.empty()); // true
```