```java
package com.ravi.vector;

import java.util.Collections;
import java.util.Vector;

//Store some manager Object, sort the Object based on ID, filter all the manager
//Object whose salary is > 50000


record Manager(Integer id, String name, Double salary)
{
}

public class VectorDemo6
{
    public static void main(String[] args)
    {
        Vector<Manager> listOfManagers = new Vector<>();
        listOfManagers.add(new Manager(333, "Scott", 72000D));
        listOfManagers.add(new Manager(444, "Smith", 65000D));
        listOfManagers.add(new Manager(222, "Alen", 44000D));
        listOfManagers.add(new Manager(111, "John", 55000D));
        listOfManagers.add(new Manager(555, "Martin", 49000D));

        System.out.println("Managed Data, Sorted Based on the Id :");

        Collections.sort(listOfManagers, (m1, m2)-> m1.id().compareTo(m2.id()));
        listOfManagers.forEach(System.out::println);

        System.out.println("Retrieve all the Manager Object whose salary > 50K");

        for(Manager manager : listOfManagers)
        {
          if(manager.salary() > 50000)
          {
             System.out.println(manager);
          }
        }
    }
}
```

***What is Fail Fast Iteartor ?

While retrieving the object from the collection by using Itearor interface or for each loop, if at any point of time the original structure is going to modify after the creation of Itearator by using any way except Iterators's own method [remove(), add(), set()] then we will get java.util.ConcurrentModificationExacption.

```java
package com.ravi.vector;

import java.util.Iterator;
import java.util.Vector;

//IPL Auction

record Player(Integer id, String name, Double basePrice)
{

}

class IPLAuction
{
  private String iplTeam;
  private Vector<Player> listOfPlayers;


  public IPLAuction(String iplTeam)
  {
      this.iplTeam = iplTeam;
      listOfPlayers = new Vector<Player>();  //Composition
  }

  public void addPlayer(Player player)
  {
      listOfPlayers.add(player);
  }

  public void displayAllThePlayers()
  {
      System.out.println("All the plyers "+iplTeam);
      listOfPlayers.forEach(player -> System.out.println(player));
  }


  public void retainOrRemove()
  {
     Iterator<Player> iterator = listOfPlayers.iterator();

     while(iterator.hasNext())
     {
        Player player = iterator.next();

        if(player.basePrice() > 100000)
        {
           iterator.remove();
        }
     }
  }
}

public class VectorDemo7
{
    public static void main(String[] args)
    {
        IPLAuction ipl = new IPLAuction("SRH");
        ipl.addPlayer(new Player(111, "Abhishek", 190000D));
        ipl.addPlayer(new Player(222, "Head", 98000D));
        ipl.addPlayer(new Player(333, "Klassen", 92000D));
        ipl.addPlayer(new Player(444, "Nitish", 189000D));

        ipl.retainOrRemove();
        ipl.displayAllThePlayers();

    }
}
```

```java
package com.ravi.vector;

import java.util.Scanner;
import java.util.Vector;

public class VectorDemo8
{
    public static void main(String[] args)
    {
        Vector<String> toDoList = new Vector<>();

         Scanner scanner = new Scanner(System.in);

         int choice;
         do
         {
             System.out.println("To Do List Menu:");
             System.out.println("1. Add Task");
             System.out.println("2. View Tasks");
             System.out.println("3. Mark Task as Completed");
             System.out.println("4. Exit");

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

              choice = Integer.parseInt(scanner.nextLine());

             switch (choice)
             {
                 case 1:
                     // Add Task
                     System.out.print("Enter task description: ");
                     String task = scanner.nextLine();
                     toDoList.add(task);
                     System.out.println("Task added successfully!\n");
                     break;
                 case 2:
                     // View Tasks
                     System.out.println("To Do List:");
                     for (int i = 0; i < toDoList.size(); i++)
                     {
                       System.out.println((i + 1) + ". " + toDoList.get(i));
                     }
                     System.out.println();
                     break;
                 case 3:
                     // Mark Task as Completed
                     System.out.print("Enter task number to mark as completed: ");
                     int taskNumber = Integer.parseInt(scanner.nextLine());

                     if (taskNumber >= 1 && taskNumber <= toDoList.size())
                     {
                         String completedTask = toDoList.remove(taskNumber - 1);
                         System.out.println("Task marked as completed: " + completedTask +
"\n");
                     }
                     else
                     {
                         System.out.println("Invalid task number!\n");
                     }
                     break;
                 case 4:
                     System.out.println("Exiting ToDo List application. Goodbye!");
                     break;
                 default:
                     System.out.println("Invalid choice. Please enter a valid option.\n");
             }

         }
         while (choice != 4);


         scanner.close();


    }

}
```

Enumeration interface has provided a method from java 9V called asIterator(), the return type of this method is Iterator interface.

public Iterator asIterator(); [Enumeration interface method]

It is mainly used for Backward compatibility.

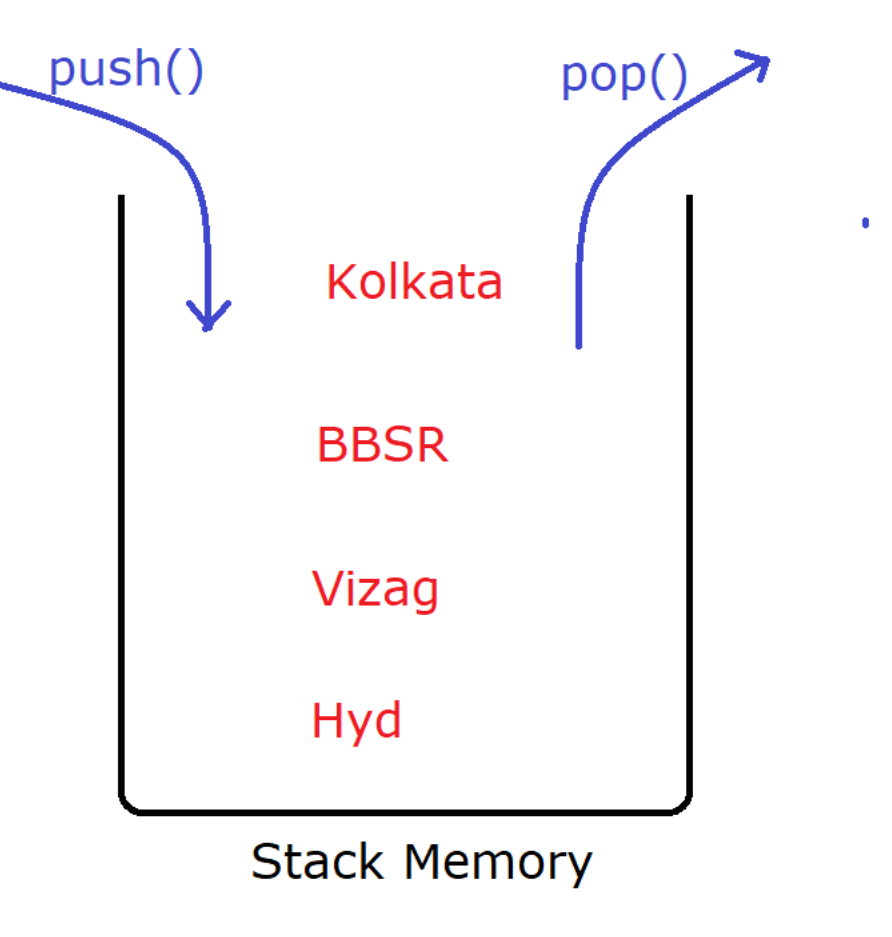
```java
package com.ravi.vector;

import java.util.Enumeration;
import java.util.Iterator;
import java.util.Vector;

record Product(int productId, String productName)
{

}

public class VectorDemo9
{
    public static void main(String[] args)
    {
        Vector<Product> listOfProduct = new Vector<>();
        listOfProduct.add(new Product(111, "Laptop"));
        listOfProduct.add(new Product(222, "Mobile"));
        listOfProduct.add(new Product(333, "Camera"));
        listOfProduct.add(new Product(444, "Bag"));
        listOfProduct.add(new Product(555, "Watch"));

        Enumeration<Product> elements = listOfProduct.elements();

        Iterator<Product> asIterator = elements.asIterator();
        asIterator.forEachRemaining(System.out::println);
    }
}
```

Stack<E> :
----
public class Stack<E> extends Vector<E>

* It is a legacy class introduced from JDK 1.0 and sub class of vector.
* It is a linear data structure which works on the basis of LIFO.

Let suppose I want to insert the following elements

Hyd, Vizag, BBSR, Kolkata



* It contains only one constructor :

Stack stk = new Stack();
Will create an empty Stack Object.

Methods :
----

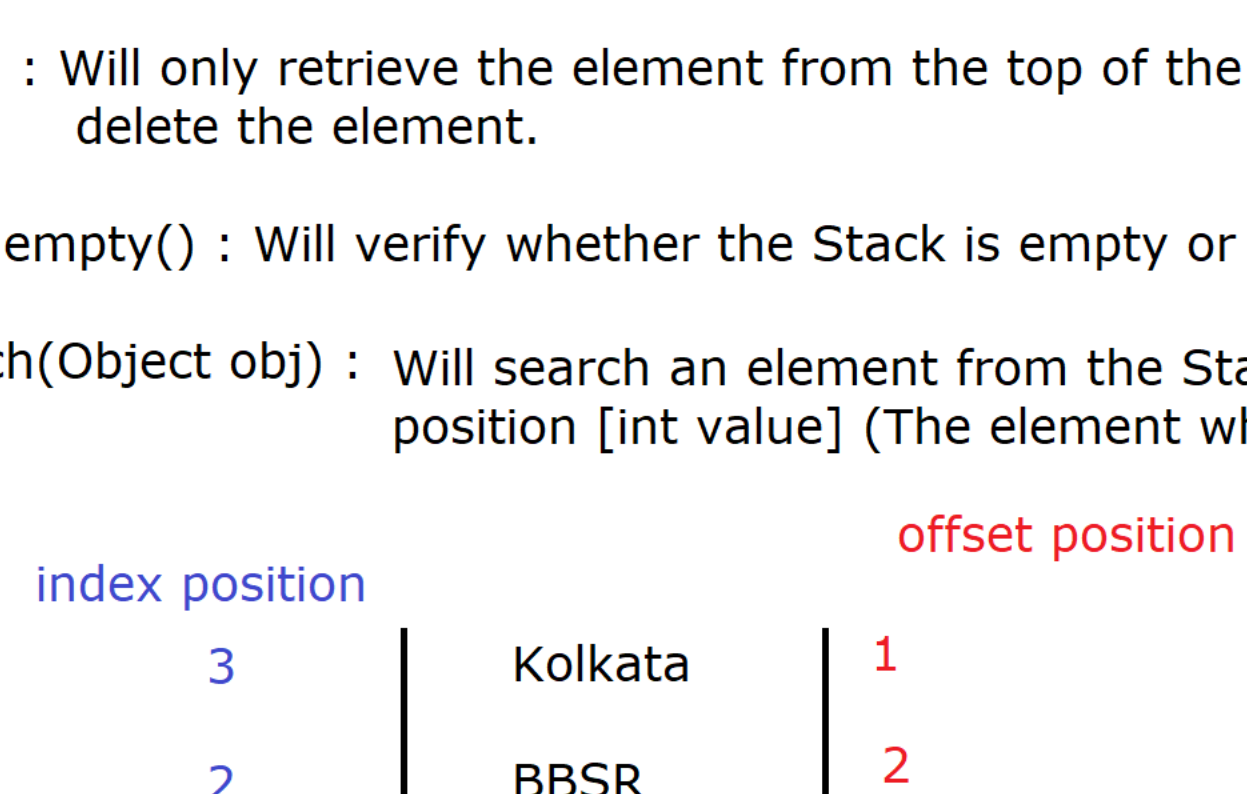
1) public E push() : Will insert the element in the top of the Stack just like addElement() method.

2) public E pop() : Will **retrieve and delete** the elements from the top of the Stack, If the stack is empty then It will throw an excption i.e java.util.EmptyStackException

3) public E peek() : Will only retrieve the element from the top of the Stack, Unlike pop() It will not delete the element.

4) public boolean empty() : Will verify whether the Stack is empty or not ?

5) public int search(Object obj) : Will search an element from the Stack and returns the offset position [int value] (The element which will be extracted first)



stk.search("Hyd"); //4
stk.search("Pune"); //-1

If given element is not available then It will return -1