Collection Framework

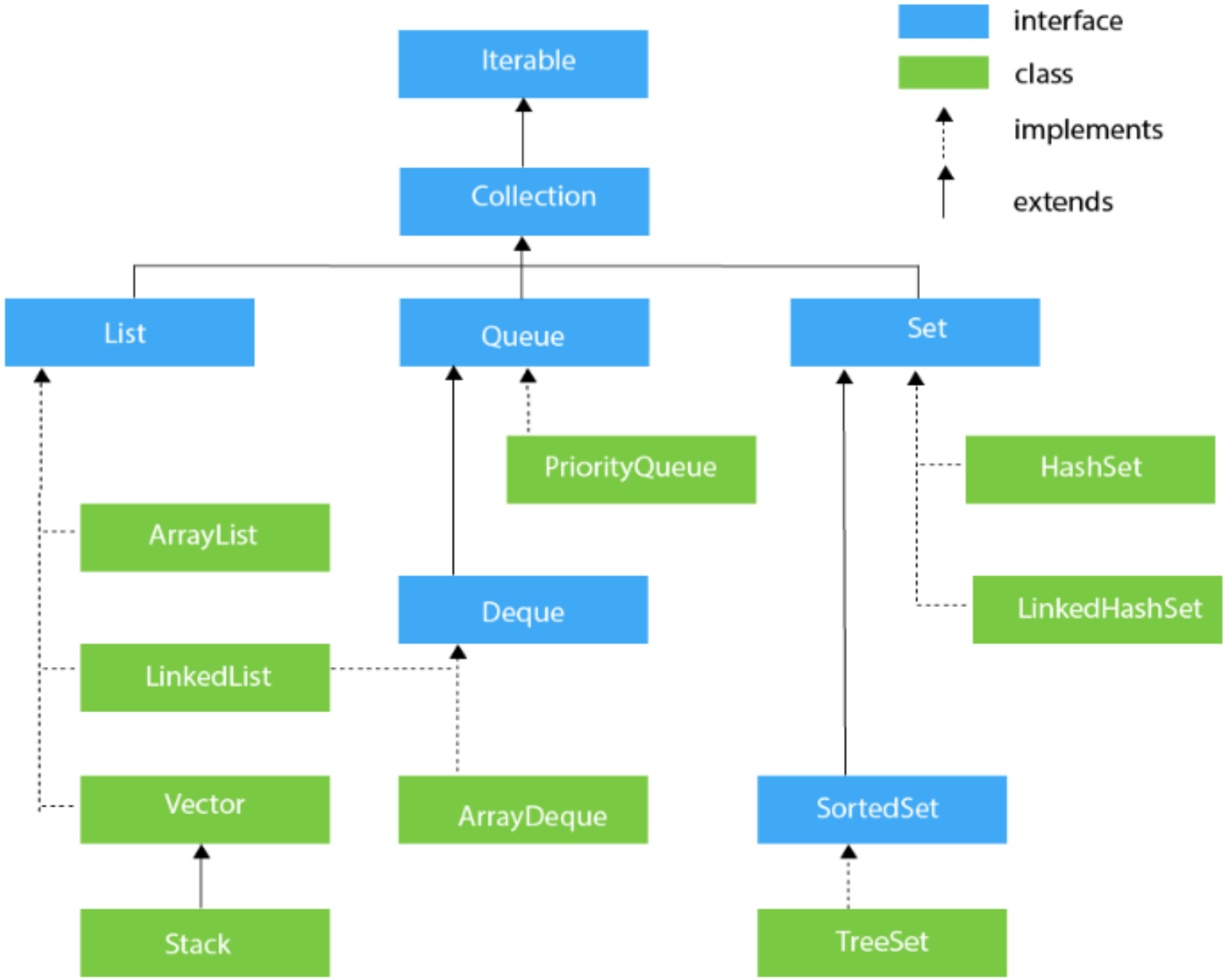
SMITA SANKHE
Assistant Professor
Department of Computer Engineering

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Collection Framework

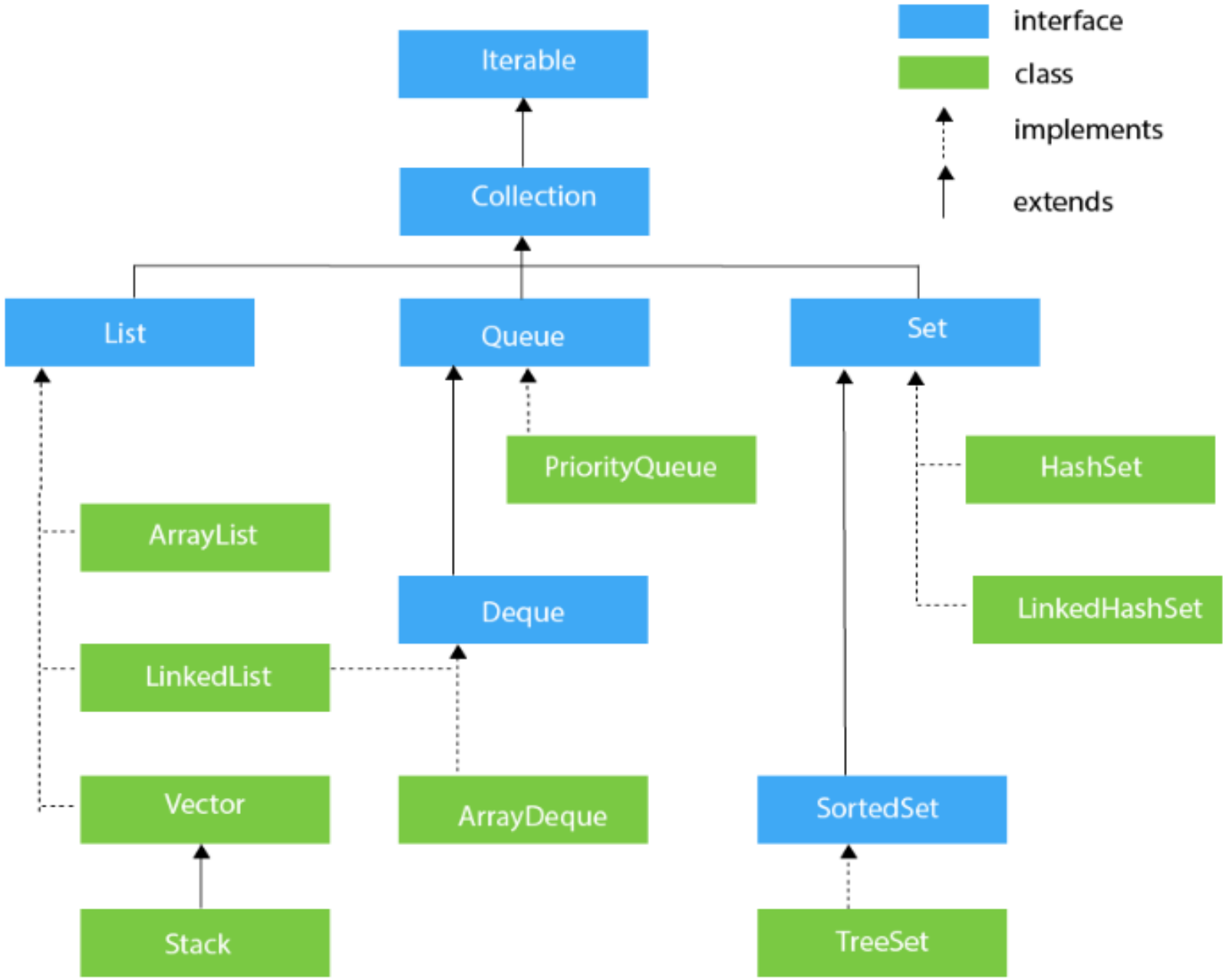
- Provides an architecture to store and manipulate the group of objects.
- Achieve all the operations that you perform on a data such as **searching, sorting, insertion, manipulation, and deletion.**
- Java Collection means a single unit of objects.
- Java Collection framework provides many **interfaces** (Set, List, Queue, Deque) and **classes** (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Iterator interface

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

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


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

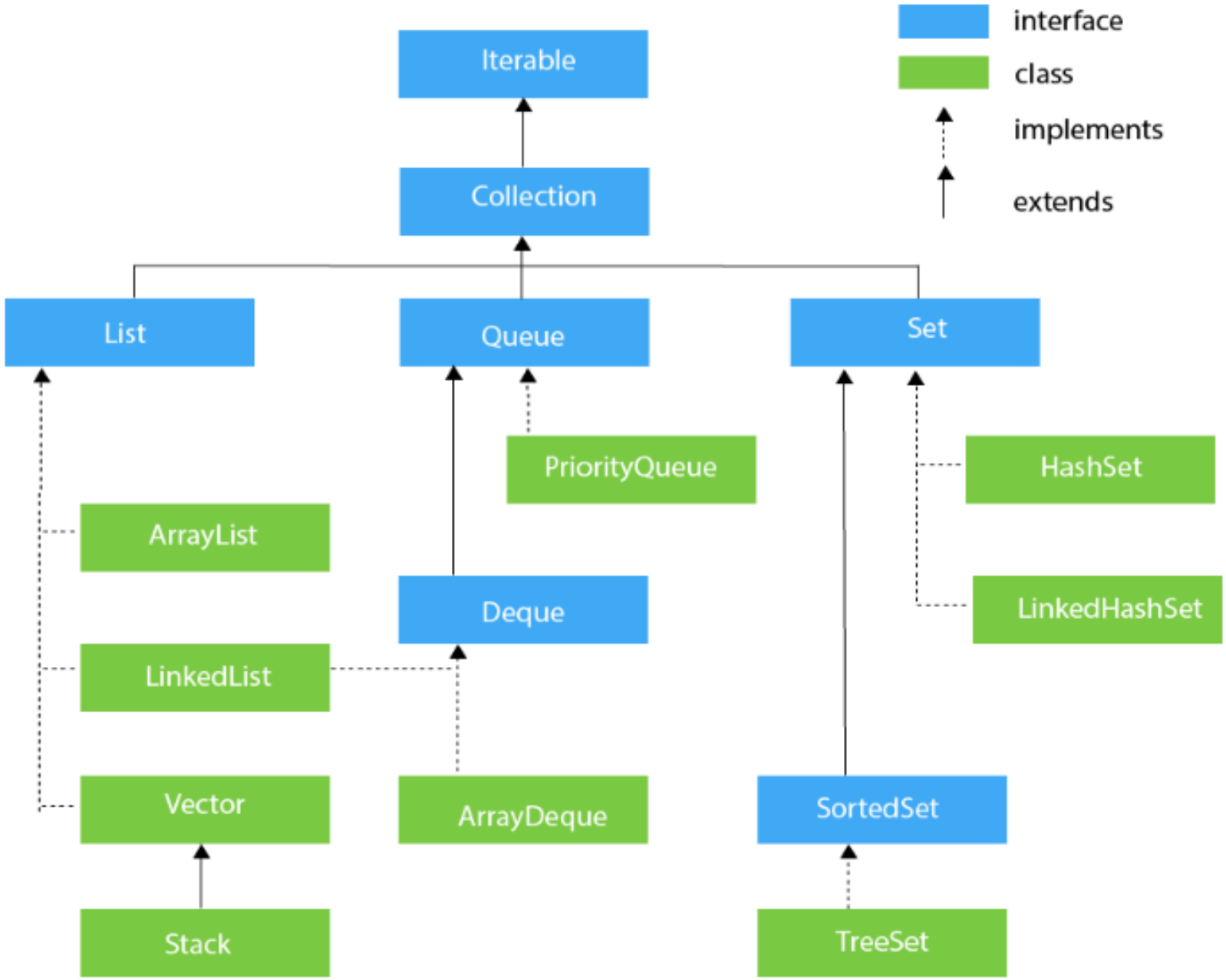
Collection Interface

- Interface which is implemented by all the classes in the collection framework.
- It declares the methods that every collection will have.

Collection interface builds the foundation on which the collection framework depends.

- Some of the methods of Collection interface are
  - Boolean add (Object obj)
  - Boolean addAll (Collection c)
  - void clear(), etc.

  which are implemented by all the subclasses of Collection interface.


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

List Interface

- List interface is the child interface of Collection interface.
- It inhibits a list type data structure in which we can store the ordered collection of objects.
- **It can have duplicate values.**
- List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
- To instantiate the List interface, we must use :

  List <data-type> list1= **new** ArrayList();

  List <data-type> list2 = **new** LinkedList();

  List <data-type> list3 = **new** Vector();

  List <data-type> list4 = **new** Stack();

ArrayList

- Java **ArrayList** class uses a **dynamic array for storing the elements**(there is *no size limit)*
- We can add or remove elements anytime, much more flexible than the traditional array.
- It is found in the ***java.util*** **package**.
- The ArrayList in Java can have the **duplicate elements** also.
- It **implements the List interface** so we can use all the methods of List interface here.
- The ArrayList maintains the insertion order internally

ArrayList Example

```java
import java.util.ArrayList;
class Main
{
    public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
    cars.get(0);
    cars.set(0, "Opel");
    cars.remove(1);
    System.out.println(cars);
    System.out.println("Cars Size "+cars.size());
    for (String i : cars) {
      System.out.println(i);
    }

    }
}
```

```
[Volvo, BMW, Ford, Mazda]
[Opel, Ford, Mazda]
Cars Size 3
Opel
Ford
Mazda
```

LinkedList

- Java LinkedList class uses a doubly linked list to store the elements.
- It provides a linked-list data structure.

Method	Description
addFirst()	Adds an item to the beginning of the list.
addLast()	Add an item to the end of the list
removeFirst()	Remove an item from the beginning of the list.
removeLast()	Remove an item from the end of the list
getFirst()	Get the item at the beginning of the list
getLast()	Get the item at the end of the list

LinkedList Example

```java
import java.util.LinkedList;
class Main
{
    public static void main(String[] args) {
    LinkedList<String> cars = new LinkedList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
    cars.get(0);
    cars.set(0, "Opel");
    cars.remove(1);
    System.out.println(cars);
    System.out.println("Cars Size "+cars.size());
    for (String i : cars) {
      System.out.println(i);
    }

    }
}
```

```
[Volvo, BMW, Ford, Mazda]
[Opel, Ford, Mazda]
Cars Size 3
Opel
Ford
Mazda
```

ArrayList v/s LinkedList

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

Vectors

- Vector implements a **dynamic array of objects**.
- Vector proves to be very useful **if you don't know the size of the array in advance** or you just need one that **can change sizes o**ver the lifetime of a program.
- Vector can contain heterogeneous objects
- We **cannot store elements of primitive data type;** first it need to be converted to objects. A vector can store any objects.
- Its defined in ***java.util*** package and class member of the Java Collections Framework.

Vectors

- Vector implements List Interface
- A vector has an initial capacity, if this capacity is reached then size of vector automatically increases.
- This default initial capacity of vectors are 10.
- Each vector tries to optimize storage management by maintaining a *capacity* and a *capacityIncrement* arguments.
- To traverse elements of a vector class we use **Enumeration** interface.

Vector Constructor and Description

Vector list = new Vector(); ⟶ Creates a default vector, which has an initial size 10.

Vector list = new Vector(int size); ⟶ Creates a vector, whose initial capacity is specified by size.

Vector list = new Vector(int size, int incr);

↓

Creates a vector, whose initial capacity is specified by size and whose increment is specified by incr.

Default Vector Example

```java
import java.util.*;
class Main
{
    public static void main(String args[]) {
        //Create a vector
        Vector<String> vec = new Vector<String>();
        //Adding elements using add() method of List
        vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");
        System.out.println("Elements are: "+vec);
        //Check size and capacity
        System.out.println("Size is: "+vec.size());
        System.out.println("Default capacity is: "+vec.capacity());
    }
}
```

```
Elements are: [Tiger, Lion, Dog, Elephant]
Size is: 4
Default capacity is: 10
```

Vector with Initial Size

```java
import java.util.*;
public class Main {
        public static void main(String args[]) {
            //Create an empty vector with initial capacity 4
            Vector<String> vec = new Vector<String>(4);
            //Adding elements to a vector
            vec.add("Tiger");
            vec.add("Lion");
            vec.add("Dog");
            vec.add("Elephant");
            //Check size and capacity
            System.out.println("Size is: "+vec.size());
            System.out.println("Default capacity is: "+vec.capacity());
            //Display Vector elements
            System.out.println("Vector element is: "+vec);

            vec.addElement("Rat");
            vec.addElement("Cat");
            vec.addElement("Deer");
            //Again check size and capacity after two insertions
            System.out.println("Size after addition: "+vec.size());
            System.out.println("Capacity after addition is: "+vec.capacity());
            //Display Vector elements again
            System.out.println("Elements are: "+vec);
        }
}
```

```
Size is: 4
Default capacity is: 4
Vector element is: [Tiger, Lion, Dog, Elephant]
Size after addition: 7
Capacity after addition is: 8
Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]
```

Vector with Initial Size and Increment

```java
import java.util.*;
public class Main {
        public static void main(String args[]) {
            //Create an empty vector with initial capacity 4
            Vector<String> vec = new Vector<String>(4,5);
            //Adding elements to a vector
            vec.add("Tiger");
            vec.add("Lion");
            vec.add("Dog");
            vec.add("Elephant");
            //Check size and capacity
            System.out.println("Size is: "+vec.size());
            System.out.println("Default capacity is: "+vec.capacity());
            //Display Vector elements
            System.out.println("Vector element is: "+vec);

            vec.addElement("Rat");
            vec.addElement("Cat");
            vec.addElement("Deer");
            //Again check size and capacity after two insertions
            System.out.println("Size after addition: "+vec.size());
            System.out.println("Capacity after addition is: "+vec.capacity());
            //Display Vector elements again
            System.out.println("Elements are: "+vec);
        }
}
```

```
Size is: 4
Default capacity is: 4
Vector element is: [Tiger, Lion, Dog, Elephant]
Size after addition: 7
Capacity after addition is: 9
Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]
```

Vector Methods

void **addElement**(Object *element*)	The object specified by *element* is added to the vector
int **capacity**()	Returns the capacity of the vector
boolean **contains**(Object *element*)	Returns **true** if element is contained by the vector, else **false**
void **copyInto**(Object *array[]*)	The elements contained in the invoking vector are copied into the array specified by array[]
elementAt(int *index*)	Returns the element at the location specified by *index*
Object **firstElement**().	Returns the first element in the vector

Vector Methods

void **insertElementAt**(Object *element*, int *index*)	*Adds element* to the vector at the location specified by *index*
boolean **isEmpty()**	Returns **true** if Vector is empty, **else false**
Object **lastElement()**	Returns the last element in the vector
void **removeAllElements()**	Empties the vector. After this method executes, the size of vector is zero.
void **removeElementAt**(int *index*)	Removes element at the location specified by *index*
void **setElementAt**(Object *element*, int *index*)	The location specified by *index* is assigned *element*

Vector Methods

void **setSize**(int *size*)	*Sets the number of elements in the vector to size. If the new size is less than the old size, elements are lost. If the new size is larger than the old, null elements are added*
int **size**()	Returns the number of elements currently in the vector

Vector add() Method

1. Java Vector add(int index, E element) Method
2. Java Vector add(E e) Method

```java
import java.util.Vector;
public class Main {
    public static void main(String arg[]) {
        //Create vector object
        Vector<String> vr = new Vector<>();
        vr.add("Java");
        vr.add("Android");
        vr.add(1, "Python");
        vr.add(3, "Hindi100");
        vr.add("Flutter");
        System.out.println("Vector is: " + vr);
        }
}
```

```
Vector is: [Java, Python, Android, Hindi100, Flutter]
```

Example: boolean add(E e)

```
import java.util.Vector;
public class VectorAddExample1 {
   public static void main(String arg[]) {
 //Create an empty Vector with an initial capacity 5
Vector<String> vc = new Vector<String>(4);
        vc.add("A");
        vc.add("B");
        vc.add("C");
        vc.add("D");
        vc.add("E");
 System.out.println("Elements of Vector are");
 for (String str : vc) {
           System.out.println("Alphabet= " +str);
        }
     }
}
```

```
import java.util.*;
public class TestJavaCollection
{
public static void main(String args[]){
Vector<String> v=new Vector<String>();
v.add("Ayush");
v.add("Amit");
v.add("Ashish");
v.add("Garima");
Iterator<String> itr=v.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Example: boolean addAll(int index, Collection<? extends E> c)

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        //Create a first empty vector
          Vector<String> vec1 = new Vector<>(4);
          //Add elements in the first vector
          vec1.add("E");
          vec1.add("F");
          vec1.add("G");
          vec1.add("H");

          //Create a second empty vector
          Vector<String> vec2 = new Vector<>(4);
          //Add elements in the second vector
          vec2.add("A");
          vec2.add("B");
          vec2.add("C");
          vec2.add("D");

          //Add elements of the vec2 at 1st element position in the vec1
          vec1.addAll(0, vec2);
          //Printing the final vector after appending
          System.out.println("Final vector list: "+vec1);
          }
}
```

```
Final vector list: [A, B, C, D, E, F, G, H]
```

Example: int capacity()

```
import java.util.Vector;
class Main {
    public static void main(String arg[]) {
        //Create an empty Vector with initial capacity of 5
        Vector<Integer> vecObject = new Vector<Integer>(5);

        //Add values in the vector
        vecObject.add(3);
        vecObject.add(5);
        vecObject.add(2);
        vecObject.add(4);
      System.out.println("Current capacity of Vector is: "+vecObject.capacity());
            }

}
```

```
Current capacity of Vector is: 5
```

Example: int size()

```java
import java.util.*;
public class Main {
    public static void main(String args[])
    {
        Vector<Integer> vec_tor = new Vector<Integer>();
        vec_tor.add(10);
        vec_tor.add(15);
        vec_tor.add(30);

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

        // Displaying the size of Vector
        System.out.println("The size is: " + vec_tor.size());
    }
}
```

```
Vector: [10, 15, 30]
The size is: 3
```

Example: void setSize(int size)

```java
import java.util.Vector;
class Main {
  public static void main(String[] args){
  Vector<String> vector = new Vector<String>();
      vector.add("Walter");
      vector.add("Anna");
      vector.add("Hank");
      vector.add("Flynn");
      vector.add("Tom");
      //Setting up the size greater than current size
      vector.setSize(10);
      System.out.println("Vector size: "+vector.size());
      System.out.println("Vector elements: ");
      for(int i=0; i < vector.size(); i++)
      { System.out.println(vector.get(i));
      }
  }
}
```

```
Vector size: 10
Vector elements:
Walter
Anna
Hank
Flynn
Tom
null
null
null
null
null
```

Example: boolean contains(Object o)

```java
import java.util.Vector;
public class Main {
    public static void main(String arg[]) {
        //Create an empty Vector
            Vector<String> vec = new Vector<>();

            //Add elements to the Vector
            vec.add("Python");
            vec.add("Java");
            vec.add("COBOL");
            vec.add("Ruby");

            //Checking if Java is present or not
            if(vec.contains("Java"))
            { System.out.println("Java is present at the index " +vec.indexOf("Java"));
            }
            else
            { System.out.println("Java is not present in the list");
            }
    }
}
```

```
Java is present at the index 1
```

Example: void copyInto(Object[ ] anArray)

```java
import java.util.Vector;
public class Main {
    public static void main(String arg[]) {

        //Create an empty Vector1
        Vector<Integer> vec = new Vector<>(5);

        //Add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);

        Integer[] arr = new Integer[5];
        //copy elements of vector into an array
        vec.copyInto(arr);

        System.out.println("Elements in an array are: ");
        for(Integer num : arr)
        { System.out.println(num);
        }
    }
}
```

```
Elements in an array are:
1
2
3
4
5
```

Example: E elementAt(int index)

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {

        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);

        //Get an element at the specified index
        System.out.println("Element at index 1 is = "+vec.elementAt(1));

        System.out.println("Element at index 3 is = "+vec.elementAt(3));
        }

}
```

```
Element at index 1 is = 2
Element at index 3 is = 4
```

Example: E firstElement()

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {

        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        //Get the first element
        System.out.println("First element of the vector is = "+vec.firstElement());
        }
}
```

```
First element of the vector is = 1
```

Example: void insertElementAt(E obj, int index)

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {

        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
       //Insert the element at 2nd position
        vec.insertElementAt(700, 2);
        System.out.println("Element in vector after insertion = "+vec);
        }
}
```

```
Element in vector after insertion = [1, 2, 700, 3, 4, 5]
```

Example: boolean isEmpty()

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);

        //Verifying if the Vector is empty or not
          System.out.println("Is the Vector empty? = " +vec.isEmpty());

        //Clear the Vector
          vec.clear();

        //Displaying the Vector Again
          System.out.println("Vector after clear(): " +vec);

          //Verifying if the Vector is empty or not
          System.out.println("Is the Vector empty? = " +vec.isEmpty());
}
}
```

```
Is the Vector empty? = false
Vector after clear(): []
Is the Vector empty? = true
```

Example: E lastElement()

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);

        //Obtain the last element of this vector
        System.out.println("The last element of a vector is: " +vec.lastElement());
}
}
```

```
The last element of a vector is: 3
```

Example: void removeAllElements()

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);

        //removeAllElements from the vector
        vec.removeAllElements();
}
}
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Example:void removeElementAt(int index)

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);

        //Remove an element
        vec.removeElement(2);

        //Remove an element at index
        vec.removeElementAt(0);

        //Checking vector and displays the element
        System.out.println("Vector element after removal: " + vec);
}
}
```

```
Vector element after removal: [3]
```

Example: E set(int index, E element)

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        //Create an empty vector
        Vector<Integer> vec = new Vector<>();

        //Add element in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);

     // Using set() method to replace 12 with 21
        System.out.println("The Object that is replaced is: " + vec.set(0, 21));

        //Checking vector and displays the element
        System.out.println("Vector element: " + vec);
}
}
```

```
The Object that is replaced is: 1
Vector element: [21, 2, 3]
```

Example: sublist from vector

```java
import java.util.*;
public class Main {
    public static void main(String arg[]) {
        Vector<String> vct = new Vector<String>();

        //adding elements to the end
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");
        vct.add("Click");

        System.out.println("Actual vector:"+vct);

        List<String> list = vct.subList(2, 4);

        System.out.println("Sub List: "+list);
}
}
```

```
Actual vector:[First, Second, Third, Random, Click]
Sub List: [Third, Random]
```

ArrayList v/s Vector

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.	Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the **Iterator** interface to traverse the elements.	A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements.

Assignment:1 Vectors

- Write a program that accepts students names from command line and stores in vector

1. Set size() of student vector
2. Add 10 records in student vector
3. Print last element in vector
4. Insert new element at index 3
5. Print sublist of vector from index 4 to 8
6. Remove all elements from vector
7. Check if vector is empty()

Methods of Vector Class

1. boolean add(E e) This method appends the specified element to the end of this Vector.

2.void add(int index, E element) This method inserts the specified element at the specified position in this Vector.

3.boolean addAll(Collection<? extends E> c) This method appends all of the elements in the specified Collection to the end of this Vector.

4.boolean addAll(int index, Collection<? extends E> c) This method inserts all of the elements in the specified Collection into this Vector at the specified position.

5.void addElement(E obj) This method adds the specified component to the end of this vector, increasing its size by one.

6.int capacity() This method returns the current capacity of this vector.

7.void clear() This method removes all of the elements from this vector.

8.clone clone() This method returns a clone of this vector.

9.boolean contains(Object o) This method returns true if this vector contains the specified element.

10.boolean containsAll(Collection<?> c) This method returns true if this Vector contains all of the elements in the specified Collection.

Methods of Vector Class(Contd..)

11.void copyInto(Object[ ] anArray) This method copies the components of this vector into the specified array.

12. E elementAt(int index) This method returns the component at the specified index.

13.Enumeration<E> elements() This method returns an enumeration of the components of this vector.

14.void ensureCapacity(int minCapacity) This method increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

15.boolean equals(Object o) This method compares the specified Object with this Vector for equality.

16.E firstElement() This method returns the first component (the item at index 0) of this vector.

17.E get(int index) This method returns the element at the specified position in this Vector.

18int hashCode()This method returns the hash code value for this Vector.

19int indexOf(Object o)This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

20int indexOf(Object o, int index)This method returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.

Methods of Vector Class(Contd..)

21 **void insertElementAt(E obj, int index)** This method inserts the specified object as a component in this vector at the specified index.

22 **boolean isEmpty()** This method tests if this vector has no components.

23 **E lastElement()** This method returns the last component of the vector.

24 **int lastIndexOf(Object o)** This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

25 **int lastIndexOf(Object o, int index)** This method returns the index of the last occurrence of the specified element in this vector, searching backwards from index, or returns -1 if the element is not found.

26 **E remove(int index)** This method removes the element at the specified position in this Vector.

27 **boolean remove(Object o)** This method removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.

28 **boolean removeAll(Collection<?> c)** This method removes from this Vector all of its elements that are contained in the specified Collection.

29 **void removeAllElements()** This method removes all components from this vector and sets its size to zero.

30 **boolean removeElement(Object obj)** This method removes the first occurrence of the argument from this vector.

Methods of Vector Class(Contd..)

31 **void removeElementAt(int index)** This method deletes the component at the specified index.

32 **protected void removeRange(int fromIndex, int toIndex)** This method removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

33 **boolean retainAll(Collection<?> c)** This method retains only the elements in this Vector that are contained in the specified Collection.

34 **E set(int index, E element)** This method replaces the element at the specified position in this Vector with the specified element.

35 **void setElementAt(E obj, int index)** This method sets the component at the specified index of this vector to be the specified object.

36 **void setSize(int newSize)** This method sets the size of this vector.

37 **int size()** This method returns the number of components in this vector.

38 **List <E> subList(int fromIndex, int toIndex)** This method returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.

39 **object[ ] toArray()**

This method returns an array containing all of the elements in this Vector in the correct order.

Methods of Vector Class(Contd..)

40 **<T> T[ ] toArray(T[ ] a)** This method returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.

41 **String toString()** This method returns a string representation of this Vector, containing the String representation of each element.

42 **void trimToSize()** This method trims the capacity of this vector to be the vector's current size.

Example:(2)

```
import java.util.*;
public class VectorDemo
{
 public static void main(String args[])
{
// initial size is 3, increment is 2
 Vector v = new Vector(3, 2);
 System.out.println("Initial size: " + v.size());
System.out.println("Initial capacity: " + v.capacity());
v.addElement(new Integer(1));
v.addElement(new Integer(2));
v.addElement(new Integer(3));
 v.addElement(new Integer(4));
```

Example:(2)

```
System.out.println("Capacity after four additions: " + v.capacity());
v.addElement(new Double(5.45));
System.out.println("Current capacity: " + v.capacity());
 v.addElement(new Double(6.08));
System.out.println("Current capacity: " + v.capacity());
v.addElement(new Float(7.4));
System.out.println("Current capacity: " + v.capacity());
if(v.contains(new Integer(3)))
 System.out.println("Vector contains 3.");
// enumerate the elements in the vector.
Enumeration vEnum = v.elements();
System.out.println("\nElements in vector:");
while(vEnum.hasMoreElements())
System.out.print(vEnum.nextElement() + " ");} }
```

Assignment:2 Vectors

- Write a program that accepts a shopping list of five items from the command line and stores them in a vector.
- Modify the program to accomplish the following:-

1. To delete an item in the list.
2. To add an item at a specified position in the list.
3. To add an item at the end of the list.
4. To print the contents of the vector.