AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

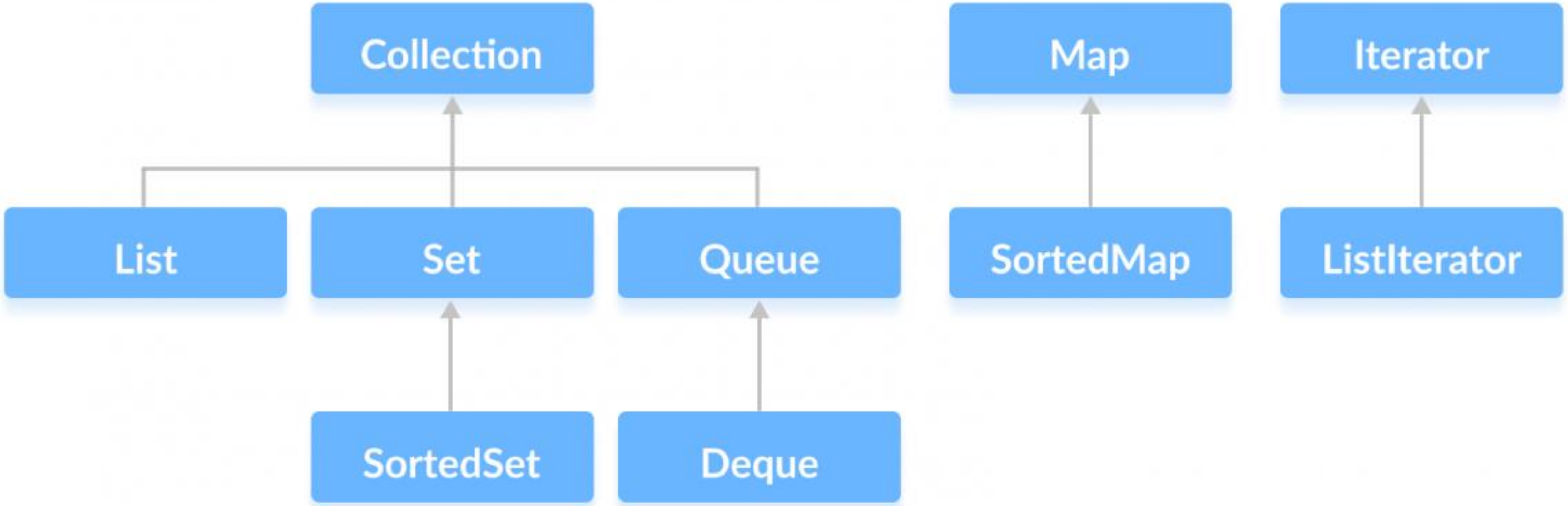
19CSE204
Object Oriented Paradigm
2-0-3-3

Java Collections
List Interface

- The **java.util** package contains one of Java's most powerful subsystems: collections
-
- The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms.

Interfaces of Collections FrameWork

Java Collections Framework



Collection

Map

Iterator

List

Set

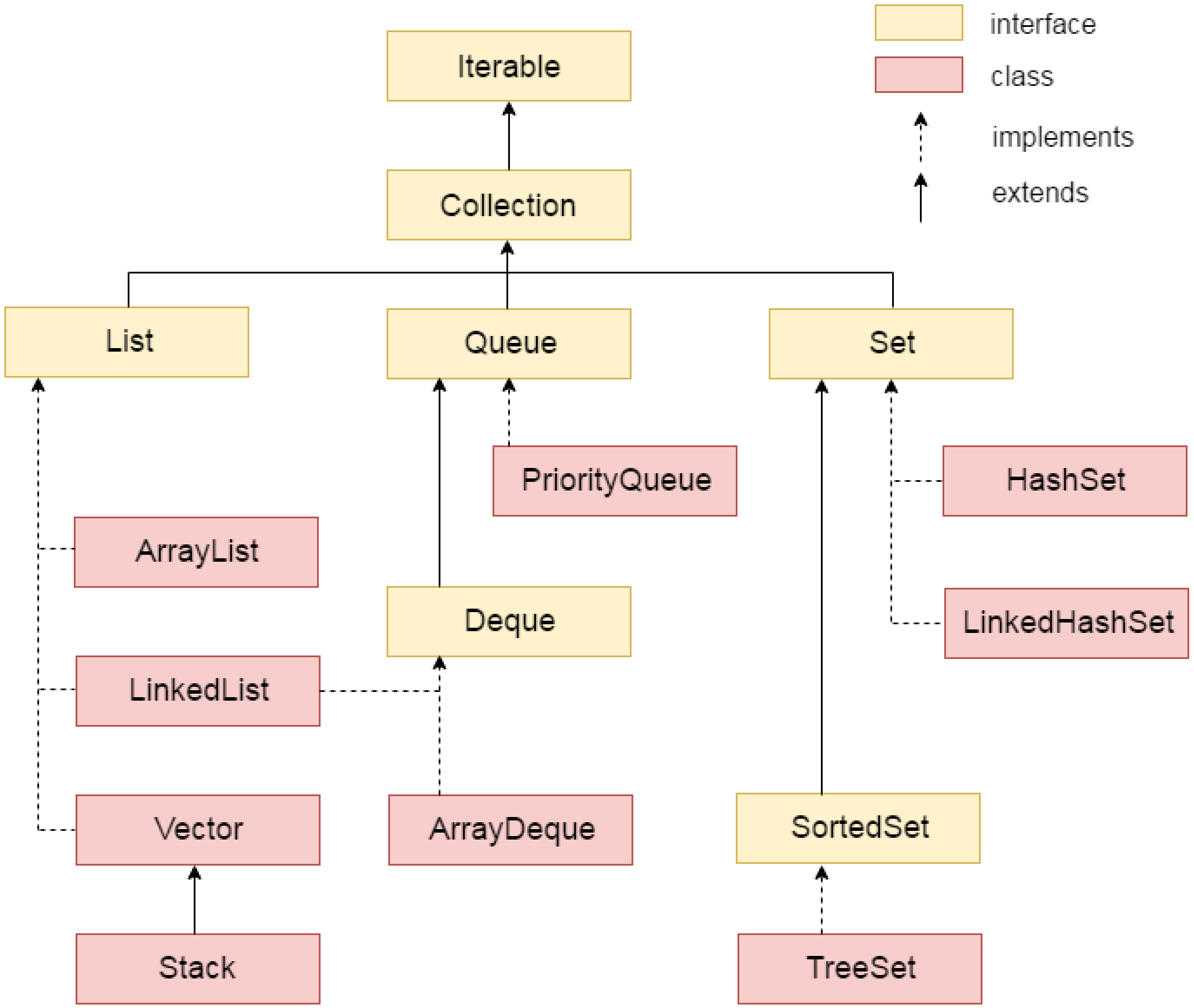
Queue

SortedMap

ListIterator

SortedSet

Deque


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

Java Collections

List Interface

- Abstract List Class
- Abstract Sequential List Class
- Array List
- Vector Class
- Stack Class
- LinkedList Class

Set Interface

- Abstract Set Class
- CopyOnWriteArraySet Class
- EnumSet Class
- ConcurrentHashMap Class
- HashSet Class
- LinkedHashSet Class

SortedSet Interface

- NavigableSet Interface
- TreeSet
- ConcurrentSkipListSet Class

Map Interface

- SortedMap Interface
- NavigableMap Interface
- ConcurrentMap Interface
- TreeMap Class
- AbstractMap Class
- ConcurrentHashMap Class
- EnumMap Class
- HashMap Class
- IdentityHashMap Class
- LinkedHashMap Class
- HashTable Class
- Properties Class

Queue Interface

- Blocking Queue Interface
- AbstractQueue Class
- PriorityQueue Class
- PriorityBlockingQueue Class
- ConcurrentLinkedQueue Class
- ArrayBlockingQueue Class
- DelayQueue Class
- LinkedBlockingQueue Class
- LinkedTransferQueue
- Deque Interface
- BlockingDeque Interface
- ConcurrentLinkedDeque Class
- ArrayDeque Class

- The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms
- The Java collections framework standardizes the way in which groups of objects are handled by your programs

Interfaces of Collections FrameWork

The Java collections framework provides various interfaces

- **List Interface:** The List interface is an ordered collection that allows us to add and remove elements like an array
- **Set Interface :**The Set interface allows us to store elements in different sets similar to the set in mathematics. It cannot have duplicate elements.
- **Queue Interface:** The Queue interface is used when we want to store and access elements in First In, First Out manner.
- **Java Map Interface:** The Map interface allows elements to be stored in key/value pairs. Keys are unique names that can be used to access a particular element in a map. And, each key has a single value associated with it
- **Java Iterator Interface:**The Iterator interface provides methods that can be used to access elements of collections

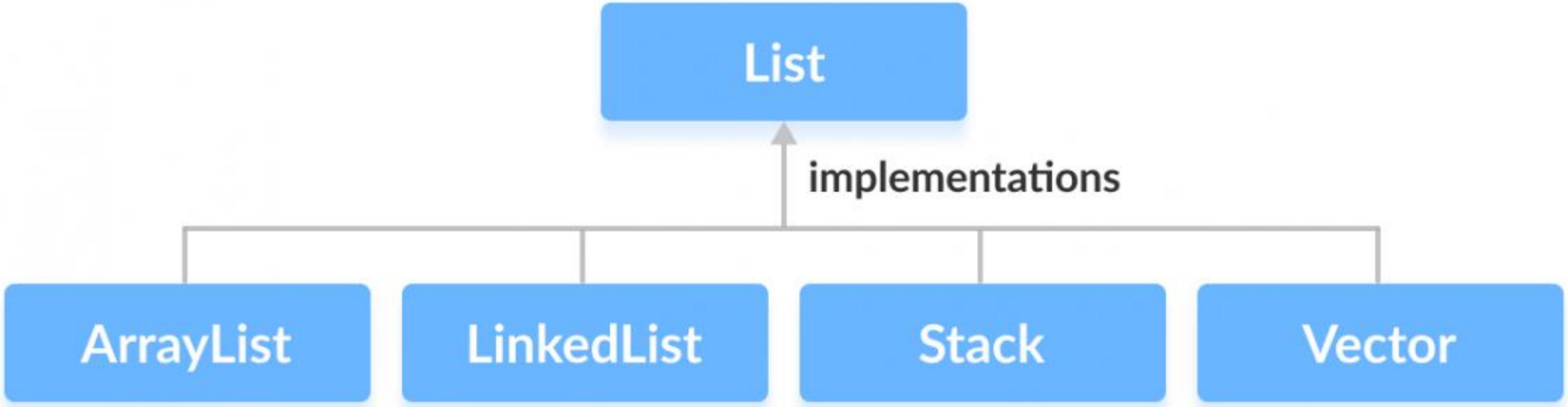
Java List Interface

ArrayList
Vector Class
Stack Class
LinkedList Class

Classes that implement List

- Since List is an interface, we cannot create objects from it.
- In order to use functionalities of the List interface, we can use these classes:

- ArrayList
- LinkedList
- Vector
- Stack

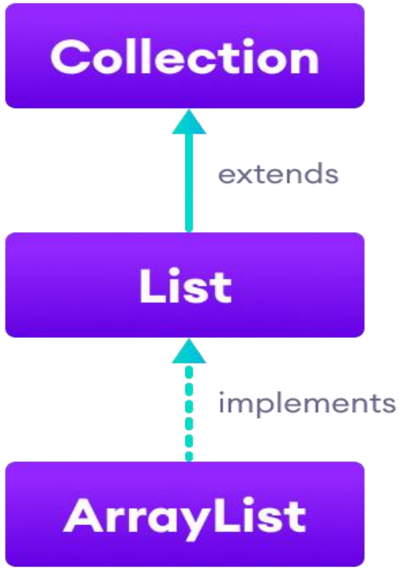


Java ArrayList

Methods of List

- Some of the commonly used methods of the Collection interface that's also available in the List interface are:

- **add()** - adds an element to a list
- **addAll()** - adds all elements of one list to another
- **get()** - helps to randomly access elements from lists
- **iterator()** - returns iterator object that can be used to sequentially access elements of lists
- **set()** - changes elements of lists
- **remove()** - removes an element from the list
- **removeAll()** - removes all the elements from the list
- **clear()** - removes all the elements from the list (more efficient than removeAll())
- **size()** - returns the length of lists
- **toArray()** - converts a list into an array
- **contains()** - returns true if a list contains specified element



Create an ArrayList

- import java.util.ArrayList;
- // ArrayList implementation of List
- ArrayList<String> list1 = new ArrayList<>();
- ArrayList<Integer> sections = new ArrayList <Integer>();
- ArrayList<Student> al=new ArrayList<Student>();

- // LinkedList implementation of List
- List<String> list2 = new LinkedList<>();

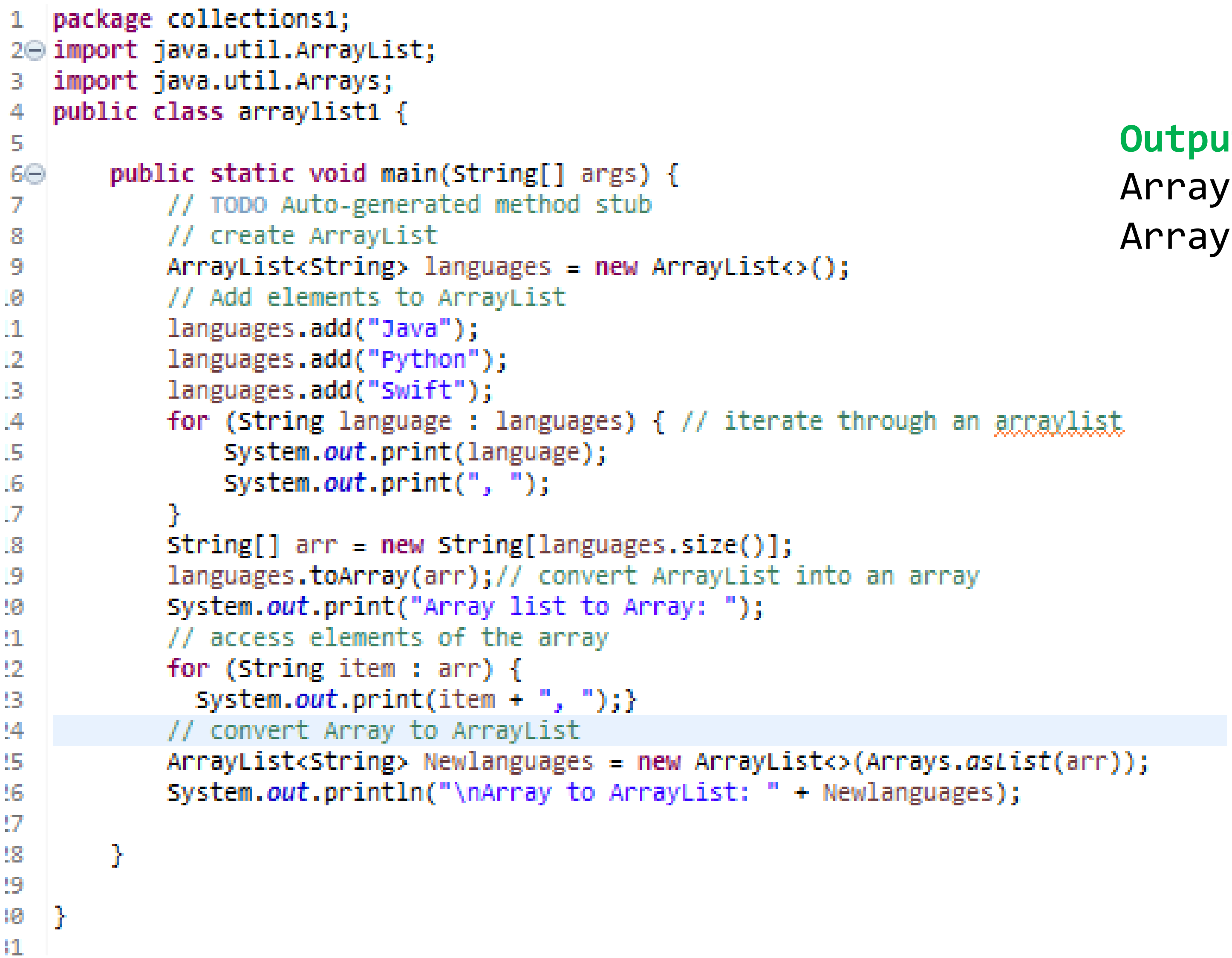
Create, Access, Add, Change, Delete Elements in ArrayList in Java

- `ArrayList<String> languages = new ArrayList<>(); // Create ArrayList`

```java
 // Add elements to ArrayList
languages.add("Java");
languages.add("Python");
languages.add("Swift");
languages.add(1, "JavaScript");
String str = languages.get(2); // Access elements to ArrayList
System.out.print("Element at index 2: " + str);
System.out.println("ArrayList: " + languages);
languages.set(2, "CPP"); // change elements to ArrayList
System.out.println("Modified ArrayList: " + languages);
String str2 = languages.remove(2); // Remove elements to ArrayList
System.out.println("Updated ArrayList: " + languages);
System.out.println("Removed Element: " + str2);
```

```
Output: Element at index 2: Swift
ArrayList: [Java, JavaScript, Python, Swift]
Modified ArrayList: [Java, JavaScript, CPP, Swift]
Updated ArrayList: [Java, JavaScript, Swift]
Removed Element: CPP
```

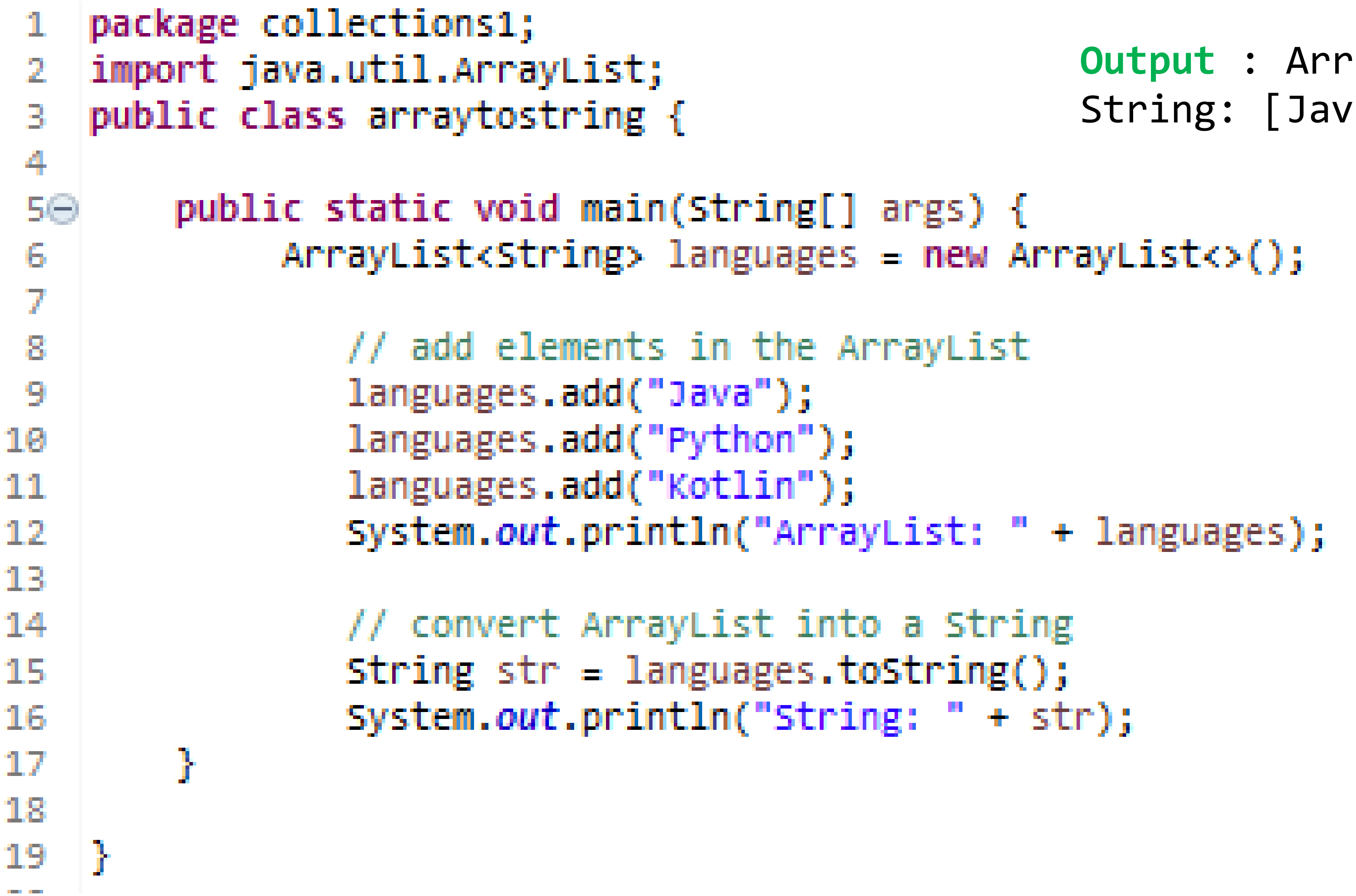
Iterate ArrayList, Convert ArrayList to array and vice versa



```java
package collections1;
import java.util.ArrayList;
import java.util.Arrays;
public class arraylist1 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // create ArrayList
        ArrayList<String> languages = new ArrayList<>();
        // Add elements to ArrayList
        languages.add("Java");
        languages.add("Python");
        languages.add("Swift");
        for (String language : languages) { // iterate through an arraylist
            System.out.print(language);
            System.out.print(", ");
        }
        String[] arr = new String[languages.size()];
        languages.toArray(arr);// convert ArrayList into an array
        System.out.print("Array list to Array: ");
        // access elements of the array
        for (String item : arr) {
          System.out.print(item + ", ");}
        // convert Array to ArrayList
        ArrayList<String> Newlanguages = new ArrayList<>(Arrays.asList(arr));
        System.out.println("\nArray to ArrayList: " + Newlanguages);

    }

}
```

```
Output :Java, Python, Swift,
Array list to Array: Java, Python, Swift,
Array to ArrayList: [Java, Python, Swift]
```

ArrayList to String



```
package collections1;
import java.util.ArrayList;
public class arraytostring {

    public static void main(String[] args) {
         ArrayList<String> languages = new ArrayList<>();

            // add elements in the ArrayList
            languages.add("Java");
            languages.add("Python");
            languages.add("Kotlin");
            System.out.println("ArrayList: " + languages);

            // convert ArrayList into a String
            String str = languages.toString();
            System.out.println("String: " + str);
    }

}
```

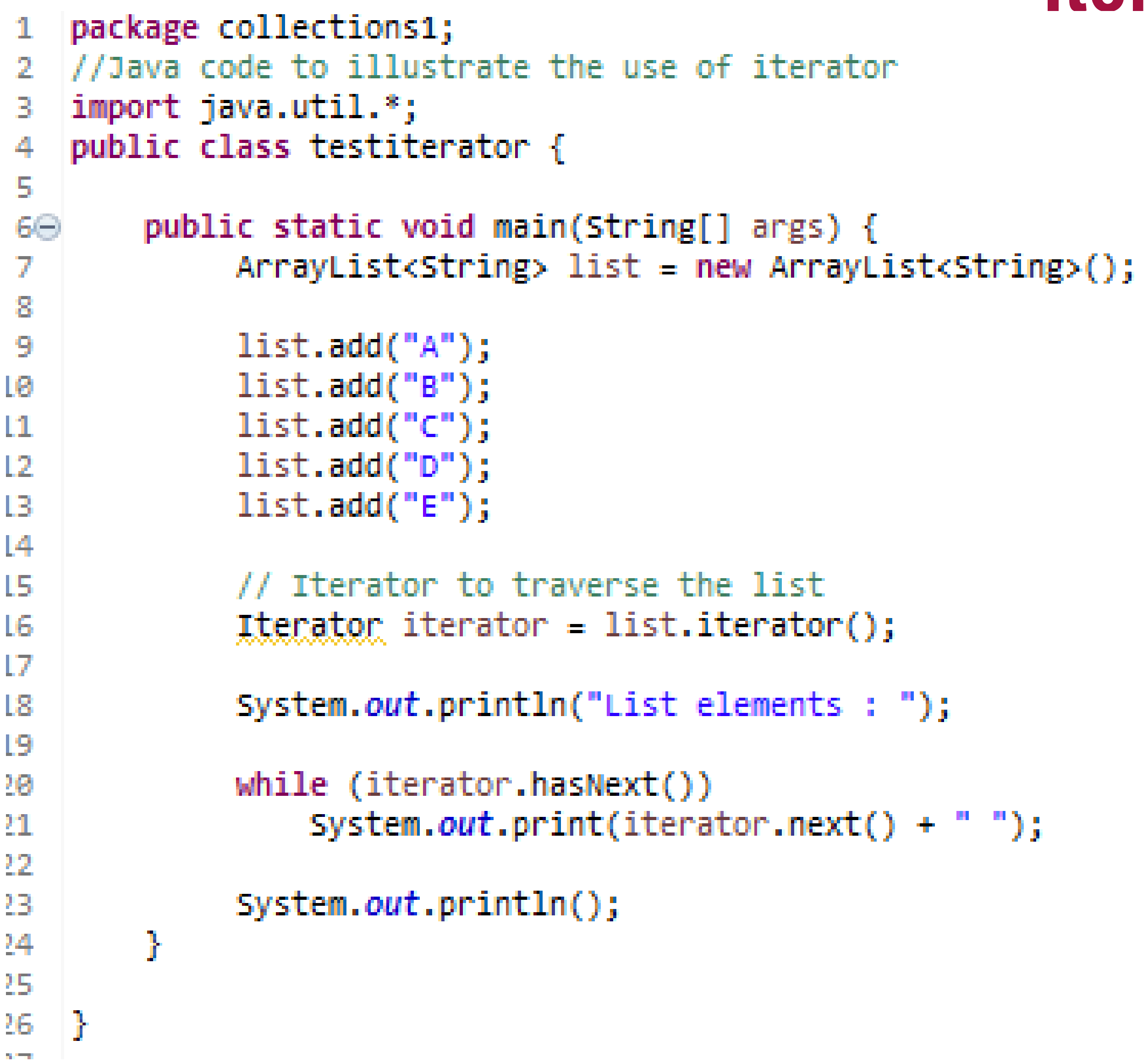
```
Output : ArrayList: [Java, Python, Kotlin]
String: [Java, Python, Kotlin]
```

Iterator to list

- **‘Iterator’** is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection.
  java.util package has **public interface Iterator** and contains three methods:

  1. **boolean hasNext()**: It returns true if Iterator has more element to iterate.
  2. **Object next()**: It returns the next element in the collection until the hasNext()method return true. This method throws ‘NoSuchElementException’ if there is no next element.
  3. **void remove()**: It removes the current element in the collection. This method throws ‘IllegalStateException’ if this function is called before next( ) is invoked.

Iterator to list :Sample program



```java
package collections1;
//Java code to illustrate the use of iterator
import java.util.*;
public class testiterator {

    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();

        list.add("A");
        list.add("B");
        list.add("C");
        list.add("D");
        list.add("E");

        // Iterator to traverse the list
        Iterator iterator = list.iterator();

        System.out.println("List elements : ");

        while (iterator.hasNext())
            System.out.print(iterator.next() + " ");

        System.out.println();
    }

}
```

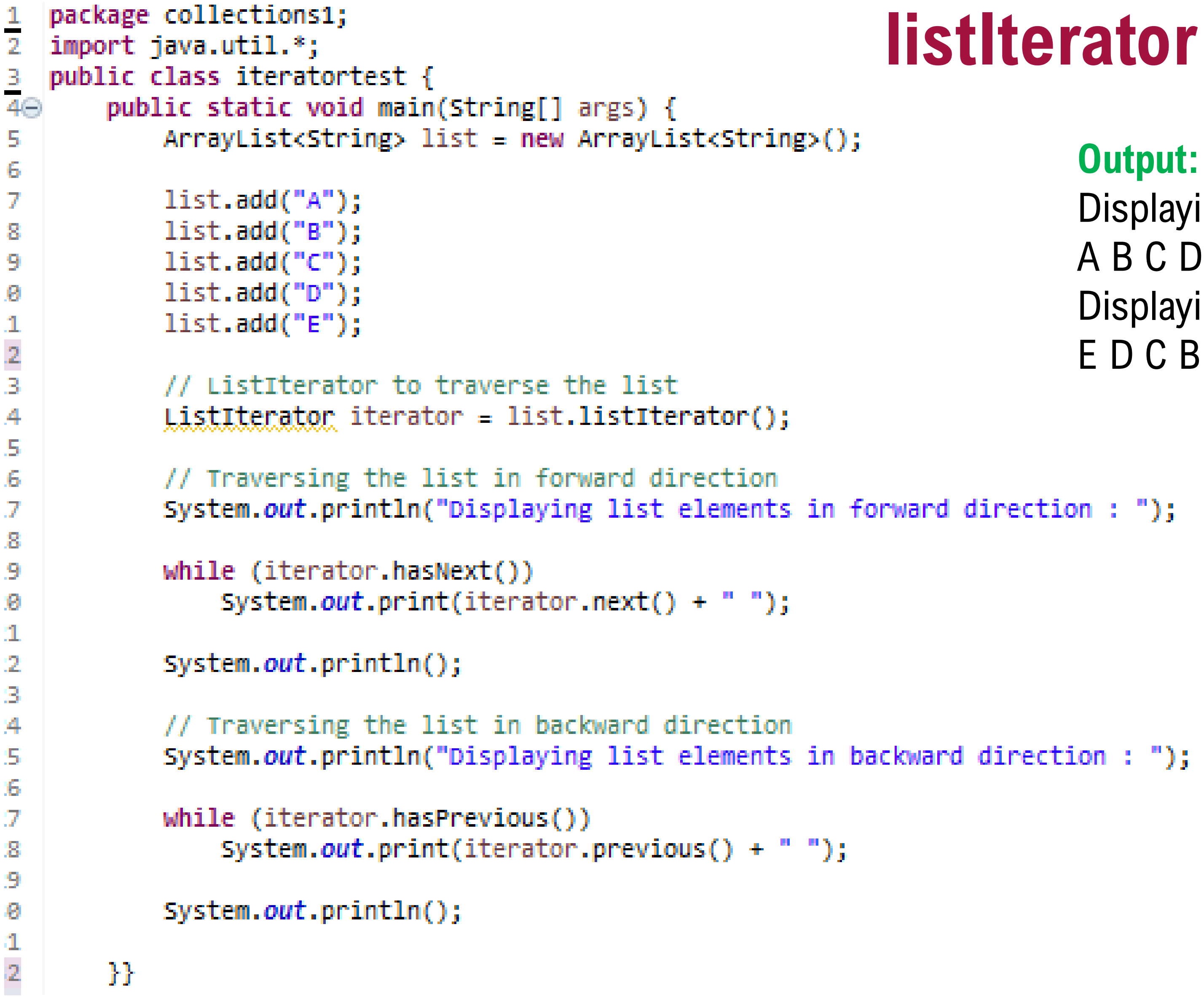
Output

```
List elements :
A B C D E
```

ListIterator

- ‘ListIterator’ in Java is an Iterator which allows users to traverse Collection in both direction. It contains the following methods:
  1. **void add(Object object)**: It inserts object immediately before the element that is returned by the next( ) function.
  2. **boolean hasNext( )**: It returns true if the list has a next element.
  3. **boolean hasPrevious( )**: It returns true if the list has a previous element.
  4. **Object next( )**: It returns the next element of the list. It throws ‘NoSuchElementException’ if there is no next element in the list.
  5. **Object previous( )**: It returns the previous element of the list. It throws ‘NoSuchElementException’ if there is no previous element.
  6. **void remove( )**: It removes the current element from the list. It throws ‘IllegalStateException’ if this function is called before next( ) or previous( ) is invoked.

listIterator :Sample program



```java
package collections1;
import java.util.*;
public class iteratortest {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();

        list.add("A");
        list.add("B");
        list.add("C");
        list.add("D");
        list.add("E");

        // ListIterator to traverse the list
        ListIterator iterator = list.listIterator();

        // Traversing the list in forward direction
        System.out.println("Displaying list elements in forward direction : ");

        while (iterator.hasNext())
            System.out.print(iterator.next() + " ");

        System.out.println();

        // Traversing the list in backward direction
        System.out.println("Displaying list elements in backward direction : ");

        while (iterator.hasPrevious())
            System.out.print(iterator.previous() + " ");

        System.out.println();

    }}
```

Output:

```
Displaying list elements in forward direction :
A B C D E
Displaying list elements in backward direction :
E D C B A
```

Methods	Descriptions
size()	Returns the length of the arraylist.
sort()	Sort the arraylist elements.
clone()	Creates a new arraylist with the same element, size, and capacity.
contains()	Searches the arraylist for the specified element and returns a boolean result.
ensureCapacity()	Specifies the total element the arraylist can contain.
isEmpty()	Checks if the arraylist is empty.
indexOf()	Searches a specified element in an arraylist and returns the index of the element.
Java ArrayList removeAll()	
Java ArrayList clear()	

Java Vector

ArrayList vs Vector :

- The Vector class synchronizes each individual operation.
  - This means whenever we want to perform some operation on vectors, the Vector class automatically applies a lock to that operation.
- It is because when one thread is accessing a vector, and at the same time another thread tries to access it, an exception called ConcurrentModificationException is generated. Hence, this continuous use of lock for each operation makes vectors less efficient.

- However, in array lists, methods are not synchronized. Instead, it uses the Collections.synchronizedList() method that synchronizes the list as a whole.

Note: It is recommended to use ArrayList in place of Vector because vectors are not threadsafe and are less efficient.

Creating a vector

- Here is how we can create vectors in Java.

- **Vector<Type> vector = new Vector<>();**
- Here, Type indicates the type of a linked list. For example,

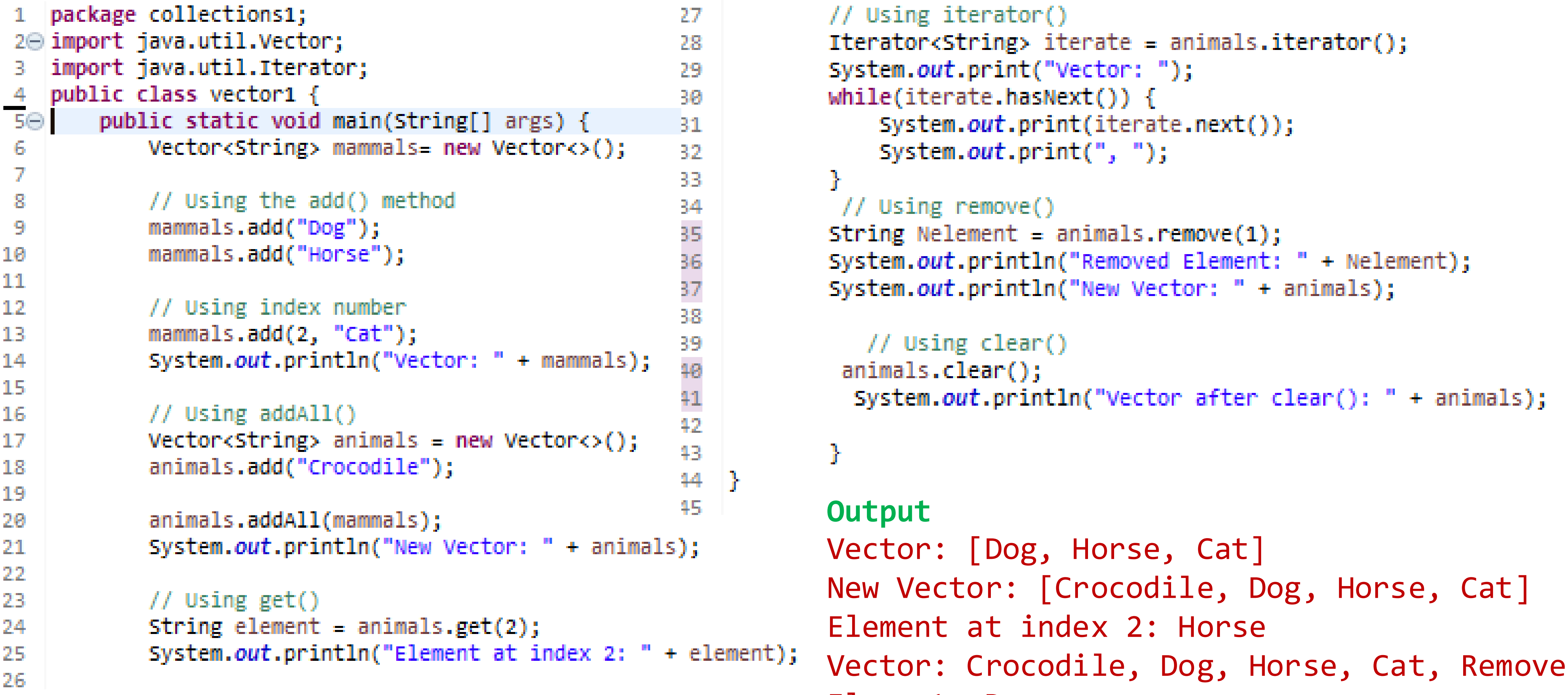
- **// create Integer type linked list**
- **Vector<Integer> vector= new Vector<>();**

- **// create String type linked list**
- **Vector<String> vector= new Vector<>();**

Vector Methods

- **add(element)** - adds an element to vectors
- **add(index, element)** - adds an element to the specified position
- **addAll(vector)** - adds all elements of a vector to another vector
- **get(index)** - returns an element specified by the index
- **iterator()** - returns an iterator object to sequentially access vector elements
- **remove(index)** - removes an element from specified position
- **removeAll()** - removes all the elements
- **clear()** - removes all elements. It is more efficient than removeAll()

Java Vector :Add, Get,Iterator,remove,clear



```java
package collections1;
import java.util.Vector;
import java.util.Iterator;
public class vector1 {
    public static void main(String[] args) {
        Vector<String> mammals= new Vector<>();

        // Using the add() method
        mammals.add("Dog");
        mammals.add("Horse");

        // Using index number
        mammals.add(2, "Cat");
        System.out.println("Vector: " + mammals);

        // Using addAll()
        Vector<String> animals = new Vector<>();
        animals.add("Crocodile");

        animals.addAll(mammals);
        System.out.println("New Vector: " + animals);

        // Using get()
        String element = animals.get(2);
        System.out.println("Element at index 2: " + element);

        // Using iterator()
        Iterator<String> iterate = animals.iterator();
        System.out.print("Vector: ");
        while(iterate.hasNext()) {
            System.out.print(iterate.next());
            System.out.print(", ");
        }
         // Using remove()
        String Nelement = animals.remove(1);
        System.out.println("Removed Element: " + Nelement);
        System.out.println("New Vector: " + animals);

           // Using clear()
         animals.clear();
          System.out.println("Vector after clear(): " + animals);

        }
}
```

Output

```
Vector: [Dog, Horse, Cat]
New Vector: [Crocodile, Dog, Horse, Cat]
Element at index 2: Horse
Vector: Crocodile, Dog, Horse, Cat, Removed
Element: Dog
New Vector: [Crocodile, Horse, Cat]
Vector after clear(): []
```

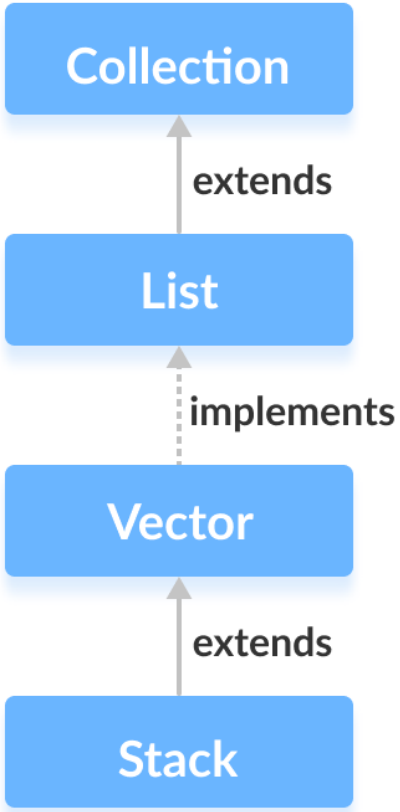
Other vector methods

- **set()** changes an element of the vector
- **size()** returns the size of the vector
- **toArray()** converts the vector into an array
- **toString()** converts the vector into a String
- **contains()** searches the vector for specified element and returns a boolean result

Java Stack Class

Java Stack Class The Stack class extends the Vector class.

- The Java collections framework has a class named Stack that provides the functionality of the stack data structure.



- **push()** To add an element to the top of the stack
- **pop()** To remove an element from the top of the stack
- **peek()** returns an object from the top of the stack
- **search()** To search an element in the stack
- **empty()** To check whether a stack is empty or not

Create a stack

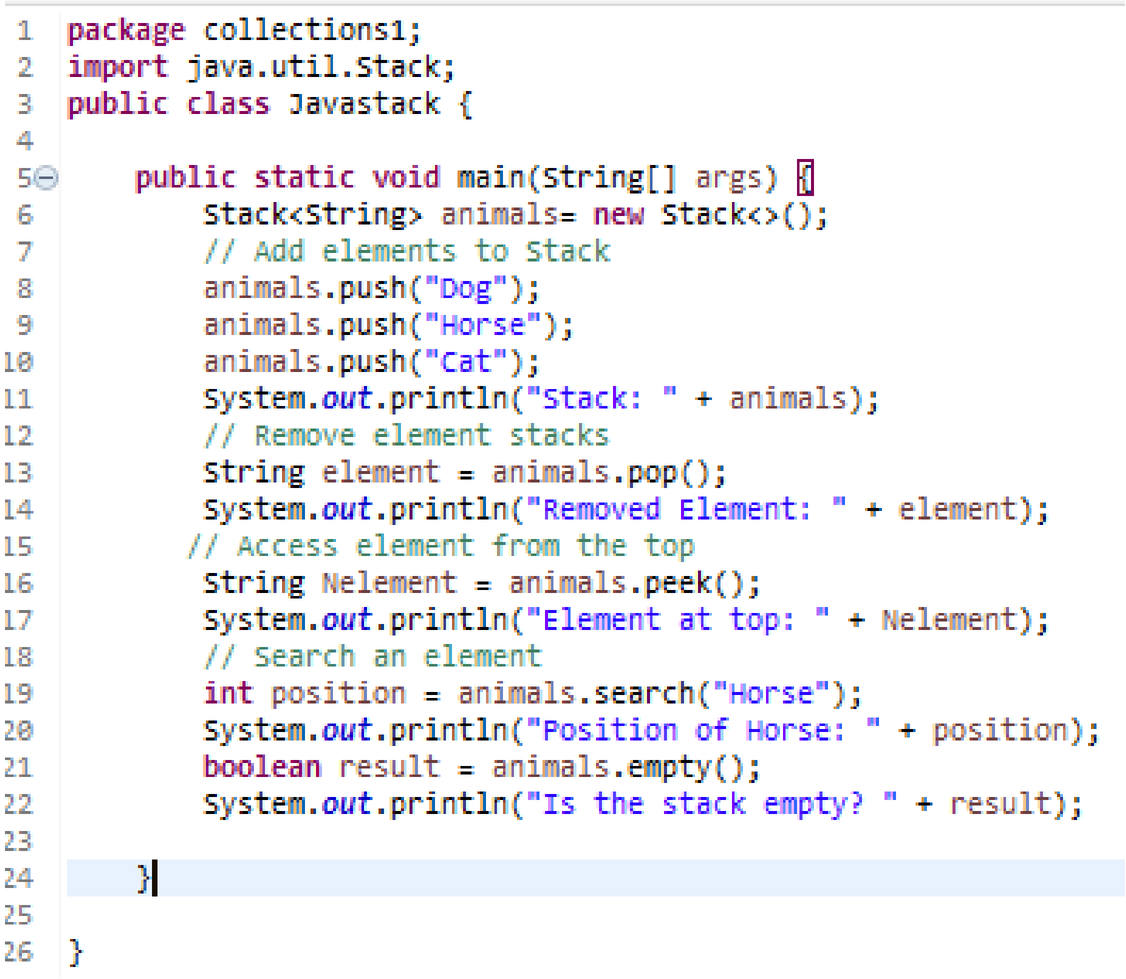
- In order to create a stack, we must import the **java.util.Stack** package first.

- Stack<Type> stacks = new Stack<>();
- Here, Type indicates the stack's type. For example,

- **// Create Integer type stack**
- Stack<Integer> stacks = new Stack<>();

- **// Create String type stack**
- Stack<String> stacks = new Stack<>();

Java Stack

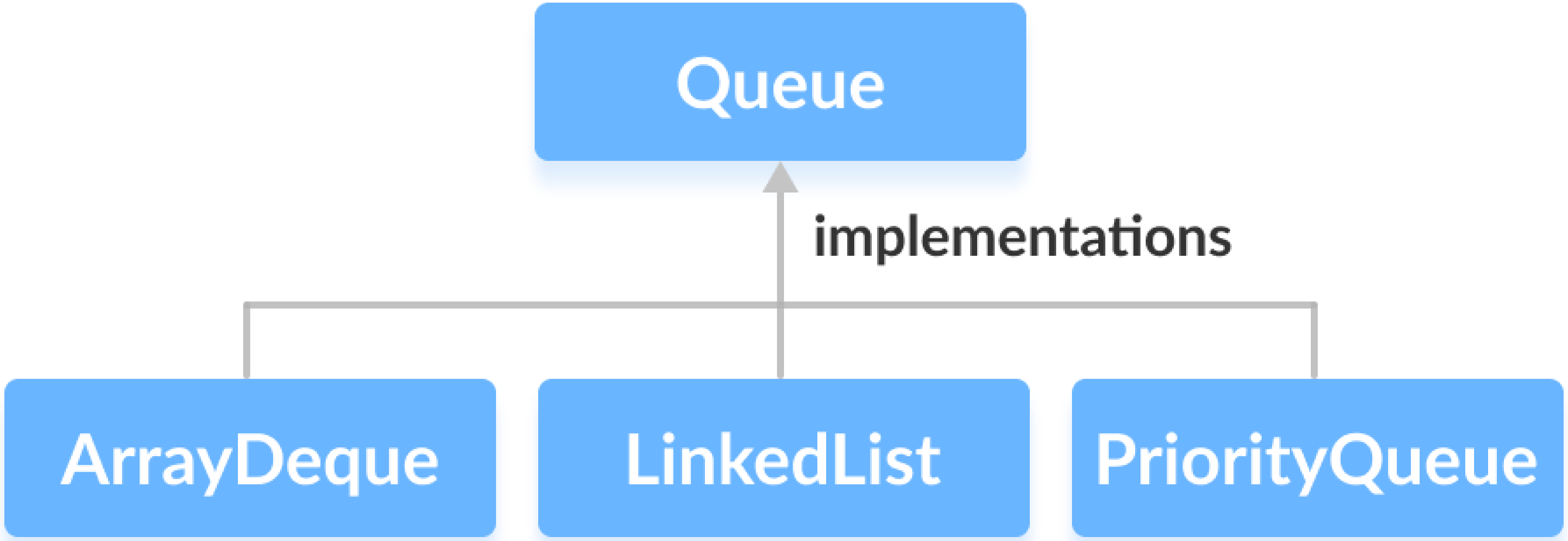


```java
package collections1;
import java.util.Stack;
public class Javastack {

    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();
        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");
        System.out.println("Stack: " + animals);
        // Remove element stacks
        String element = animals.pop();
        System.out.println("Removed Element: " + element);
       // Access element from the top
        String Nelement = animals.peek();
        System.out.println("Element at top: " + Nelement);
        // Search an element
        int position = animals.search("Horse");
        System.out.println("Position of Horse: " + position);
        boolean result = animals.empty();
        System.out.println("Is the stack empty? " + result);

    }

}
```

Output:

```
Stack: [Dog, Horse, Cat]
Removed Element: Cat
Element at top: Horse
Position of Horse: 1
Is the stack empty? false
```

Java Queue



You will learn more about these in your data Structures Course

Namah Shivaya