Week 14: Lists and Iterators

4009 DATA STRUCTURES & ALGORITHM (DSA)

Learning Outcomes

- ❖Create, manipulate and understand the data structure of Arraylists
- ❖Define and understand the list data structure
- ❖Differentiate between a traditional loop, an advanced loop and the iterator.
- ❖Identity and implement iterator statements with LinkedList or Arraylist in Java
- ❖Implement an Arraylist in Java
- ❖Manipulate Arraylist coding in Java and use of Java’s inbuilt methods

Content Overview

- The List ADT
- Application of Lists in the real world
- The list abstract data type (ADT)
- Example of list operations
- A simple version of the List Interface
- Different Lists in Java Programming
- Array Lists
- A simple ArrayList implementation
- A generic ArrayList implementation
- Iterators
- Iterator Code breakdown
- Explanation of Iterator statement
- Explanation of the traditional (for loop method)
- Using the advanced looping method (For-Each loop)

Data Structure Trade-offs

Data structures	Advantages	Disadvantages
Array	Quick insertion, very fast access to index known	Slow search, slow deletion, fixed size
Ordered Array	Quicker Search than unsorted array	Slow insertion and deletion, fixed size
Stack	Provide Last-in First-out access	Slow access to other items
Queue	Provide First-in First out access	Slow access to other items
Linked List	Quick Insertion, Quick deletion	Slow search
Binary Tree	Quick search, insertion, deletion (if tree balanced)	Deletion algorithm is complex
Hash Table	Very fast access if key known, fast insertion	Slow deletion, access slow if key not known, inefficient memory usage
Heap	Fast insertion, deletion, access to large item	Slow access to other items
Graphs	Model real world situations	Some algorithms are slow and complex

The List ADT

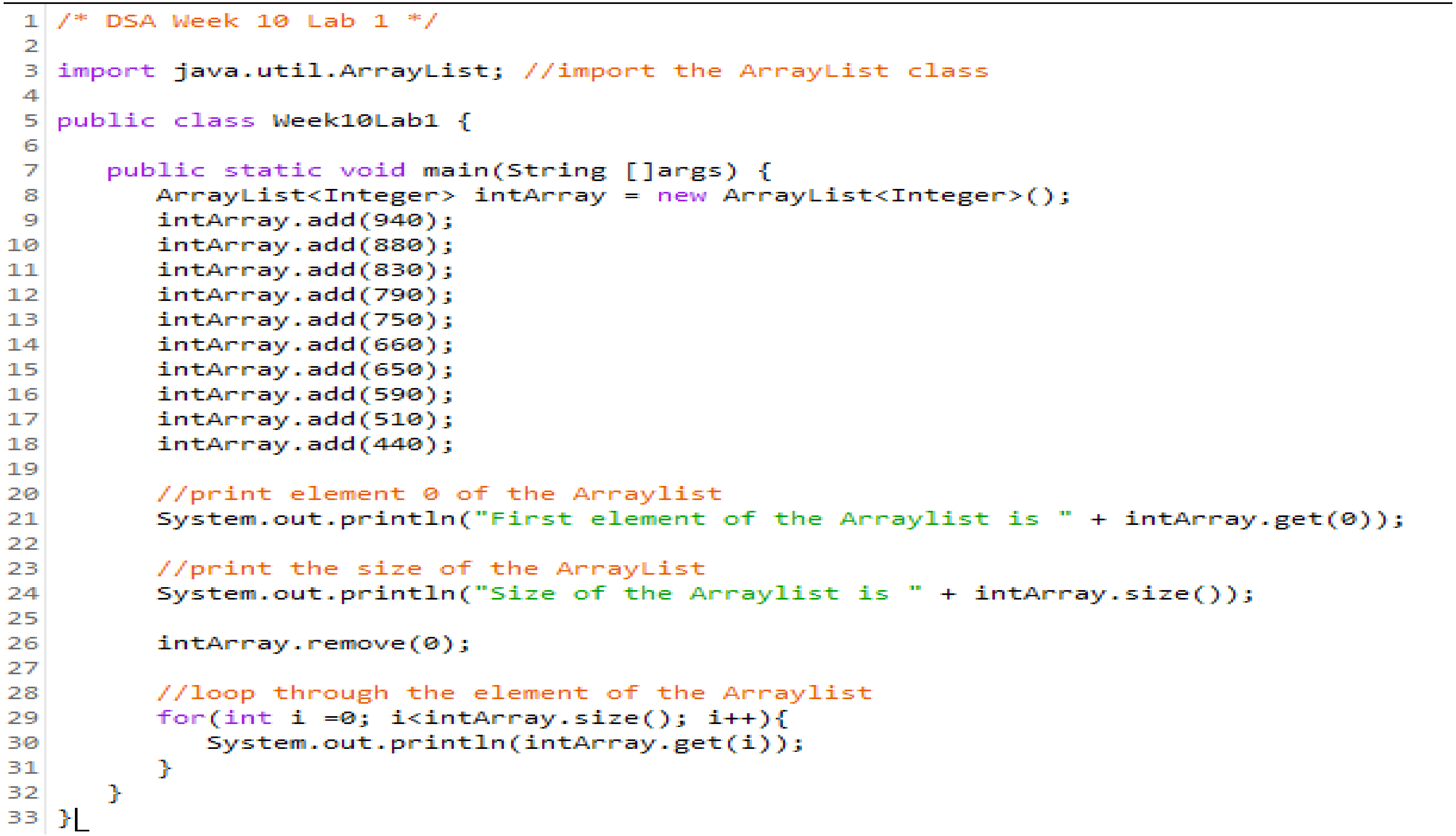
- The list is abstract datatype which represents a linear order sequence of elements. The list has support of adding and removing elements.
- The list data structure is efficient for data storage, data retrieval and where elements are accessed using an index position.

Application of Lists in the real world

- Frequently used in applications that require multiple daily transactions or operations.
- Applications that use the insert, delete and update methods or data handing prefer the data structure lists.

The list abstract data type (ADT)

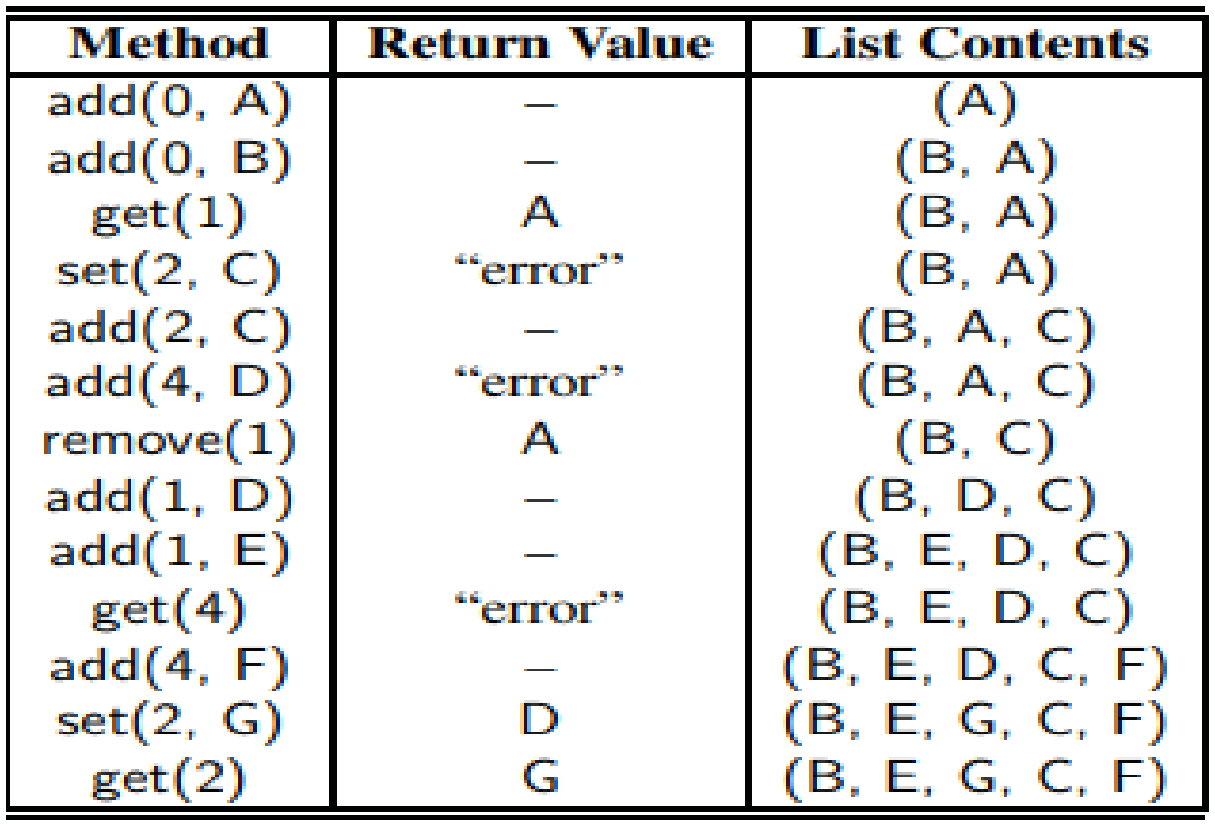
- The list abstract data type (ADT) that supports the following index-based methods:
  - **size():** returns the number of elements in the list.
  - **isEmpty( ):** Returns a boolean indicating whether the list is empty.
  - **get(i):** Returns the element of the list having index i:
  - **set(i, e):** Replaces the element at index i with e and returns the old element that was replaced.
  - **add(i, e):** Inserts a new element e into the list so that it has index i, moving all subsequent elements one index later in the list.
  - **Remove(i):** Removes and returns the element at index i, moving all subsequent elements one index earlier in the list.
- Note: when using the methods errors occur if i is not in range of the list.



```java
/* DSA Week 10 Lab 1 */

import java.util.ArrayList; //import the ArrayList class

public class Week10Lab1 {

    public static void main(String []args) {
        ArrayList<Integer> intArray = new ArrayList<Integer>();
        intArray.add(940);
        intArray.add(880);
        intArray.add(830);
        intArray.add(790);
        intArray.add(750);
        intArray.add(660);
        intArray.add(650);
        intArray.add(590);
        intArray.add(510);
        intArray.add(440);

        //print element 0 of the Arraylist
        System.out.println("First element of the Arraylist is " + intArray.get(0));

        //print the size of the ArrayList
        System.out.println("Size of the Arraylist is " + intArray.size());

        intArray.remove(0);

        //loop through the element of the Arraylist
        for(int i =0; i<intArray.size(); i++){
            System.out.println(intArray.get(i));
        }
    }
}
```

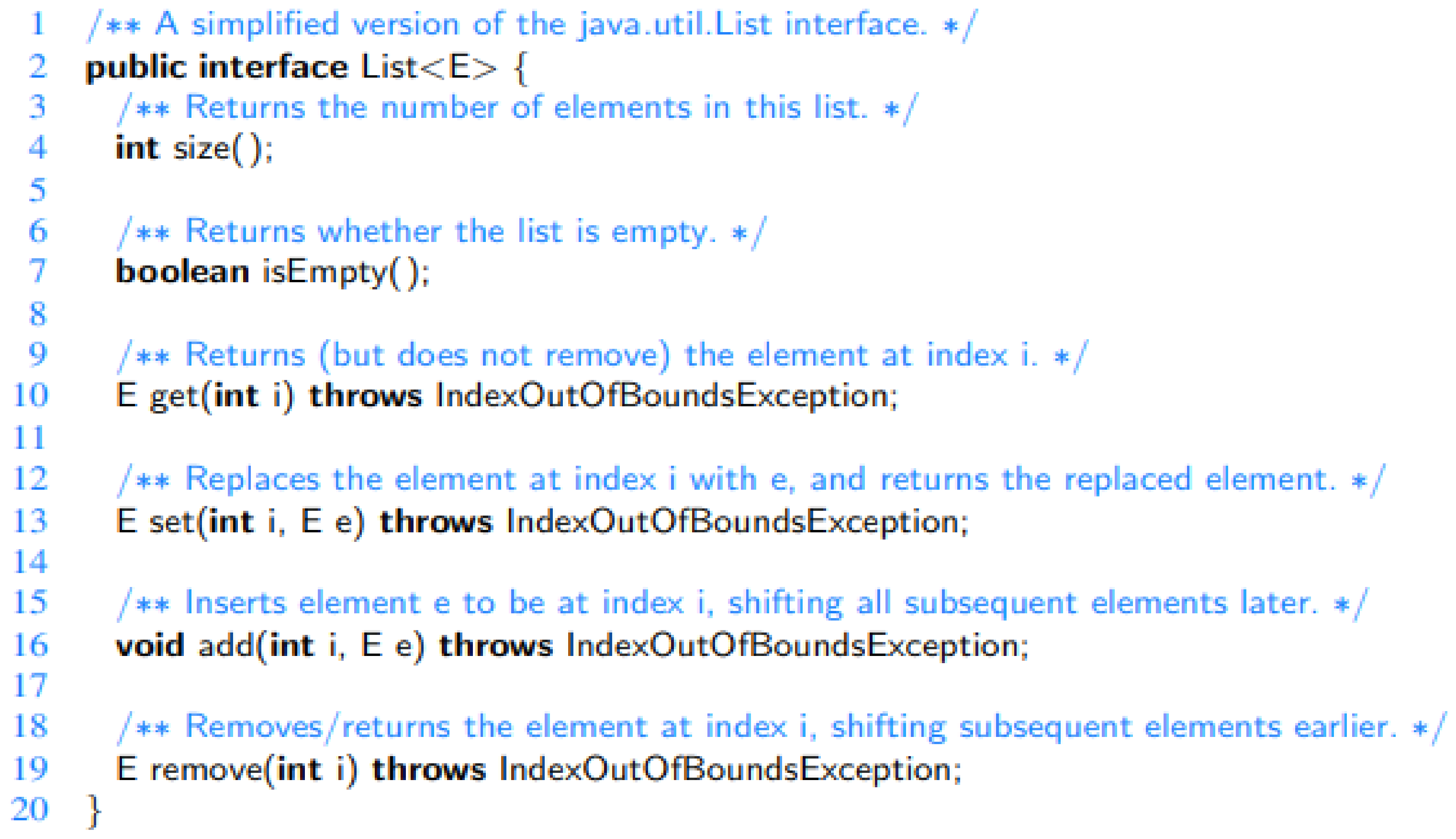
Example of list operations

- The following table shows a series of list operations and their effects



Method	Return Value	List Contents
add(0, A)	–	(A)
add(0, B)	–	(B, A)
get(1)	A	(B, A)
set(2, C)	"error"	(B, A)
add(2, C)	–	(B, A, C)
add(4, D)	"error"	(B, A, C)
remove(1)	A	(B, C)
add(1, D)	–	(B, D, C)
add(1, E)	–	(B, E, D, C)
get(4)	"error"	(B, E, D, C)
add(4, F)	–	(B, E, D, C, F)
set(2, G)	D	(B, E, G, C, F)
get(2)	G	(B, E, G, C, F)

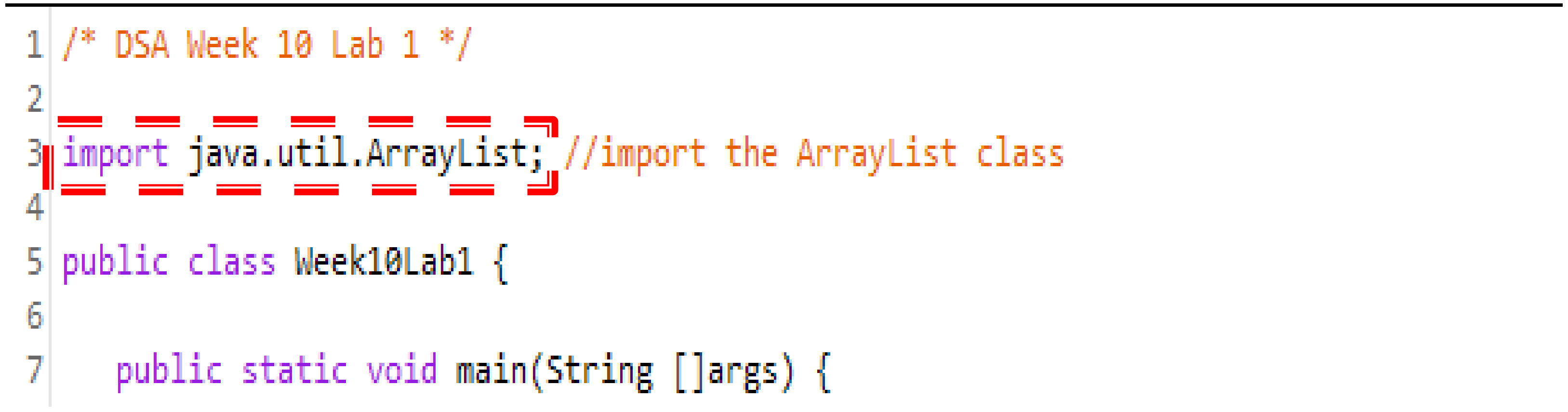
A simple version of the List Interface



```
/** A simplified version of the java.util.List interface. */
public interface List<E> {
  /** Returns the number of elements in this list. */
  int size();

  /** Returns whether the list is empty. */
  boolean isEmpty();

  /** Returns (but does not remove) the element at index i. */
  E get(int i) throws IndexOutOfBoundsException;

  /** Replaces the element at index i with e, and returns the replaced element. */
  E set(int i, E e) throws IndexOutOfBoundsException;

  /** Inserts element e to be at index i, shifting all subsequent elements later. */
  void add(int i, E e) throws IndexOutOfBoundsException;

  /** Removes/returns the element at index i, shifting subsequent elements earlier. */
  E remove(int i) throws IndexOutOfBoundsException;
}
```

java.util.ArrayList class

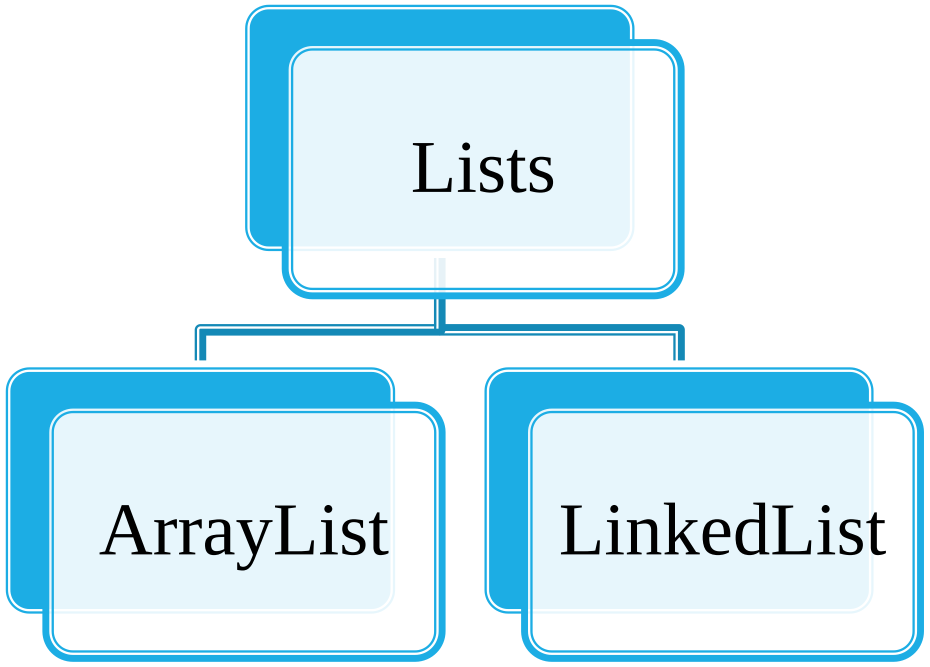
- Like Arrays and LinkedList, Java provides a class named java.util.ArrayList that implements the data structure ArrayList
- NOTE: before creating your program’s class, you must import java.util.ArrayList; class in order to use and manipulate Stacks in Java.



```
/* DSA Week 10 Lab 1 */

import java.util.ArrayList; //import the ArrayList class

public class Week10Lab1 {

   public static void main(String []args) {
```

Different Lists in Java Programming

- Under the lists data structure there are several types of lists. In this course you will learn Arraylist and LinkedList.



Array concept revised

- Location and description of an element within an array are easily found using an index.
- The first element in the array is index 0 and the last element is index n -1, assume that n denotes the total number of element.

High Scores	940	880	830	790	750	660	650	590	510	440
	0	1	2	3	4	5	6	7	8	9

indices

Array Lists (1/2)

- Implementing a list can be done using Arrays.
- ***For example***, using A, where A[i] stores a reference to element with the index i. Here we assume the array is a fixed capacity or size.
- However, array-based list has unbounded capacity until Arrays data structure. Such as unbound list is an arraylist in Java.
- With a representation based on an array A, the get(i) and set(i, e) methods are easy to implement by accessing A[i] (assuming i is a legitimate index).
- Methods add(i, e) and remove(i) are more time consuming, as they require shifting elements up or down to maintain our rule of always storing an element whose list index is i at index i of the array. (See Figure 7.1.) Our initial implementation of the ArrayList class follows in Code Fragments 7.2 and 7.3.

Array Lists (2/2)

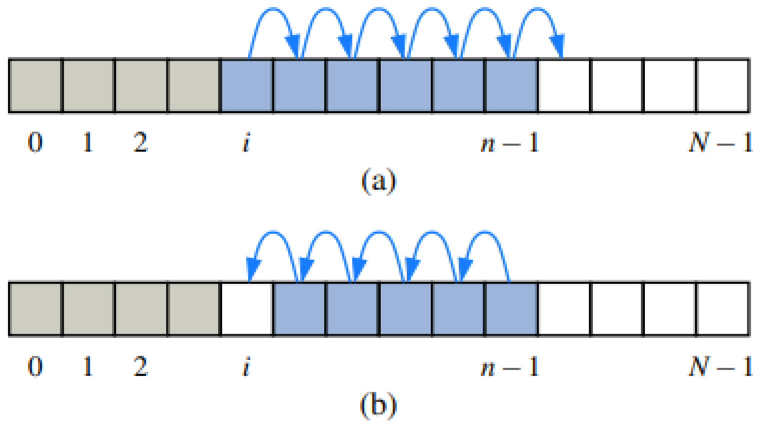


Figure 7.1: Array-based implementation of an array list that is storing *n* elements: (a) shifting up for an insertion at index *i*; (b) shifting down for a removal at index *i*.

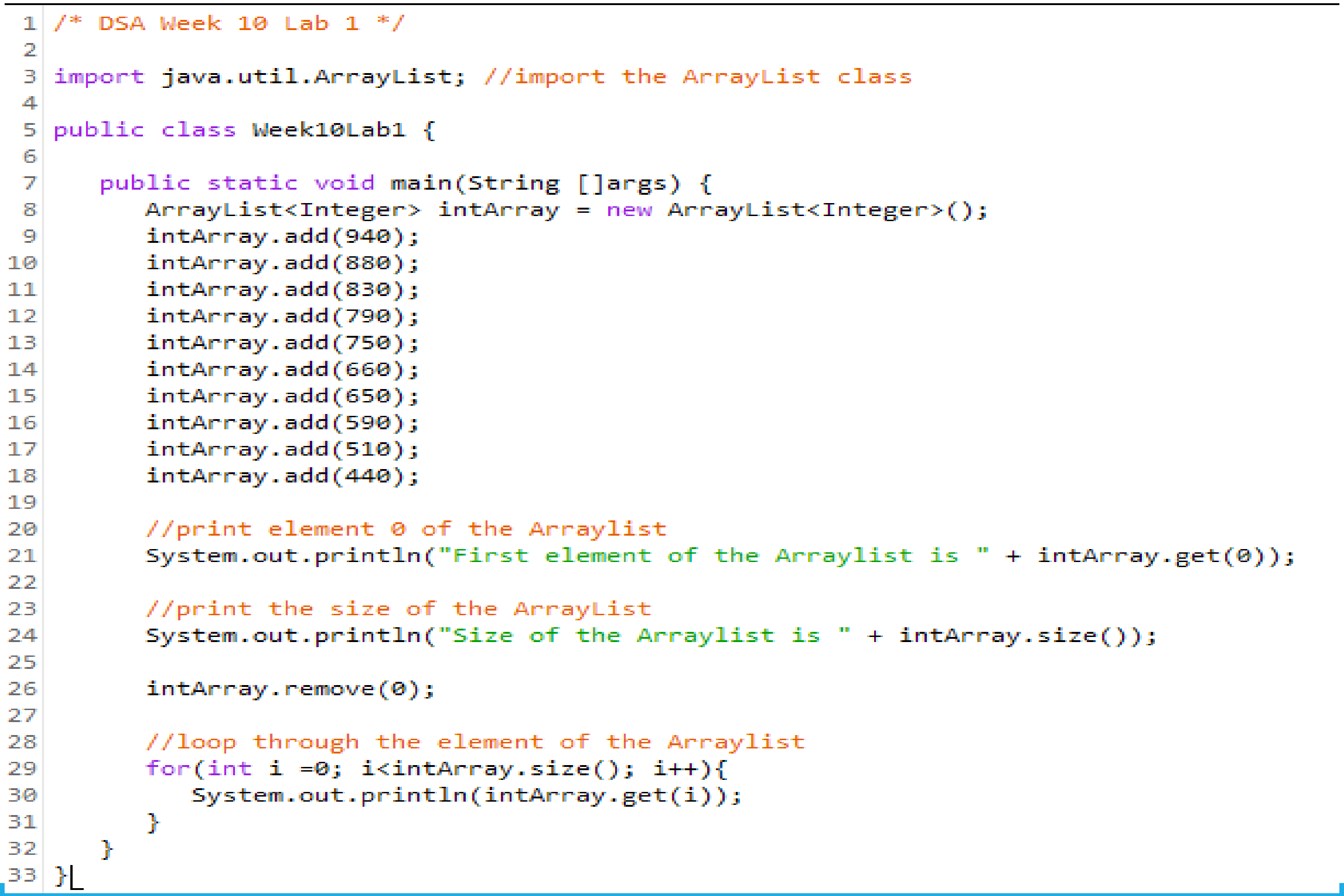
```java
public class ArrayList<E> implements List<E> {
  // instance variables
  public static final int CAPACITY=16;     // default array capacity
  private E[ ] data;                       // generic array used for storage
  private int size = 0;                    // current number of elements
  // constructors
  public ArrayList( ) { this(CAPACITY); }  // constructs list with default capacity
  public ArrayList(int capacity) {         // constructs list with given capacity
    data = (E[ ]) new Object[capacity];    // safe cast; compiler may give warning
  }
```

Code Fragment 7.2: An implementation of a simple ArrayList class with bounded capacity. (Continues in Code Fragment 7.3.)

```java
  // public methods
  /** Returns the number of elements in the array list. */
  public int size( ) { return size; }
  /** Returns whether the array list is empty. */
  public boolean isEmpty( ) { return size == 0; }
  /** Returns (but does not remove) the element at index i. */
  public E get(int i) throws IndexOutOfBoundsException {
    checkIndex(i, size);
    return data[i];
  }
  /** Replaces the element at index i with e, and returns the replaced element. */
  public E set(int i, E e) throws IndexOutOfBoundsException {
    checkIndex(i, size);
    E temp = data[i];
    data[i] = e;
    return temp;
  }
  /** Inserts element e to be at index i, shifting all subsequent elements later. */
  public void add(int i, E e) throws IndexOutOfBoundsException,
                                          IllegalStateException {
    checkIndex(i, size + 1);
    if (size == data.length)                    // not enough capacity
      throw new IllegalStateException("Array is full");
    for (int k=size−1; k >= i; k−−)             // start by shifting rightmost
      data[k+1] = data[k];
    data[i] = e;                                // ready to place the new element
    size++;
  }
  /** Removes/returns the element at index i, shifting subsequent elements earlier. */
  public E remove(int i) throws IndexOutOfBoundsException {
    checkIndex(i, size);
    E temp = data[i];
    for (int k=i; k < size−1; k++)              // shift elements to fill hole
      data[k] = data[k+1];
    data[size−1] = null;                        // help garbage collection
    size−−;
    return temp;
  }
  // utility method
  /** Checks whether the given index is in the range [0, n−1]. */
  protected void checkIndex(int i, int n) throws IndexOutOfBoundsException {
    if (i < 0 || i >= n)
      throw new IndexOutOfBoundsException("Illegal index: " + i);
  }
}
```

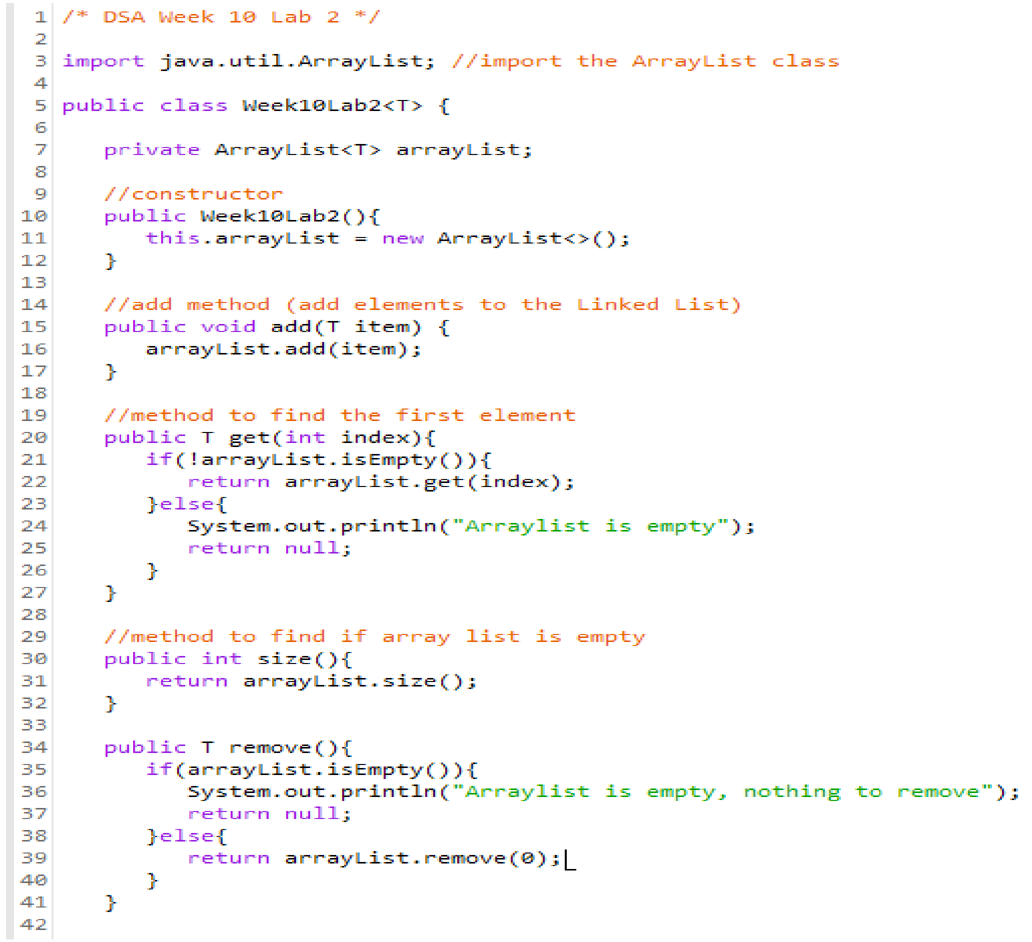
Code Fragment 7.3: An implementation of a simple ArrayList class with bounded capacity. (Continued from Code Fragment 7.2.)

A simple ArrayList implementation

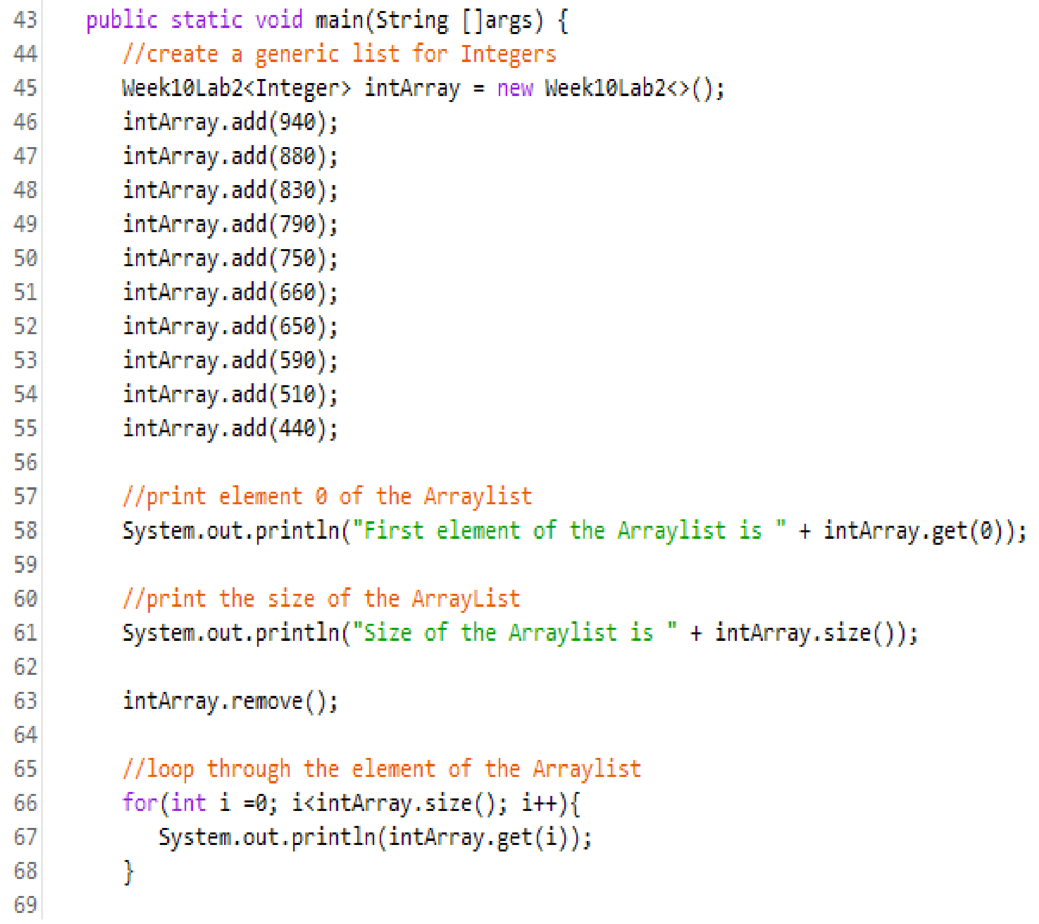


```
/* DSA Week 10 Lab 1 */

import java.util.ArrayList; //import the ArrayList class

public class Week10Lab1 {

   public static void main(String []args) {
      ArrayList<Integer> intArray = new ArrayList<Integer>();
      intArray.add(940);
      intArray.add(880);
      intArray.add(830);
      intArray.add(790);
      intArray.add(750);
      intArray.add(660);
      intArray.add(650);
      intArray.add(590);
      intArray.add(510);
      intArray.add(440);

      //print element 0 of the Arraylist
      System.out.println("First element of the Arraylist is " + intArray.get(0));

      //print the size of the ArrayList
      System.out.println("Size of the Arraylist is " + intArray.size());

      intArray.remove(0);

      //loop through the element of the Arraylist
      for(int i =0; i<intArray.size(); i++){
         System.out.println(intArray.get(i));
      }
   }
}
```

A generic ArrayList implementation (1/2)

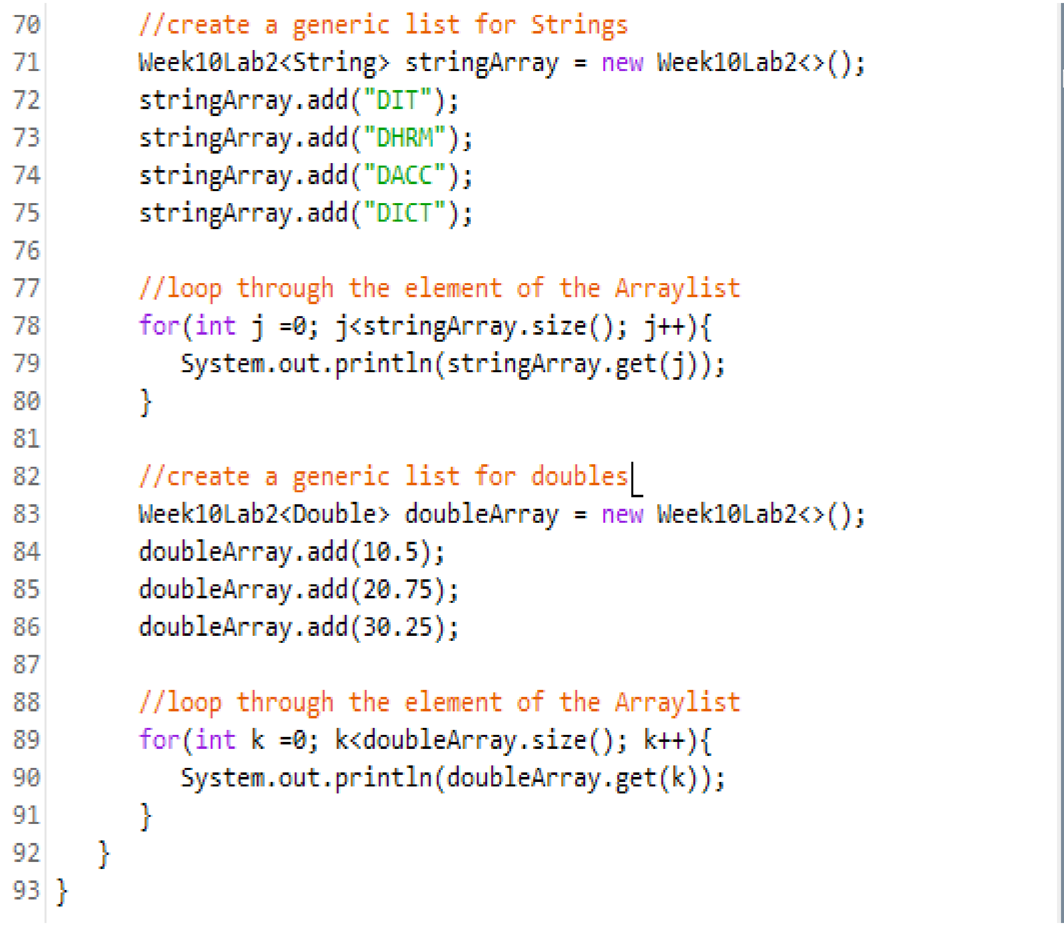


```java
/* DSA Week 10 Lab 2 */

import java.util.ArrayList; //import the ArrayList class

public class Week10Lab2<T> {

   private ArrayList<T> arrayList;

   //constructor
   public Week10Lab2(){
      this.arrayList = new ArrayList<>();
   }

   //add method (add elements to the Linked List)
   public void add(T item) {
      arrayList.add(item);
   }

   //method to find the first element
   public T get(int index){
      if(!arrayList.isEmpty()){
         return arrayList.get(index);
      }else{
         System.out.println("Arraylist is empty");
         return null;
      }
   }

   //method to find if array list is empty
   public int size(){
      return arrayList.size();
   }

   public T remove(){
      if(arrayList.isEmpty()){
         System.out.println("Arraylist is empty, nothing to remove");
         return null;
      }else{
         return arrayList.remove(0);
      }
   }

```

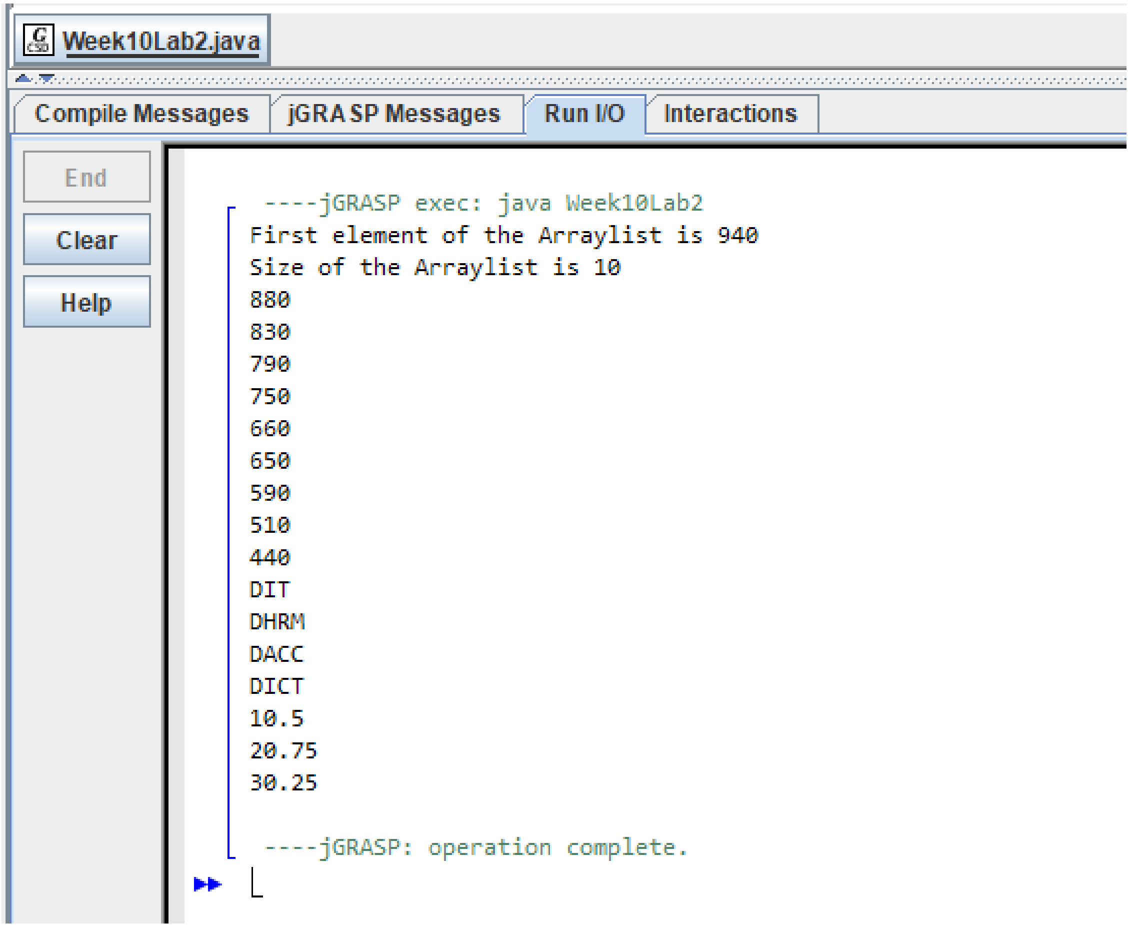


```java
   public static void main(String []args) {
      //create a generic list for Integers
      Week10Lab2<Integer> intArray = new Week10Lab2<>();
      intArray.add(940);
      intArray.add(880);
      intArray.add(830);
      intArray.add(790);
      intArray.add(750);
      intArray.add(660);
      intArray.add(650);
      intArray.add(590);
      intArray.add(510);
      intArray.add(440);

      //print element 0 of the Arraylist
      System.out.println("First element of the Arraylist is " + intArray.get(0));

      //print the size of the ArrayList
      System.out.println("Size of the Arraylist is " + intArray.size());

      intArray.remove();

      //loop through the element of the Arraylist
      for(int i =0; i<intArray.size(); i++){
         System.out.println(intArray.get(i));
      }

```

A generic ArrayList implementation (2/2)



```java
        //create a generic list for Strings
        Week10Lab2<String> stringArray = new Week10Lab2<>();
        stringArray.add("DIT");
        stringArray.add("DHRM");
        stringArray.add("DACC");
        stringArray.add("DICT");

        //loop through the element of the Arraylist
        for(int j =0; j<stringArray.size(); j++){
           System.out.println(stringArray.get(j));
        }

        //create a generic list for doubles
        Week10Lab2<Double> doubleArray = new Week10Lab2<>();
        doubleArray.add(10.5);
        doubleArray.add(20.75);
        doubleArray.add(30.25);

        //loop through the element of the Arraylist
        for(int k =0; k<doubleArray.size(); k++){
           System.out.println(doubleArray.get(k));
        }
    }
}
```



```
 ----jGRASP exec: java Week10Lab2
First element of the Arraylist is 940
Size of the Arraylist is 10
880
830
790
750
660
650
590
510
440
DIT
DHRM
DACC
DICT
10.5
20.75
30.25

 ----jGRASP: operation complete.
```

Iterators

- An iterator is a software design pattern that abstracts the process of scanning through a sequence of elements, one element at a time.
- An iterator is an object that can be used to loop through collections like Arraylist and Linkedlist.
- Iterating is the technical term for looping

java.util.Iterator class

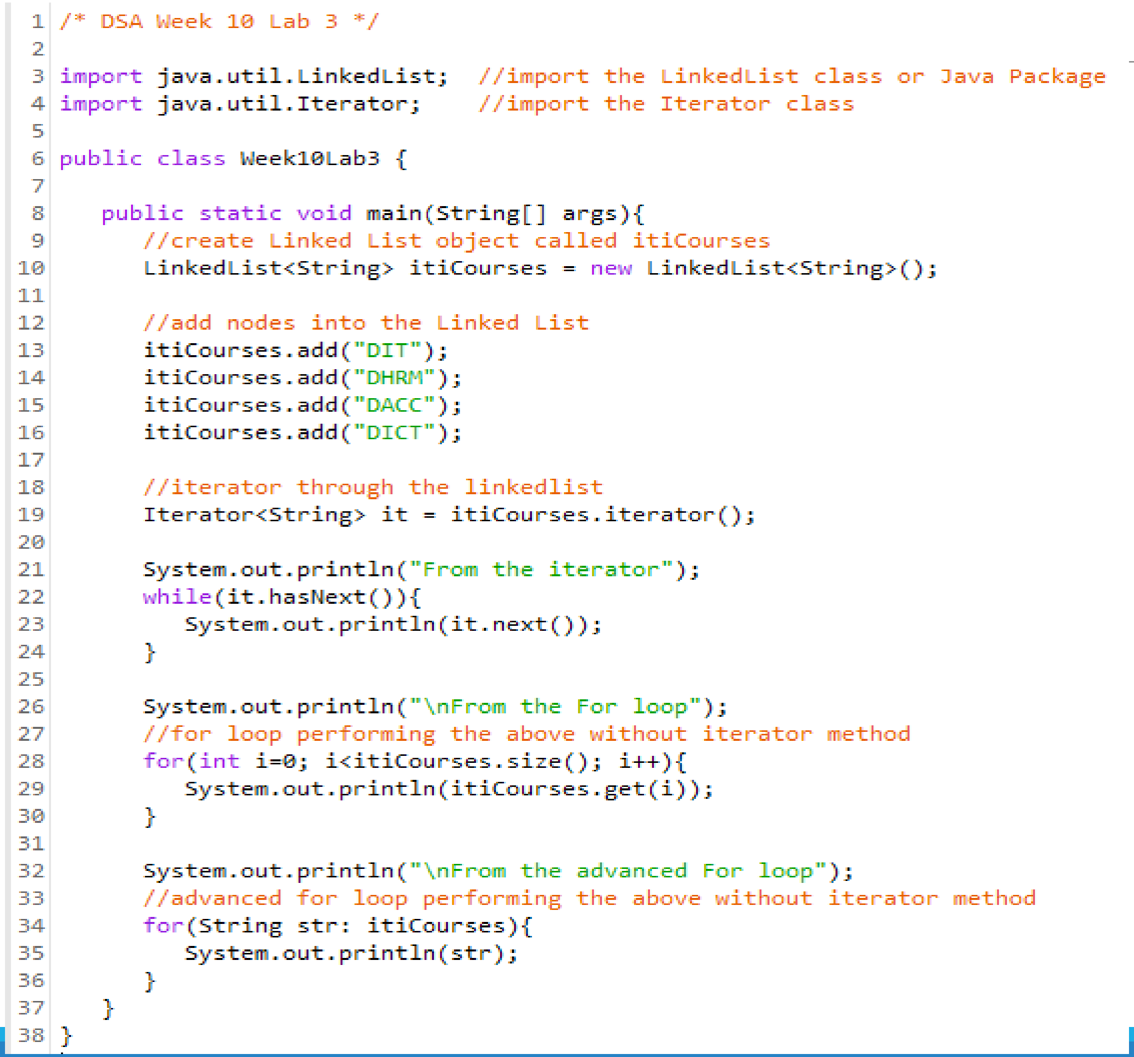
- Java provides a class named java.util.Iterator that allows the iterator features with data structure in a Java program
- NOTE: before creating your program’s class, you must import java.util.iterator; class in order to iterator or loop through elements in a data structure
-
- When the import statement is declared, you can now use the three methods:
  - **hasNext( ):** Returns true if there is at least one additional element in the sequence, and false otherwise.
  - **next( ):** Returns the next element in the sequence.
  - **Remove():** Removes from the collection the element returned by the most recent call to next( ). Throws an IllegalStateException if next has not yet been called, or if remove was already called since the most recent call to next.

java.util.Iterator class

- The combination of these two methods allows a general loop construct for processing elements of the iterator. For example, if we let variable, iter, denote an instance of the Iterator<String> type, then we can write the following:

```
while (iter.hasNext( )) {
        String value = iter.next( );
        System.out.println(value);
}
```

Iterator Code breakdown



```java
/* DSA Week 10 Lab 3 */

import java.util.LinkedList;  //import the LinkedList class or Java Package
import java.util.Iterator;    //import the Iterator class

public class Week10Lab3 {

   public static void main(String[] args){
      //create Linked List object called itiCourses
      LinkedList<String> itiCourses = new LinkedList<String>();

      //add nodes into the Linked List
      itiCourses.add("DIT");
      itiCourses.add("DHRM");
      itiCourses.add("DACC");
      itiCourses.add("DICT");

      //iterator through the linkedlist
      Iterator<String> it = itiCourses.iterator();

      System.out.println("From the iterator");
      while(it.hasNext()){
         System.out.println(it.next());
      }

      System.out.println("\nFrom the For loop");
      //for loop performing the above without iterator method
      for(int i=0; i<itiCourses.size(); i++){
         System.out.println(itiCourses.get(i));
      }

      System.out.println("\nFrom the advanced For loop");
      //advanced for loop performing the above without iterator method
      for(String str: itiCourses){
         System.out.println(str);
      }
   }
}
```

Step 1: import statements to use LinkedList and Iterator

Step 2: create class with public modifier then give a class name ‘Week10Lab3’

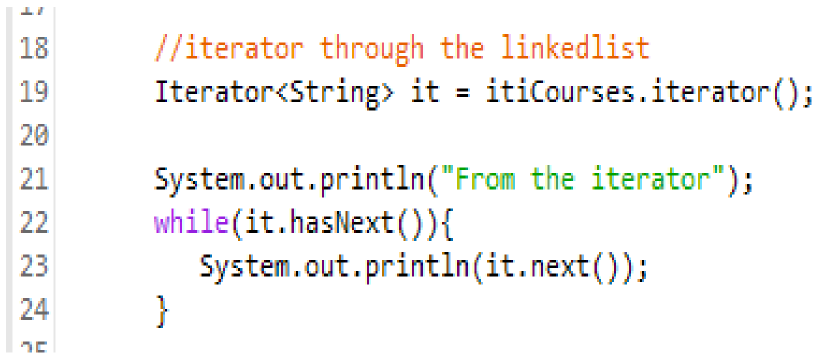
Step 3: create the main method of the program

Step 4: create the data structure LinkedList that holds datatype Strings. The Linkedlist will store to value called itCourses that stores the String values.

Step 5: use .add() method to insert elements or String values into the LinkedList.

Step 6: you can use either iterator statement, tradition loop like for loop or use an advanced loop method called a for-each loop to display each element in the linkedlist.

Explanation of Iterator statement

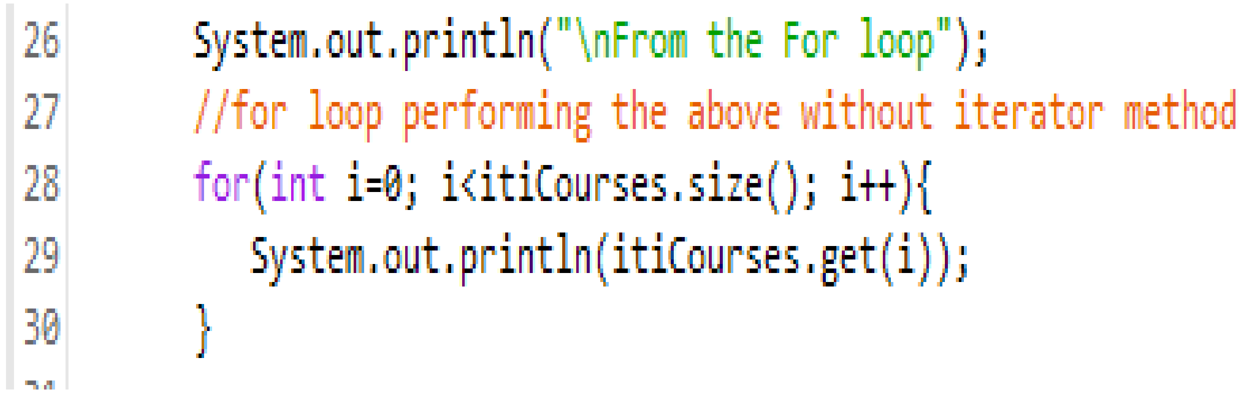


```
    //iterator through the linkedlist
    Iterator<String> it = itiCourses.iterator();

    System.out.println("From the iterator");
    while(it.hasNext()){
       System.out.println(it.next());
    }
```

Using the iterator method:

1. Create an object called iterator that loops Strings assigned to value ‘it’
2. Display a message showing ‘From the interator’.
3. Using a while loop iterator using the it.hasNext() to check through the linkedlist for elements
4. Display each element using the next() method from the iterator

Explanation of the traditional (for loop method)

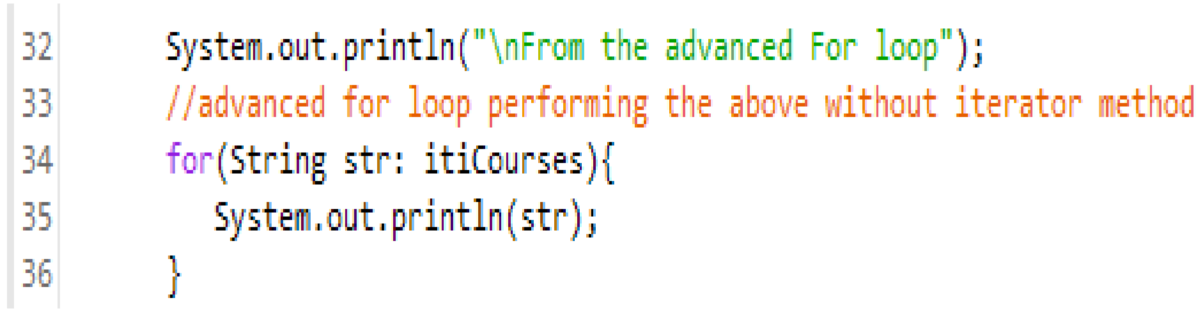


```
      System.out.println("\nFrom the For loop");
      //for loop performing the above without iterator method
      for(int i=0; i<itiCourses.size(); i++){
         System.out.println(itiCourses.get(i));
      }
```

Using the traditional for loop method:

1. Display a message showing ‘From the For loop’.
2. Using the for-loop syntax – for (initalisation, condition, increment)
3. Display each element of the Linkedlist using the for loop

Using the advanced looping method (For-Each loop)



```
System.out.println("\nFrom the advanced For loop");
//advanced for loop performing the above without iterator method
for(String str: itiCourses){
   System.out.println(str);
}
```

The for-each loop is simplest loop to use with data structures such as Arrays and LinkedList.

Using the advanced looping method:

1. Display a message showing *'From the advanced For loop'*.
2. Using the for-loop syntax – for (DataType variableName : datastructureName).

   for (String str : itiCourses)
3. Display each element of the Linkedlist using the for-each loop

Topic next week

Week 15 Test 2 (10%)

Week 16 General trees & binary trees

DSA Week 14 activities

Refer to print out materials.

- This week, you are required to complete the questions and two labs.
  - Refer to the print out, answer all week 14 questions.
  - Refer to the print out, complete the two labs activities using the lab computers.

Note: You can complete the activities in any order, however, make afford to complete and understand everything which prepares you for well for the Final Exam.