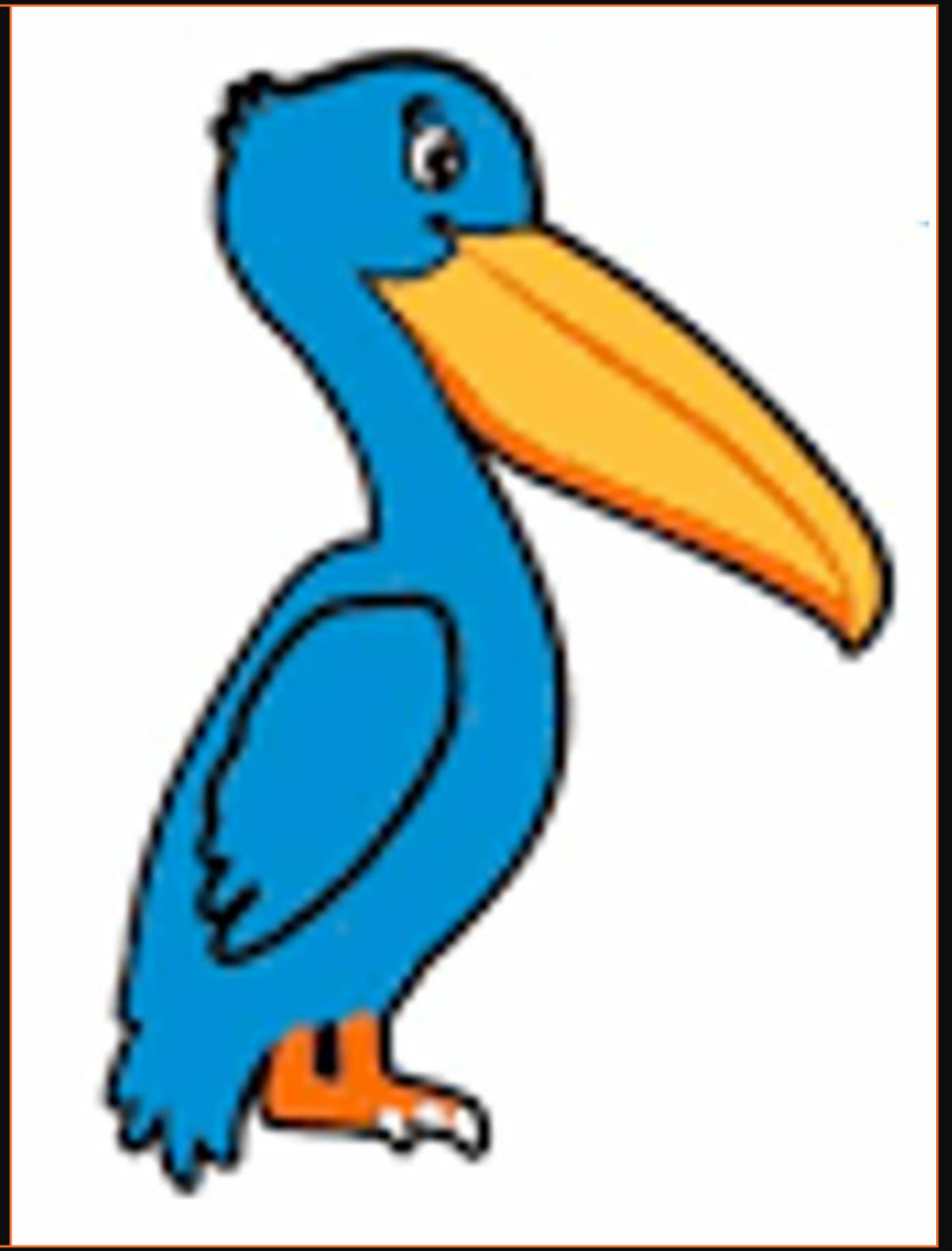
AP Computer Science A

Java Programming Essentials [Ver.4.0]

Unit 4: Data Collections

CHAPTER 15: ARRAYLIST

DR. ERIC CHOU
IEEE SENIOR MEMBER



AP Computer Science Curriclum

- ArrayList Methods (T4.8)
- ArrayList Traversal (T4.9)

Objectives:

- What is List? What is ArrayList? And, What is ArrayList of Objects?
- Generic Programming – Type Variable
- Declaration – Instantiation – Initialization
- ArrayList Methods
- ArrayList Processing I: list equality, list searching, sub-list functions, list traversal, and list processing

Overview

Lecture 1

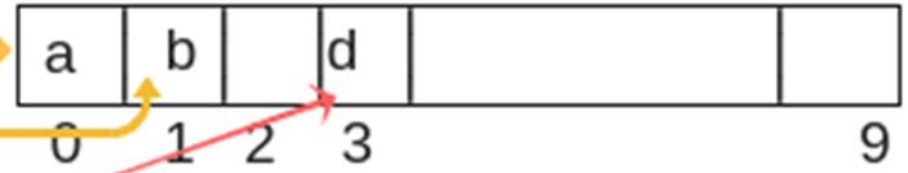
What is List?

- A list is an ordered collection where elements are maintained by index positions because it uses an index based structure. In the list **interface**, the order is retained in which we add elements and we will get the same sequence while retrieving elements.
- It is used to store a collection of elements where duplicate elements are allowed.
- List interface has three concrete subclasses. They are ArrayList, LinkedList, Vector, and Stack. These three subclasses implement the list interface.

Properties of List Interface in Java

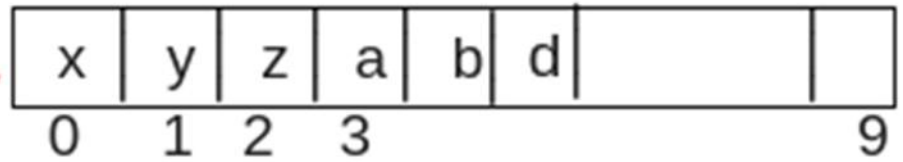
- 1. The list **allows** storing **duplicate** elements in Java.
- 2. It **maintains** insertion **order**.
- 3. It allows for **storing** many **null keys**.
- 4. The list uses a **Resizable** array for their implementation. Resizable means we can increase or decrease the size of the array.
- 5. Except for LinkedList, ArrayList, and Vector is **indexed based** structure.
- 6. It provides a special **Iterator** called a ListIterator that allow accessing the elements in the forward direction using hasNext() and next() methods.

1. list.add("a");
   list.add("b");



2. list.add(3,"d");

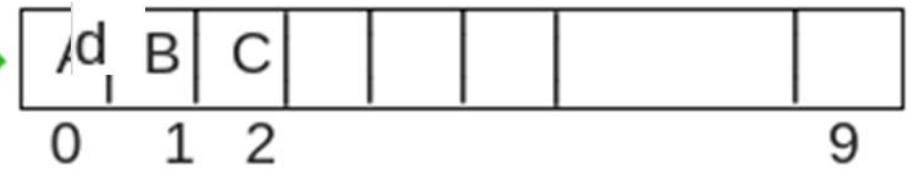
3. list.addAll(list1);



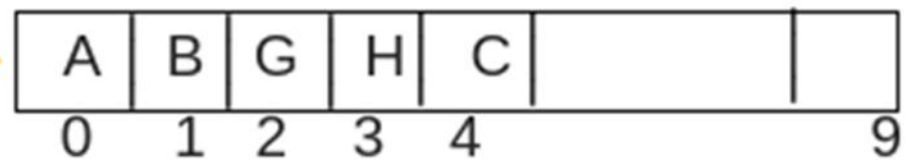
list

4. list1.add("A");
   list1.add("B");
   list1.add("C");
   list2.add("G");
   list2.add("H");
   list1.addAll(2 , list2);

list1



list2



ArrayList

Lecture 2

ArrayList in Java

- java.util.ArrayList is **Resizable-array** that can grow or shrink as needed. It is created with an initial size. When this size is exceeded, the collection is automatically grown. When objects are removed, the array can be shrunk.
- Elements in ArrayList are placed according to the **zero-based index**. That is the first element will be placed at 0 index and last element at index (n-1) where n is the size of ArrayList.
- It uses a **Dynamic array** internally for storing the group of elements or data.

Why ArrayList?

Memory allocation. Programming convenience.

- Arrays have the disadvantage that, once they have been created, their lengths can never be changed. We are often faced with situations, however, where the natural thing to do is to delete or insert array elements. One way to achieve this is to create a new array into which we then copy all the elements we want to keep together with any additional elements we want to insert.
- Therefore, we need a different data structure that can behave like array but allows us to grow or shrink the ***"array"*** without copying all the elements. For this need, ArrayList serves this purpose very well.

Why ArrayList?

- To overcome this difficulty, the Java language provides a class called ArrayList. An instance of ArrayList can be used to store data just like a regular Java array. Individual elements of arrays and ArrayList objects are accessed in similar ways, using an index.
- Unlike arrays, however, an ArrayList object allows us to change its length by deleting elements or inserting elements (at any point, not just at the end).

Declaration of ArrayList

ArrayList like a train. The datatype it carrys is like the cargo.

- In declaring a variable of type **ArrayList** we use a statement like this:

```
ArrayList<String> aList;
```

- The word between the *angle brackets*, **<…>**, indicates the data type of the elements that the **ArrayList** will store. In this case, **aList** is declared to be an **ArrayList** each of whose elements will be a **String**.

Declaration of ArrayList

ArrayList like a train. The datatype it carrys is like the cargo.

- The following statement creates an **ArrayList** of **String**s and then assigns it to **aList**:

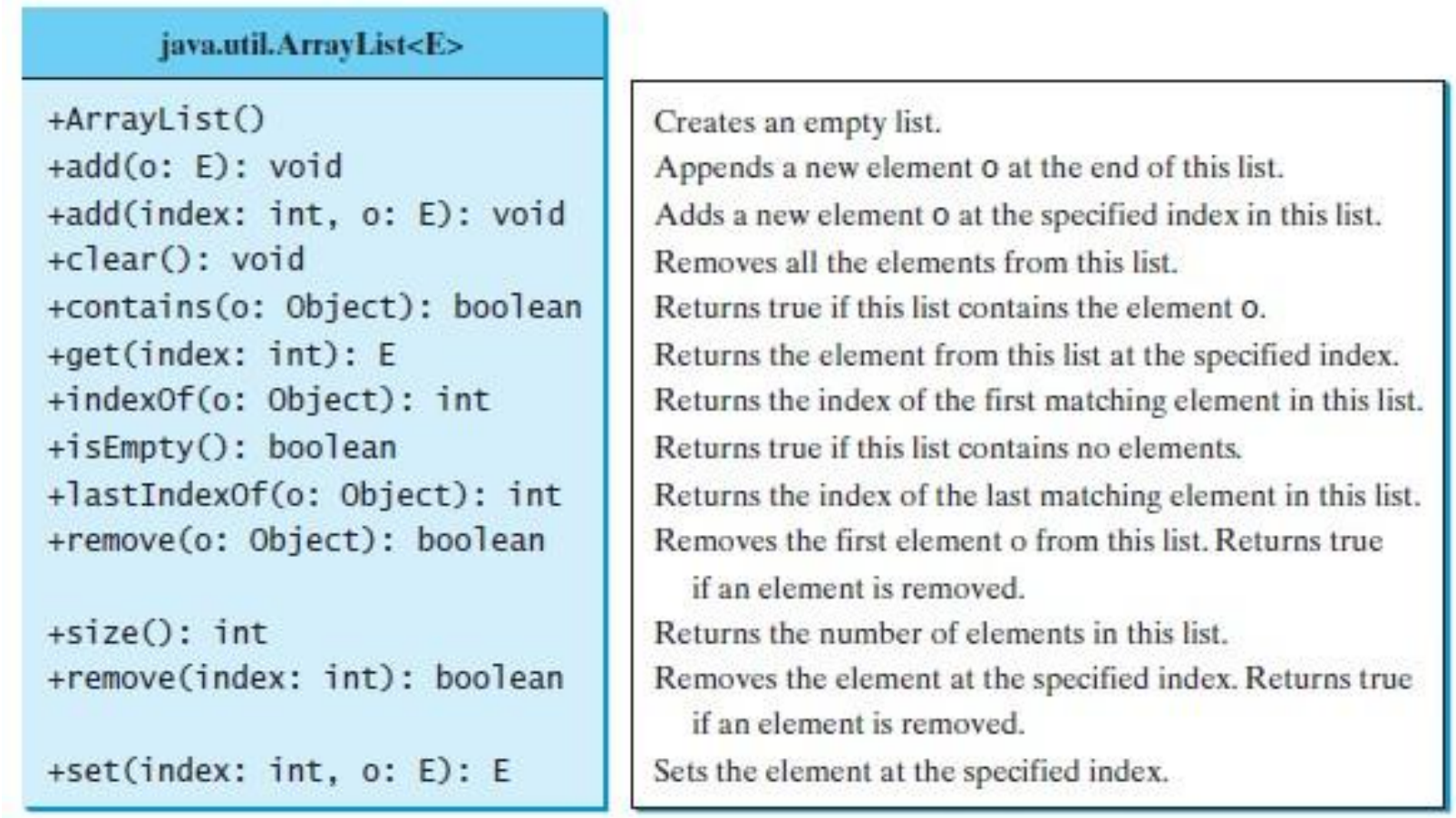
```
aList = new ArrayList<String>();
```

- We can also declare and assign to the variable in a single statement:

```
ArrayList<String> aList =
                    new ArrayList<String>();
```

The ArrayList Class

- You can create an array to store objects. But the array's size is fixed once the array is created. Java provides the ArrayList class that can be used to store an unlimited number of objects.



Creation of ArrayList

- Creation of ArrayList:
  ArrayList<Integer> list = new ArrayList<Integer>();
- Adding Elements:
  list.add(3); list.add(new Integer(4)); list.add(null); list.add(5);
- Removing an Element:
  list.remove(2);
- Printing a list:
  System.out.println(list.get(2));
  System.out.println(list);

Demonstration Program

CreationofList.java

```java
import java.util.ArrayList;
public class CreationOfList
{
    public static void main(String[] args){
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(3);list.add(new Integer(4)); list.add(null); list.add(5);
        list.remove(2);
        System.out.println(list.get(2));
        System.out.println(list);
    }
}
```

Generic Programming Type - Variable

Lecture 3

Generic Data Type <E>

Can be String, Integer, Double, … (but not primitive data types)

- If `E` is the name of a data type, the `ArrayList<E>` class is an example of a so-called *generic class* with *type parameter* whatever the replacement for `E` is. (Before we replace `E` by the name of an actual data type, it is called a *formal type parameter*.)

Raw ArrayList

- It is also possible to declare an **ArrayList** without using a type parameter. Such a usage treats **ArrayList** as a so-called *raw class*. Raw **ArrayLists** require careful handling. However, they are not included in the AP subset and we have very little to do with them in this course.
- Raw ArrayLists are ArrayList of Object class type.

NO ArrayList<int>

- Generics in Java is not applicable to primitive types as in **int**. You should use the **wrapper** types as in **Integer**:
- `int a = 3;`
- `ArrayList<Integer> aList = new ArrayList<Integer>();`
- `// ArrayList<Integer>(); is a constructor call like`
- `//   Scanner(System.in);`

- `aList.add(a); // put data of int type here is OK,`
  `// because of auto-boxing`

Basic ArrayList Operations

Lecture 4

Differences and Similarities between Arrays and ArrayList

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

How to create an ArrayList?

- We can create an ArrayList by writing a simple statement like this:
- This statement creates an ArrayList with the name alist with type “String”. The type determines which type of elements the list will have. Since this list is of “String” type, the elements that are going to be added to this list will be of type “String”.
- `ArrayList<String> alist=new ArrayList<String>();`
- Similarly we can create ArrayList that accepts int elements.
- `ArrayList<Integer> list=new ArrayList<Integer>();`

How to add elements to an ArrayList?

- We add elements to an ArrayList by using add() method, this method has couple of variations, which we can use based on the requirement. For example: If we want to add the element at the end of the List then simply do it like this:
- `alist.add("Steve"); // This will add "Steve" at the end of List`
- To add the element at the specified location in ArrayList, we can specify the index in the add method like this:
- `alist.add(3, "Steve"); // This will add "Steve"`
- `// at the fourth position`

Demonstration Program

JavaExample1.java

```java
import java.util.*;
class JavaExample1{
    public static void main(String args[]){
        ArrayList<String> alist=new ArrayList<String>();
        alist.add("Steve");
        alist.add("Tim");
        alist.add("Lucy");
        alist.add("Pat");
        alist.add("Angela");
        alist.add("Tom");
        //displaying elements
        System.out.println(alist);
        //Adding "Steve" at the fourth position
        alist.add(3, "Steve");
        //displaying elements
        System.out.println(alist);
    }
}
```

How to remove elements from ArrayList?

- We use remove() method to remove elements from an ArrayList, Same as add() method, this method also has few variations.
  1. Removing an object.
  2. Removing the object at a specific index location.

Demonstration Program

JavaExample2.java

```java
import java.util.*;
class JavaExample2{
    public static void main(String args[]){
        ArrayList<String> alist=new ArrayList<String>();
        alist.add("Steve");
        alist.add("Tim");
        alist.add("Lucy");
        alist.add("Pat");
        alist.add("Angela");
        alist.add("Tom");
        //displaying elements
        System.out.println(alist);
        //Removing "Steve" and "Angela"
        alist.remove("Steve");
        alist.remove("Angela");
        //displaying elements
        System.out.println(alist);
        //Removing 3rd element
        alist.remove(2);
        //displaying elements
        System.out.println(alist);
    }
}
```

Iterating ArrayList

- In the above examples, we have displayed the ArrayList elements just by referring the ArrayList instance, which is definitely not the right way to displays the elements. The correct way of displaying the elements is by using an advanced for loop like this.
- ArrayList is a subclass of collections. Therefore, it is also an iterable class. It can be used in an enhanced for-loop

Demonstration Program

JavaExample3.java

```java
import java.util.*;
class JavaExample3{
   public static void main(String args[]){
      ArrayList<String> alist=new ArrayList<String>();
      alist.add("Gregor Clegane");
      alist.add("Khal Drogo");
      alist.add("Cersei Lannister");
      alist.add("Sandor Clegane");
      alist.add("Tyrion Lannister");

      //iterating ArrayList
      for(String str:alist)
         System.out.println(str);
      }
}
```

ArrayList Example in Java

- This example demonstrates how to **create**, **initialize**, **add** and **remove** elements from ArrayList. In this example we have an ArrayList of type “String”. We have added 5 String element in the ArrayList using the method add(String E), this method adds the element at the end of the ArrayList.
- We are then adding two more elements in the ArrayList using method add(int index, String E), this method adds the specified element at the specified index, index 0 indicates first position and 1 indicates second position.

ArrayList Example in Java

- We are then removing the elements “Chaitanya” and “Harry” from the ArrayList and then we are removing the second element of the ArrayList using method remove(int index). Since we have specified the index as 1 (remove(1)), it would remove the second element.

Demonstration Program

JavaExample4.java

Demonstration Program

DistinctNumbers.java

Distinct Numbers

(1) Add numbers into an array list only when it does not exist in the arraylist.

(2) Use ArrayList for creating a non-recurring list of element (finding the distinct members in a set). This is a very important function in data analysis.

(3) Use 0 as sentinel to terminate the input process.

(4) Understand the basic nature of arraylists.

```java
import java.util.ArrayList;
import java.util.Scanner;

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

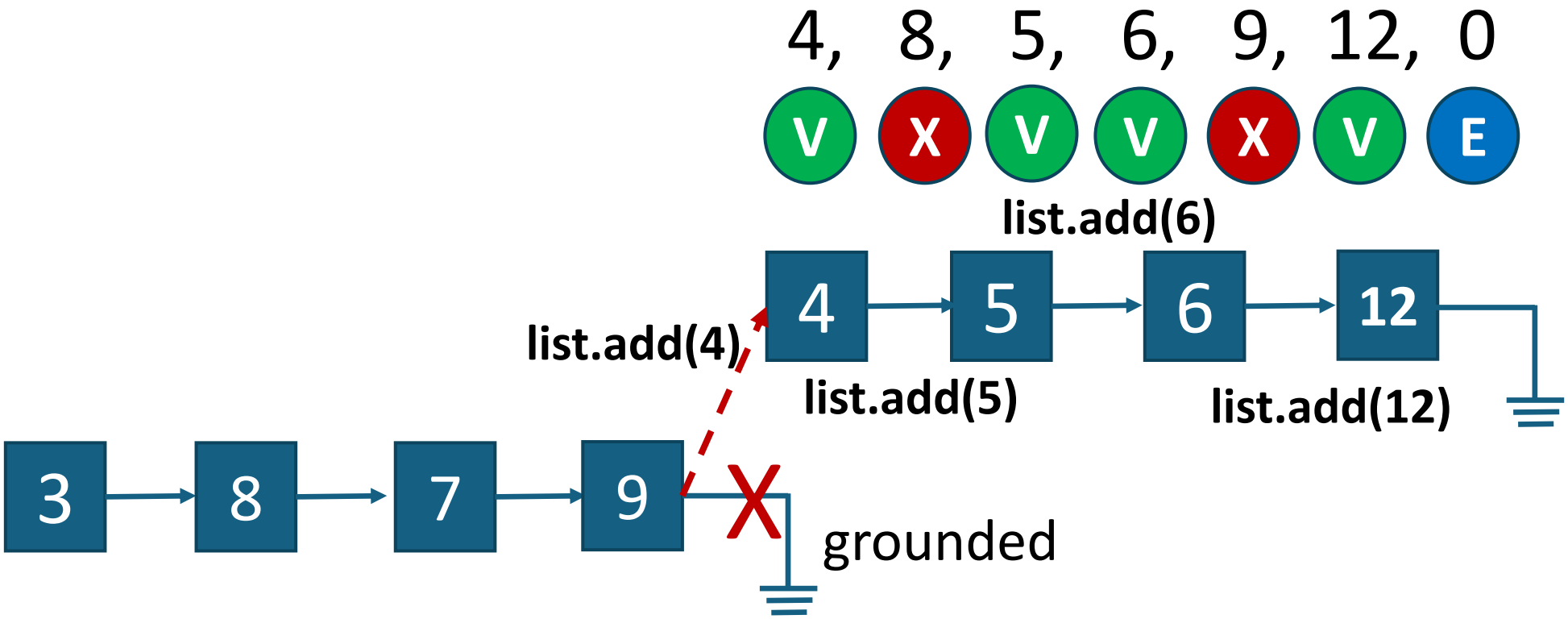
    Scanner input = new Scanner(System.in);
    System.out.print("Enter integers (input ends with 0): ");
    int value;

    do {
      value = input.nextInt(); // Read a value from the input

      if (!list.contains(value) && value != 0)
        list.add(value); // Add the value if it is not in the list
    } while (value != 0);

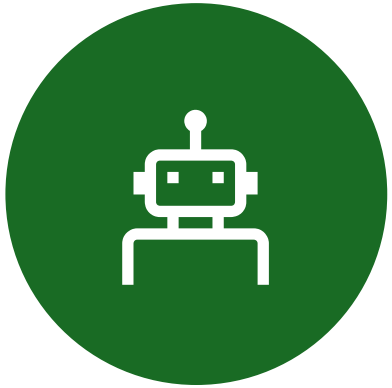
    // Display the distinct numbers
    System.out.print("Print by Loop: ");
    for (int i = 0; i < list.size(); i++)
      System.out.print(list.get(i) + " ");
    System.out.println();
    System.out.println("Print by List: "+list);
  }
}
```

Adding a Distinct Number into an ArrayList



Features of ArrayList used in DistinctNumbers.java

(1) CONTAINS()



(2) RANDOM INSERTION.

(3) FLEXIBLE LENGTH.

Wrapper Classes as Objects

Lecture 5

The purpose of Re-visiting Wrapper Class:

ArrayList only Supports Objects (Wrapper Class), no Primitive Data Types

- ArrayList is a data structure **similar to** arrays and strings.
- ArrayList can only carry **objects** (wrapper classes and other object classes). It cannot carry primitive data types.
- ArrayList is just like Arrays. It is an object class. It has iterator interface and can be accessed by indexes. Index can be int type while elements need to be of Integer type.
- Therefore, we re-visit Integer, Double wrapper classes here to get the background information for processing ArrayLists

Wrapper Classes (Objects)

- Boolean
- Character
- Short
- Byte
- Integer
- Long
- Float
- Double

NOTE:

(1) The wrapper classes do not have no-arg constructors. (need to set some value)

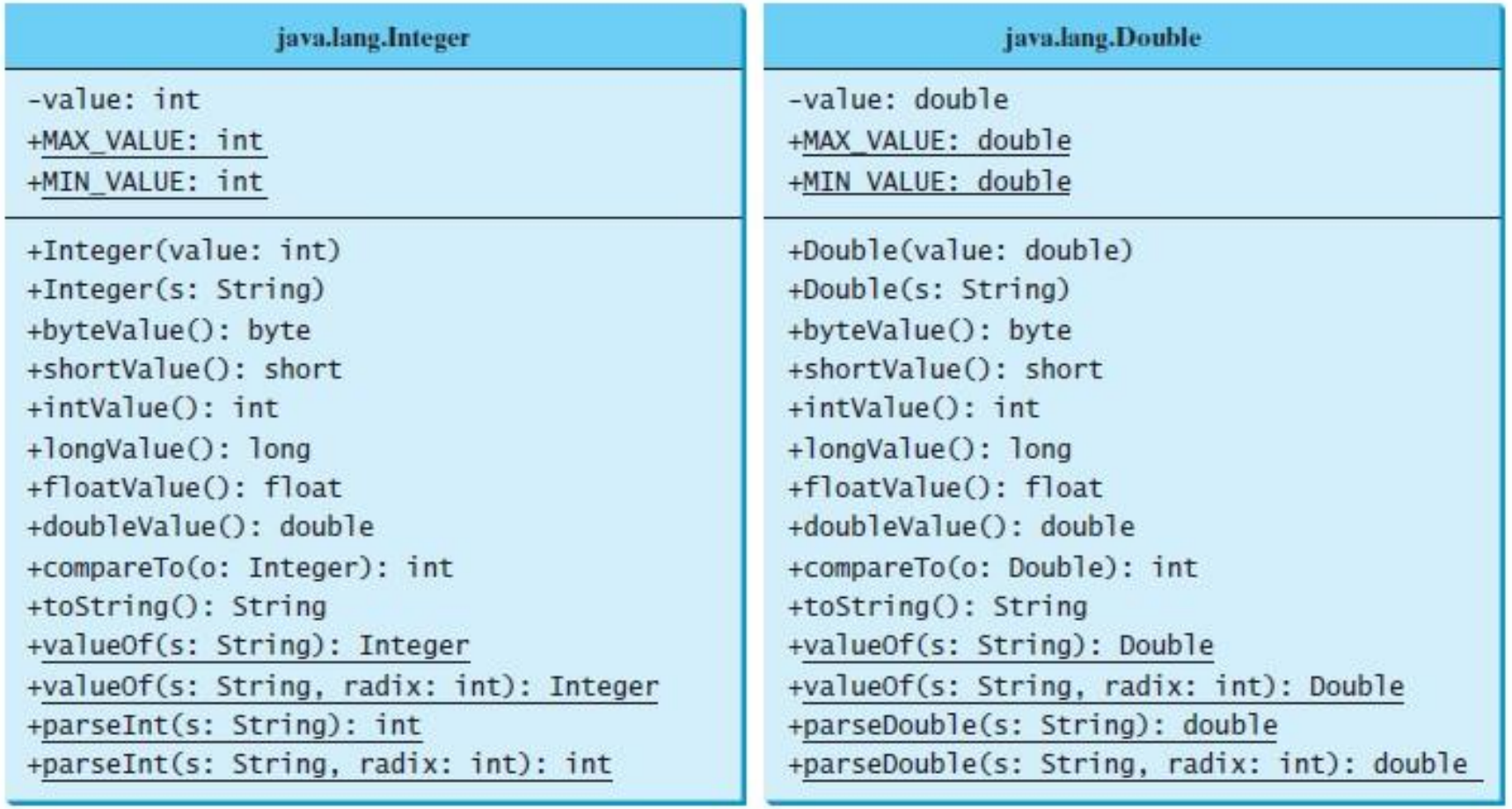
(2) The instances of all wrapper classes are immutable, i.e., their internal values cannot be changed once the objects are created.

(3) Wrapper class provides **methods**. Forms **objects**.

The toString, equals, and hashCode Methods

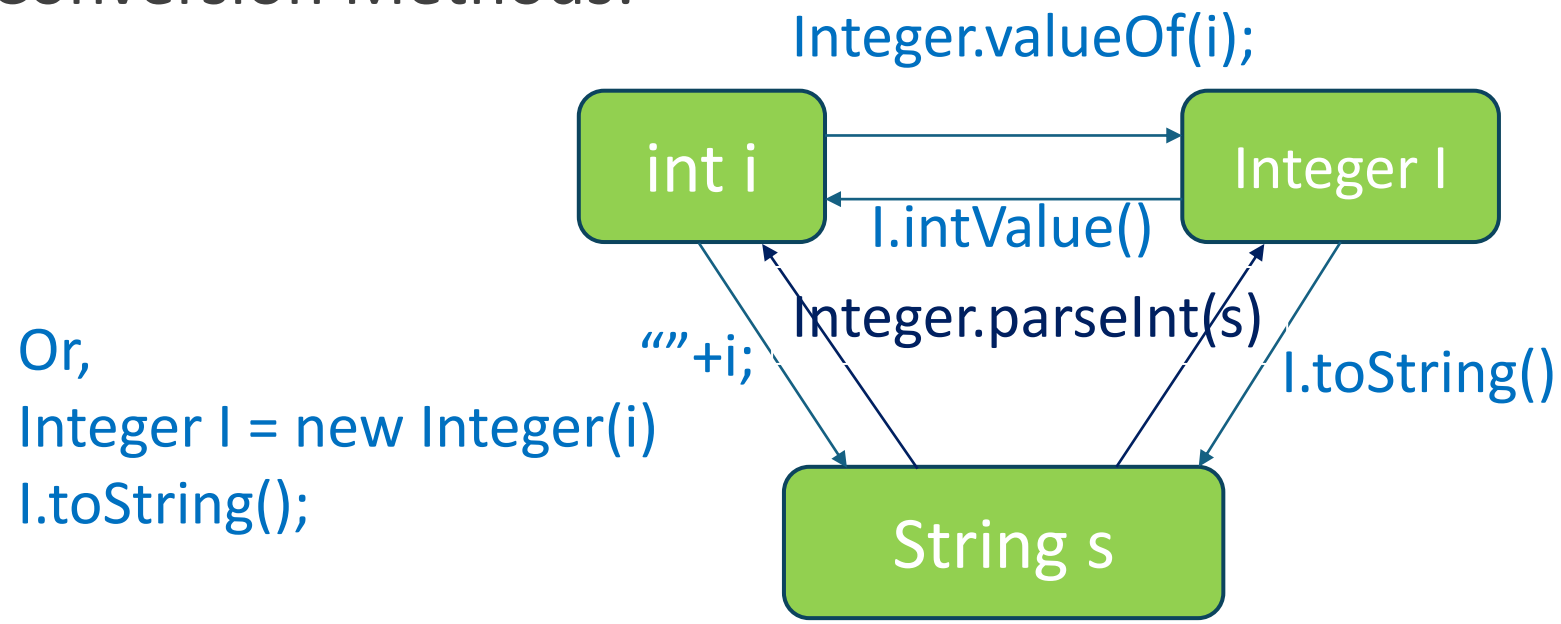
- Each wrapper class overrides the toString, equals, and hashCode methods defined in the Object class.
- Since all the numeric wrapper classes and the Character class implement the Comparable interface, the **compareTo** method is implemented in these classes.

The Integer and Double Classes



The `Integer` Class and the `Double` Class

- Constructors Integer(int i); Double(double d);
- Class Constants `MAX_VALUE`, `MIN_VALUE`
- Conversion Methods:



Numeric Wrapper Class Constructors

You can construct a wrapper object either from a primitive data type value or from a string representing the numeric value. The constructors for Integer and Double are:

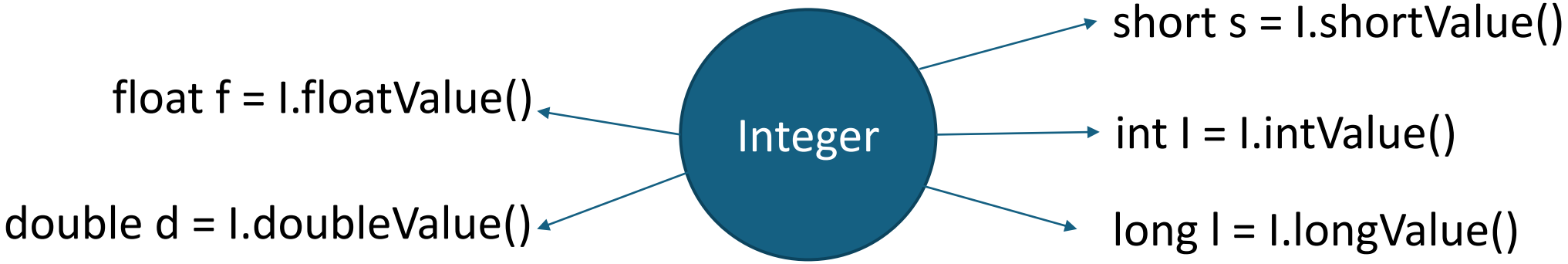
- public Integer(int value)
- public Integer(String s)
- public Double(double value)
- public Double(String s)

Numeric Wrapper Class Constants

- Each numerical wrapper class has the constants MAX_VALUE and MIN_VALUE. MAX_VALUE represents the maximum value of the corresponding primitive data type.
- For Byte, Short, Integer, and Long, MIN_VALUE represents the minimum byte, short, int, and long values. For Float and Double, MIN_VALUE represents the minimum *positive* float and double values.
- The following statements display the maximum integer (2,147,483,647), the minimum positive float (1.4E-45), and the maximum double floating-point number (1.79769313486231570e+308d).

Conversion Methods (Out-going methods)

- Each numeric wrapper class implements the abstract methods doubleValue, floatValue, intValue, longValue, and shortValue, which are defined in the Number class.
- These methods "convert" objects into primitive type values.



The Static valueOf Methods (In-coming Method)

- The numeric wrapper classes have a useful class method, valueOf(String s). This method creates a new object initialized to the value represented by the specified string. For example:

```
Double doubleObject = Double.valueOf("12.4");

Integer integerObject = Integer.valueOf("12");
```

The Methods for Parsing Strings into Numbers
(Incoming Method)

- You have used the parseInt method in the Integer class to parse a numeric string into an int value and the parseDouble method in the Double class to parse a numeric string into a double value.
- Each numeric wrapper class has two overloaded parsing methods to parse a numeric string into an appropriate numeric value.

Automatic Conversion Between Primitive Types and Wrapper Class Types

JDK 1.5 allows primitive type and wrapper classes to be converted automatically. **But not** in ArrayList's method argument list. For example, the following statement in (a) can be simplified as in (b):

```
Integer[] intArray = {new Integer(2),
  new Integer(4), new Integer(3)};
```

Equivalent

```
Integer[] intArray = {2, 4, 3};
```

(a) New JDK 1.5 boxing (b)

```
Integer[] intArray = {1, 2, 3};
System.out.println(intArray[0] + intArray[1] + intArray[2]);
```

Unboxing

Normally, interchangeable but be careful for method calls.

In Summary, int and Integer are same data of Two Different Formats

1. Auto-**boxing** and auto-**unboxing** allow them convert themselves back and forth automatically.
2. **Wrapper Classes** are object classes. Primitive data types are non-object. Objects are allowed to be on the ArrayList data structure while primitive data types are not. So, it is just like you travel with trains, before jumping on board of a train, you need to purchase your ticket. And, the being object type is the ticket to the train.
3. For **numerical computation**, primitive-type array may be more efficient. For **data**, **text** and **language processing**, object-type data structure (List, Sets, Map, Queue) may offer better solution.

ArrayList Creation

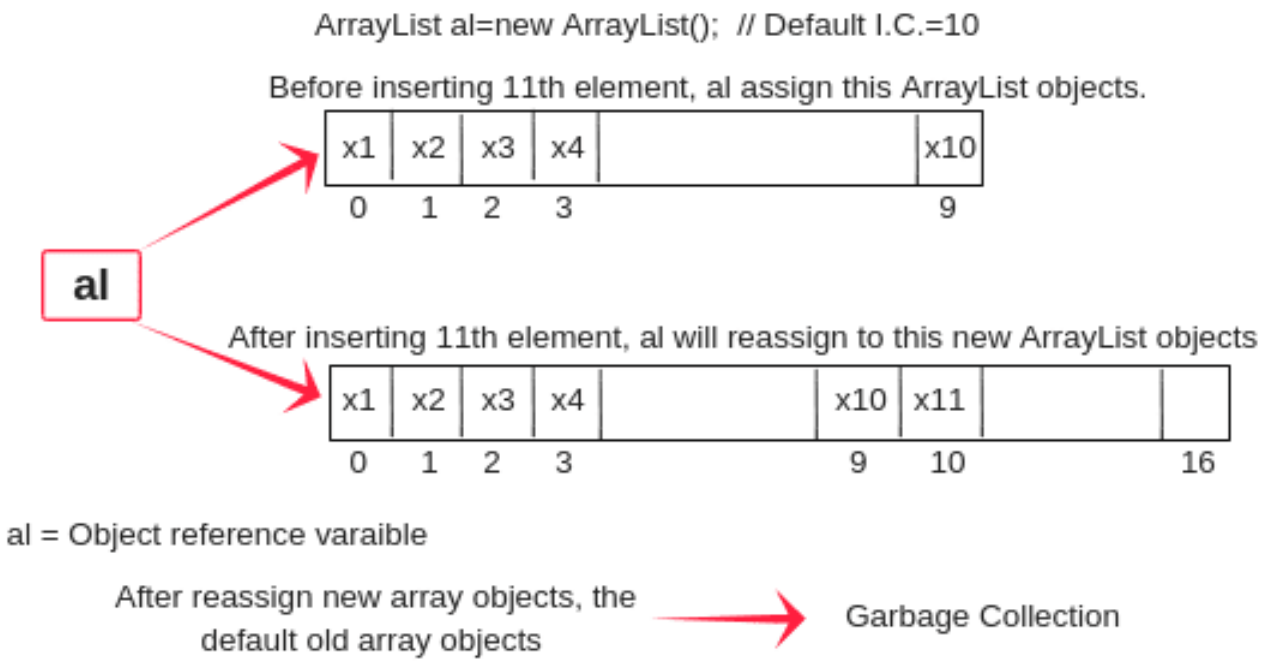
Lecture 6

ArrayList Creation

- Creating an array list object is very simple. First, we will declare an array list variable and call the array list constructor to instantiate an array list object and then assign it to the variable.
- There is different ways or methods of constructing an ArrayList object. They are as follows:

ArrayList al=new ArrayList();

- It creates an empty ArrayList with default initial capacity of 10. i.e. we can store only 10 elements as shown in below diagram.



ArrayList al=new ArrayList(int initialCapacity);

- It creates an empty ArrayList with initial capacity. If you know the initial capacity, you can directly use this object. So, the performance of the system by default will be improved.
- For example, suppose our requirement is 500 elements, we will go for this method like this.
- ```
  ArrayList list2=new ArrayList(500);
  ```
  ```

ArrayList al=new ArrayList(Collection c);

- It creates an array list that is initialized with the elements of collection c.
- **For example:**
- ```
  ArrayList list3=new ArrayList(list1);
  ```
- ```
  // list1 is the elements of collection.
  ```

Constructor for ArrayList

- `ArrayList()  // only creating the arraylist point`
- `ArrayList(Collection c)// creating arraylist with collection c`
- `ArrayList(int capacity)// creating an arraylist of capacity elements`
- Without **<E>** Generic Data Type Declaration, you will get a warning, but the program will still work.
- With **<E>**
- `ArrayList<E>()  // only creating the arraylist point`
- `ArrayList<E>(Collection c) // creating arraylist with a collection c`
- `ArrayList<E>(int capacity) // creating an arraylist of capacity`

ArrayList Initialization

Lecture 7

Initialization of ArrayList in Java

- There are three methods to initialize ArrayList. They are as follows:
- Method 1: Using Arrays.asList.
- **Syntax:**
- ```
  ArrayList<Type> list=new ArrayList<Type>(
  ```
- ```
   Arrays.asList(Object o1, Object o2, .. so on));
  ```
- **For example:**
- ```
  ArrayList<String> ar=
  ```
- ```
   new ArrayList<String>(Arrays.asList("A", "B", "C"));
  ```

Initialization of ArrayList in Java

Method 2: Using Normal way

- **Syntax:**
- ```
  ArrayList<Type> obj=new ArrayList<Type>();
  ```
- ```
  obj.add("Obj o1");
  ```
- ```
  obj.add("Obj o2");
  ```
- ```
  and so on.
  ```

Initialization of ArrayList in Java

Method 3: Using Anonymous Inner class

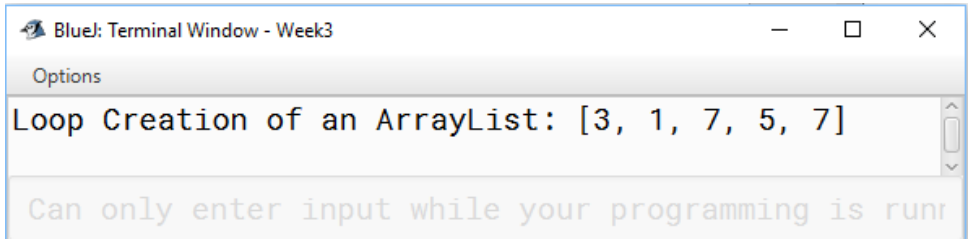
- Syntax:

```
ArrayList<Type> arl=new Arraylist<Type>() {{
          add(Object o1);
          add(Object o2);
          add(Object o3);
              . .. .. . . ..
              . . . . . . .
    }};
```

Creation of ArrayList:

Constructor, Loop Instantiation with add()

```
public static void createArrayListInLoop(){
    int count =5;
    ArrayList<Integer> aList= new ArrayList<Integer>();
    for (int i=0; i<5; i++){
       aList.add((int)(Math.random()*8));
    }
    System.out.println("Loop Creation of an ArrayList: "+aList);
}
```



Create an ArrayList from an Array

by new ArrayList<Integer>(Arrays.asList(arrayname));

```
public static void IntegerArrayToArrayList(){
  Integer[] ary = {1, 2, 3, 4}; // cannotuse int[] ary
                                // Integer[] ary = {new
Integer(1),
                                // …, new Integer(4)); OK
  ArrayList<Integer> aList=new
ArrayList<Integer>(Arrays.asList(ary));
  ArrayList<Integer> bList=new ArrayList<Integer>(
    Arrays.asList(new Integer[]{1, 2, 3, 4}
    );  // Example os using Anonymous Integer array
  String aa = aList.toString();
  System.out.println("aList printed by toString():"+aa);
  System.out.println("aList printed by default:"+aList);
  System.out.println("bList printed by default:"+bList);
}
```

```
Convert Integer array to ArrayList Example:.........
aList printed by toString(): [1, 2, 3, 4]
aList printed by default:    [1, 2, 3, 4]
bList printed by default:    [1, 2, 3]
```

Convert Primitive Data Type to Wrapper Type

Loop

- **Convert int[] to Integer[]:**

```
int[] primitiveArray = {1, 2, 3, 4, 5};
Integer[] objectArray = new Integer[PrimitiveArray.length];
for (int i=0; i<primitiveArray.length; i++){
  objectArray[i] = Integer.valueOf(primitiveArray[i]);
}
```

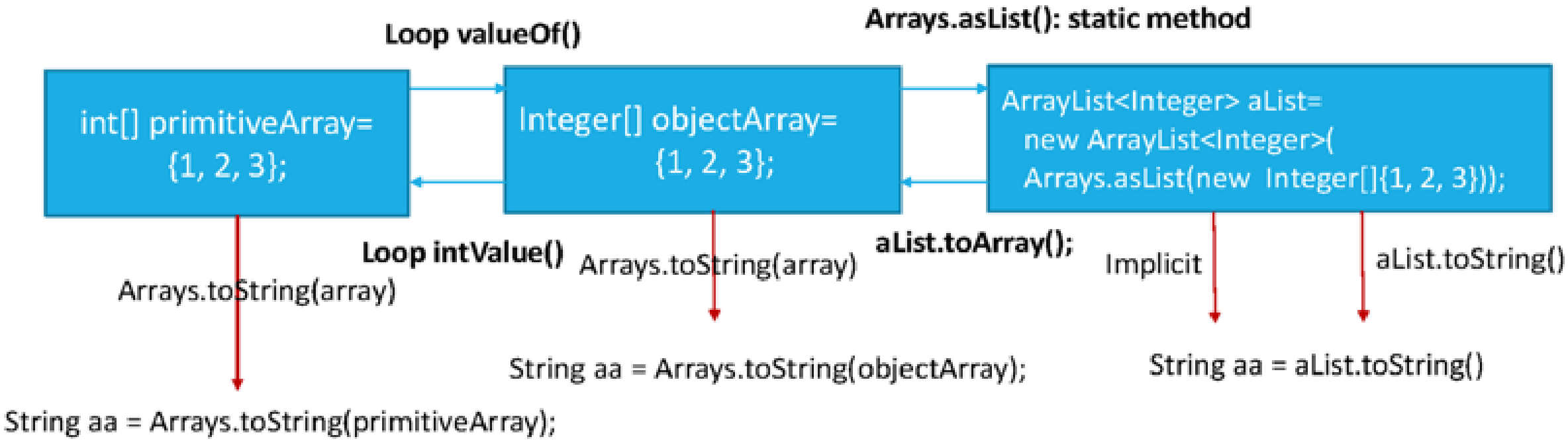
- **Convert Integer[] to int[]:**

```
Integer[] objectArray = {1, 2, 3, 4, 5};
int[] primitiveArray = new Integer[objectArray.length];
for (int i=0; i<objectArray.length; i++){
  primitiveArray[i] = objectArray[i].intValue();
}
```

```
Convert int array to Integer array Example:.........
int to Integer: [1, 2, 3, 4, 5] to [1, 2, 3, 4, 5]
Convert Integer array to int array Example:.........
Integer to int: [5, 4, 3, 2, 1] to [5, 4, 3, 2, 1]
```

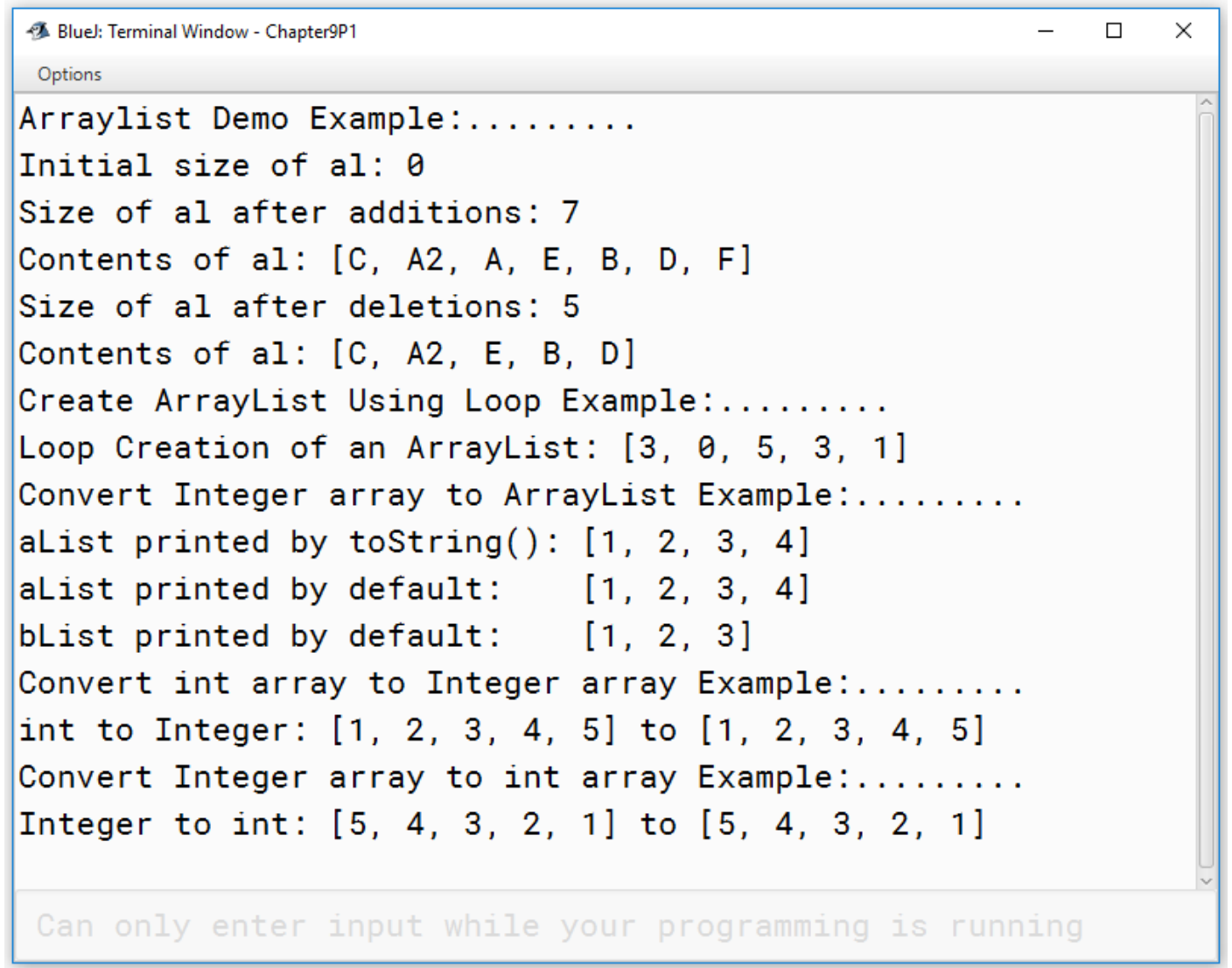
Conversion

Primitive Array – Object Array - ArrayList



Demonstration Program

ArrayListCreation.java


BlueJ: Terminal Window - Chapter9P1
Options
Arraylist Demo Example:.....
Initial size of al: 0
Size of al after additions: 7
Contents of al: [C, A2, A, E, B, D, F]
Size of al after deletions: 5
Contents of al: [C, A2, E, B, D]
Create ArrayList Using Loop Example:.....
Loop Creation of an ArrayList: [3, 0, 5, 3, 1]
Convert Integer array to ArrayList Example:.....
aList printed by toString(): [1, 2, 3, 4]
aList printed by default: [1, 2, 3, 4]
bList printed by default: [1, 2, 3]
Convert int array to Integer array Example:.....
int to Integer: [1, 2, 3, 4, 5] to [1, 2, 3, 4, 5]
Convert Integer array to int array Example:.....
Integer to int: [5, 4, 3, 2, 1] to [5, 4, 3, 2, 1]
Can only enter input while your programming is running

ArrayList Methods

Lecture 8

ArrayList Methods

1) **add(Object o)**: This method adds an object o to the arraylist.

```
obj.add("hello");
```

This statement would add a string hello in the arraylist at last position.

2) **add(int index, Object o)**: It adds the object o to the array list at the given index.

```
obj.add(2, "bye");
```

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) **remove(Object o)**: Removes the object o from the ArrayList.

```
obj.remove("Chaitanya");
```

This statement will remove the string “Chaitanya” from the ArrayList.

4) **remove(int index)**: Removes element from a given index.

```
obj.remove(3);
```

It would remove the element of index 3 (4th element of the list – List starts with o).

ArrayList Methods

5) **set(int index, Object o)**: Used for updating an element. It replaces the element present at the specified index with the object o.

```
obj.set(2, "Tom");
```

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6) **int indexOf(Object o)**: Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

```
int pos = obj.indexOf("Tom");
```

This would give the index (position) of the string Tom in the list.

7) **Object get(int index)**: It returns the object of list which is present at the specified index.

```
String str= obj.get(2);
```

Function get would return the string stored at 3rd position (index 2) and would be

ArrayList Methods

8) **int size()**: It gives the size of the ArrayList – Number of elements of the list.

```
int numberofitems = obj.size();
```

9) **boolean contains(Object o)**: It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

```
obj.contains("Steve");
```

It would return true if the string “Steve” is present in the list else we would get false.

10) **clear():** It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

```
obj.clear();
```

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values to the left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

ArrayList Processing I

Lecture 9

Topics in ArrayList Processing I

ArrayListProcessingI.java

- **Main Theme:** Element Management
  (1) List comparison: equals
  (2) contains(Object), indexOf(Object) // needs the equals method
  (3) adding elements, add() and addAll()
  (4) subList(int, int) // similar to substring()
  removing elements, remove(), clear(), and removeAll()
  combined with indexOf(), binarySearch()
  (5) List traversal
  (6) All of array processing methods can be done using arraylists.

`addAll(`**list**`)` `addAll(`**index**`, `**list**`)`	adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)
`contains(`**value**`)`	returns true if given value is found somewhere in this list
`containsAll(`**list**`)`	returns true if this list contains every element from given list
`equals(`**list**`)`	returns true if given other list contains the same elements
`iterator()` `listIterator()`	returns an object used to examine the contents of the list (seen later)
`lastIndexOf(`**value**`)`	returns last index value is found in list (-1 if not found)
`remove(`**value**`)`	finds and removes the given value from this list
`removeAll(`**list**`)`	removes any elements found in the given list from this list
`retainAll(`**list**`)`	removes any elements *not* found in given list from this list
`subList(`**from**`, `**to**`)`	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
`toArray()`	returns the elements in this list as an array

Topics in ArrayList Processing I

List Equality:

equals(list)

Searching:

contains(Object), indexOf(Object):

element by element comparison. (Comparable Objects)

Topics in ArrayList Processing I

SubList:

subList(from, end) // similar to substring():

get sub-list from the current list.

add(Object), addAll(list), addAll(index, list):

add all elements in the list to the current arraylist.

remove(Object), remove(index), removeAll(list):

remove all elements in list from current arraylist

retainAll(list):

keep only the element in the list from the current arraylist

Topics in ArrayList Processing I

- **List Conversion:**
- list.toArray():
- convert an arraylist to an array.
- Arrays.asList(array):
- convert an array to a list.

Searching

list.equals(list2)

- This function is used to check the equality of two lists. No the equality of elements. The equality check function of each object needs to be defined in the element class.

```
public static void equalityDemo() {
    System.out.println("Equality Example:..... ");
    // create an array list
    ArrayList<String> a1 = new ArrayList<String>();
    // add elements to the array list
    a1.add("C");
    a1.add("A");
    a1.add("E");
    a1.add("B");
    a1.add("D");
    a1.add("F");
    ArrayList<String> a2 = new ArrayList<String>(Arrays.asList(new String[]{
       "C", "A", "E", "B", "D", "F"
    }));
    System.out.println("Added List and Anonymous Array List are equal is "+a1.equals(a2));
}
```

contains(Objects)

Does the object exist in the arraylist, and where it is?

- The contains() method checks a given object is present in the list (and is equivalent to indexOf(elem)>=0).

```
public static void containsExample() {
        System.out.println("Arraylist contains() Example:..... ");
        ArrayList<Integer> arrlist = new ArrayList<Integer>();
        arrlist.add(20); arrlist.add(25); arrlist.add(10);
arrlist.add(15);
        for (Integer number : arrlist) {
          System.out.println("Number = " + number);
        }
        boolean retval = arrlist.contains(10);
        if (retval == true) {
            System.out.println("element 10 is contained in the list");
        }
        else {
            System.out.println("element 10 is not contained in the
list");
        }
}
```

```
Arraylist contains() Example:.........
Number = 20
Number = 25
Number = 10
Number = 15
```

indexOf(Objects)

Does the object exist in the arraylist, and where it is?

- The indexOf() method try to find the index for the element.

```
 public static void indexOfExample() {
       ArrayList<String> arrlist = new
ArrayList<String>(5);
       arrlist.add("G");
       arrlist.add("E");
       arrlist.add("F");
       arrlist.add("M");
       System.out.println("Size of list: " +
arrlist.size());
       for (String value : arrlist) {
         System.out.println("Value = " + value);
       }
       int retval=arrlist.indexOf("E");
       System.out.println("The element E is at index
" + retval);
}
```

```
ArrayList indexOf() Example:.........
Size of list: 4
Value = G
Value = E
Value = F
Value = M
The element E is at index 1
```

Sub-Lists

subList(start, end)

List subList(int fromIndex, int toIndex) exclusive of toIndex element

There are three obvious use cases for this method:

- Fast clearing parts of the list
- Using it for-each iteration over the part of the list
- Using it in the recursive algorithms.

subList(start, end)

List subList(int fromIndex, int toIndex) exclusive of toIndex element

```
public static void subListExample(){
  ArrayList<String> al = new
ArrayList<String>();
  al.add("Steve");
  al.add("Justin");
  al.add("Ajeet");
  al.add("John");
  al.add("Arnold");
  al.add("Chaitanya");
  System.out.println("Original ArrayList
Content: "+al);
  ArrayList<String> al2 = new
ArrayList<String>(al.subList(1, 4));
```

```
Arraylist subList(start, end) Example:.........
Original ArrayList Content: [Steve, Justin, Ajeet, John, Arnold, Chaitanya]
SubList stored in ArrayList: [Justin, Ajeet, John]
SubList stored in List: [Justin, Ajeet, John]
```

.add(): append an element
.addAll() concatenate another list

```
ArrayList<String> al = new ArrayList<String>();
al.add("Apple"); al.add("Orange");al.add("Grapes");al.add("Mango");
System.out.println("ArrayList1 before addAll:"+al);

//ArrayList2
ArrayList<String> al2 = new ArrayList<String>();
al2.add("Fig"); al2.add("Pear"); al2.add("Banana"); al2.add("Guava");
System.out.println("ArrayList2 content:"+al2);

//Adding ArrayList2 in ArrayList1 at 3rd position(index =2)
al.addAll(2, al2);
System.out.println("ArrayList1 after adding ArrayList2 at 3rd Pos:\n"+al);
```

```
Arraylist addAll() Example:.........
ArrayList1 before addAll:[Apple, Orange, Grapes, Mango]
ArrayList2 content:[Fig, Pear, Banana, Guava]
ArrayList1 after adding ArrayList2 at 3rd Pos:
[Apple, Orange, Fig, Pear, Banana, Guava, Grapes, Mango]
```

Two ways to remove()

remove(object) and remove(index)

- remove(object) is to remove the object in the ArrayList.
- remove(index) is to remove the object currently at index location.

```
1  import java.util.ArrayList;
2  public class TwoWayRemove1
3  {
4     public static void main(String[] args){
5         ArrayList<Integer> numbers = new ArrayList<Integer>();
6         numbers.add(1); numbers.add(2); numbers.add(3);
7         System.out.println("ArrayList contains: "+numbers);
8         numbers.remove(1); numbers.remove(3);
9         // 3 is int type, this will cause exception
10    }
11 }
```

```
ArrayList contains: [1, 2, 3]

 Can only enter input while your programming is running

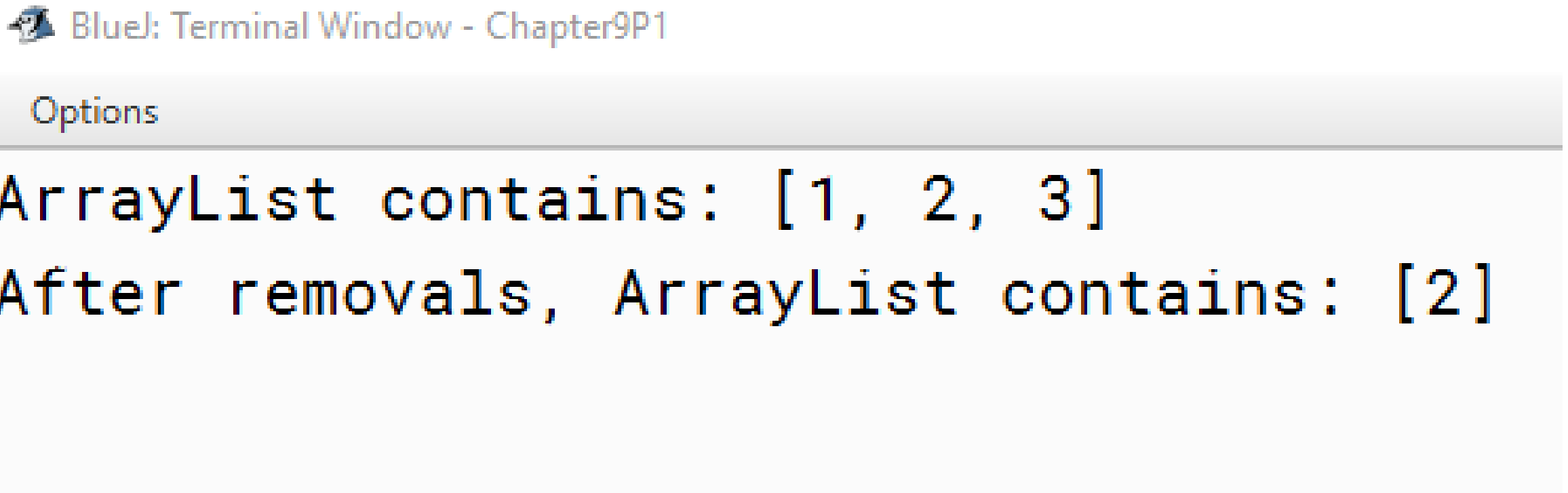
java.lang.IndexOutOfBoundsException: Index: 3, Size: 2
        at java.util.ArrayList.rangeCheck(ArrayList.java:653)
        at java.util.ArrayList.remove(ArrayList.java:492)
        at TwoWayRemove1.main(TwoWayRemove1.java:8)
```

Two ways to remove()

remove(object) and remove(index)

- remove(object) is to remove the object in the ArrayList.
- remove(index) is to remove the object currently at index location.

```
1  import java.util.ArrayList;
2  public class TwoWayRemove2
3  {
4     public static void main(String[] args){
5         ArrayList<Integer> numbers = new ArrayList<Integer>();
6         numbers.add(1); numbers.add(2); numbers.add(3);
7         System.out.println("ArrayList contains: "+numbers);
8         numbers.remove(new Integer(1)); numbers.remove(new Integer(3));
9         System.out.println("After removals, ArrayList contains: "+numbers);
10    }
11 }
```


BlueJ: Terminal Window - Chapter9P1
Options
ArrayList contains: [1, 2, 3]
After removals, ArrayList contains: [2]

removeAll(Collection<E> c)

like difference of sets (new A=A-B)

```
• public static void removeAllExample() {
        ArrayList<String> color_list = new
ArrayList<String>();
•       color_list.add("White");
•       color_list.add("Black");
•       color_list.add("Red");
•
•       ArrayList<String> sample = new ArrayList<String>();
•       sample.add("Green");
•       sample.add("Red");
•       sample.add("White");
•       sample.removeAll(color_list);
•
•       System.out.println("First List :"+ color_list);
•       System.out.println("Second List :"+ sample);
•    }
```

```
removeAll(Object) Example:.........
ArrayList contains : [1, 2, 3]
ArrayList after removal : [1]
```

List Traversal by Iterator

Traverse through ArrayList using Iterator

- Because ArrayList is of Collection type, it may not be always be convenient to use index to locate an object when the arraylists get too long. You need some stream handler to help.
- Iterator Class pointer (data stream handler) is the handler for ArrayList. It just behaves similarly to input stream handler (Scanner class) for file or console.

Remove Object from ArrayList using Iterator

Demo Program: RevemoByIterator.java

- Iterator behaves like input stream handler,
- `Scanner input = new Scanner(ifile);`
- `Iterator<Integer> itr = numbers.iterator();`

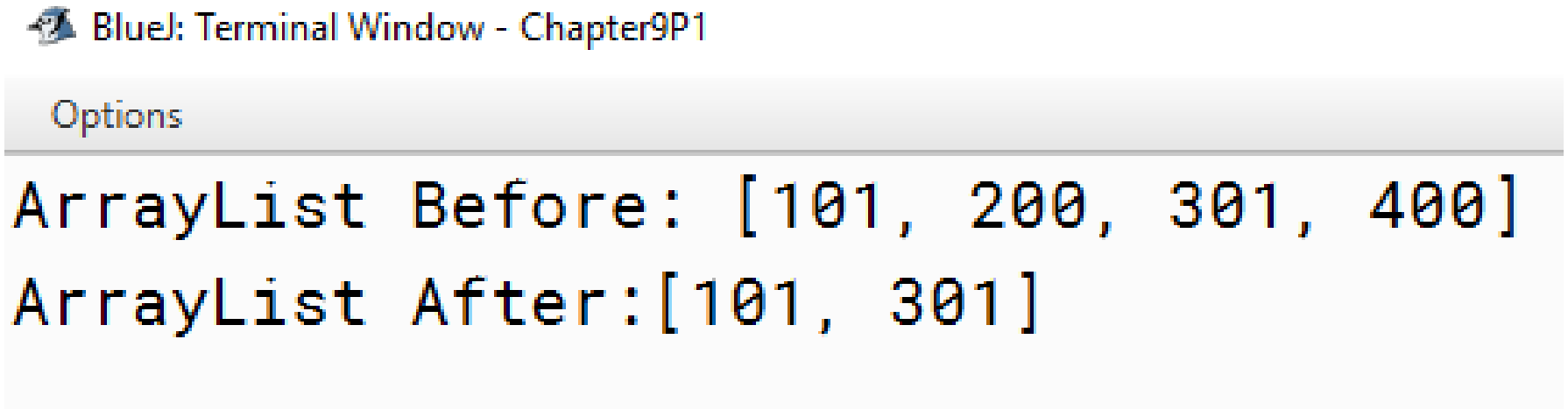
Traversal by Iterator:

```
Iterator<Integer> itr = numbers.iterator();
      while (itr.hasNext()){
         Integer number = itr.next();
         if (number %2 ==0){
            itr.remove();
          }
       }
```

Analogy:

```
Scanner in = new Scanner (System.in);
while (in.hasNext()){
      int number = in.nextInt();
      if (number%2==0){
        System.out.println(number);
      }
}
```

```java
import java.util.*;
public class RemoveByIterator
{
    public static void main(String[] args){
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(101); numbers.add(200); numbers.add(301); numbers.add(400);
        System.out.println("ArrayList Before: "+numbers);
        Iterator<Integer> itr = numbers.iterator(); // remove all even numbers;
        while (itr.hasNext()){
            Integer number = itr.next();
            if (number %2 ==0){
                itr.remove();
             }
         }
        System.out.println("ArrayList After:"+numbers);
    }
}
```


BlueJ: Terminal Window - Chapter9P1
Options
ArrayList Before: [101, 200, 301, 400]
ArrayList After:[101, 301]

Remove Object from ArrayList using Iterator

Demo Program: RemoveByIterator2.java

- This is actually a subtle details of Java Programming, not obvious for first timers, as compiler will not complain, even if you sue remove() method from java.util.ArrayList, while using Iterator.
- You will only realize your mistake, when you see ConcurrentModificationException, which itself is misleading and you may spend countless hours finding another thread, which is modifying that ArrayList, because of Concurrent word. Let's see an example.
- Iterator is not an index!!!
- Iterator is a reference.

```java
import java.util.*;
public class RemoveByIterator2
{
    public static void main(String[] args){
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(101); numbers.add(200); numbers.add(301); numbers.add(400);
        System.out.println("ArrayList Before: "+numbers);
        Iterator<Integer> itr = numbers.iterator(); // remove all even numbers;
        while (itr.hasNext()){
            Integer number = itr.next();
            if (number %2 ==0){
                numbers.remove(number);
             }
         }
        System.out.println("ArrayList After:"+numbers);
    }
}
```

```
ArrayList Before: [101, 200, 301, 400]
```

```
Can only enter input while your programming is running
```

```
java.util.ConcurrentModificationException
        at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)
        at java.util.ArrayList$Itr.next(ArrayList.java:851)
        at RemoveByIterator.main(RemoveByIterator.java:10)
```

Demonstration Program

ArrayListProcessing1.java

```
Equality Example:.........
Added List and Anonymous Array List are equal is true
Arraylist contains() Example:.........
Number = 20
Number = 25
Number = 10
Number = 15
element 10 is contained in the list
ArrayList indexOf() Example:.........
Size of list: 4
Value = G
Value = E
Value = F
Value = M
The element E is at index 1
Arraylist subList(start, end) Example:.........
Original ArrayList Content: [Steve, Justin, Ajeet, John, Arnold, Chaitanya]
SubList stored in ArrayList: [Justin, Ajeet, John]
SubList stored in List: [Justin, Ajeet, John]
Arraylist addAll() Example:.........
ArrayList1 before addAll:[Apple, Orange, Grapes, Mango]
ArrayList2 content:[Fig, Pear, Banana, Guava]
ArrayList1 after adding ArrayList2 at 3rd Pos:
[Apple, Orange, Fig, Pear, Banana, Guava, Grapes, Mango]
```

```
removeAll(Object) Example:.........
ArrayList contains : [1, 2, 3]
ArrayList after removal : [1]
Remove Using Iterator itr.nremove() Example:
ArrayList Before : [101, 200, 301, 400]
ArrayList After : [101, 301]
removeAll(Collection<E> c) Example:.........
First List :[White, Black, Red]
Second List :[Green]
```

List Processing I

Conversion of Array Processing Methods to ArrayList Processing

(Lab: saved for students' practice: ArrayListProcessingArrayProcessing.java)

1. Initializing arrays with input values.
2. Initializing arrays with random values
3. Printing arrays
4. Summing all elements
5. Finding the largest element
6. Finding the smallest index of the largest element
7. Random shuffling
8. Shifting elements

Lab

Array Processing to

ArrayList Processing

```
ArrayList contains: [1, 2, 3]
After removals, ArrayList contains: [2]
Enter 5 values: 4 8 12 7 9
Array After Initialization: [4.0, 8.0, 12.0, 7.0, 9.0]

Array After Random Input: [80.58, 13.59, 67.35, 7.48, 0.39]

Sum of myList:   169.3900
max of myList:    80.5800
Index   0 has the max of myList:    80.5800
After Random Shuffle: [13.59, 7.48, 67.34, 0.39, 80.58]

After Left Shifting: [7.48, 67.34, 0.39, 80.58, 13.59]

After Right Shifting: [13.59, 7.48, 67.34, 0.39, 80.58]
```