Array Lists

Array Lists

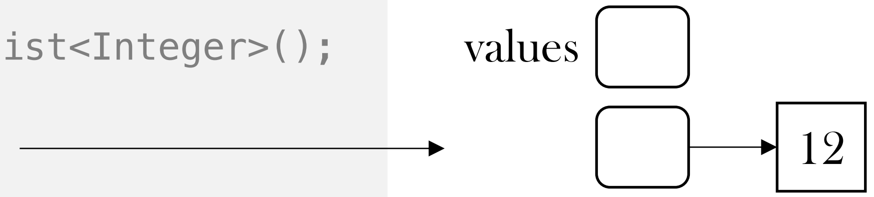
- Arrays can store multiple elements, but their size is fixed
  - This is an important limitation of arrays
- Array lists are more flexible than arrays
  - You can add and remove elements to an array list dynamically
  - Size of an array list changes dynamically
- Example:
  - Store many integers in a structure but you do not know how many integers will be stored
  - With arrays, you should pre-allocate the array

Array Lists: Defining an Array List

```
import java.util.ArrayList; // import ArrayLists

public class App {
 public static void main(String[] args) {

   // create an array list
   ArrayList<Integer> values = new ArrayList<Integer>();

   values.add(12); // add integer values
   values.add(4);
   values.add(80);
   values.add(7);

   System.out.println(values); // print array list


   values.remove(Integer.valueOf(4)); // remove 4
   System.out.println(values); // print array list
 }
}
```

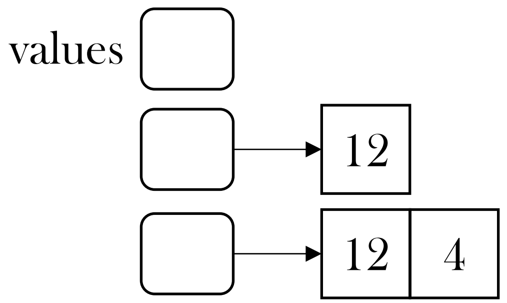
Array Lists: Adding Elements

```
import java.util.ArrayList; // import ArrayLists

public class App {
 public static void main(String[] args) {

   // create an array list
   ArrayList<Integer> values = new ArrayList<Integer>();

   values.add(12); // add integer values
   values.add(4);
   values.add(80);
   values.add(7);

   System.out.println(values); // print array list


   values.remove(Integer.valueOf(4)); // remove 4
   System.out.println(values); // print array list
 }
}
```



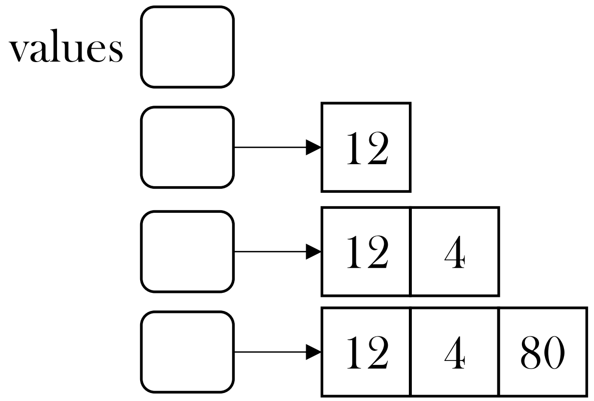
Array Lists

```
import java.util.ArrayList; // import ArrayLists

public class App {
 public static void main(String[] args) {

   // create an array list
   ArrayList<Integer> values = new ArrayList<Integer>();

   values.add(12); // add integer values
   values.add(4);
   values.add(80);
   values.add(7);

   System.out.println(values); // print array list


   values.remove(Integer.valueOf(4)); // remove 4
   System.out.println(values); // print array list
 }
}
```



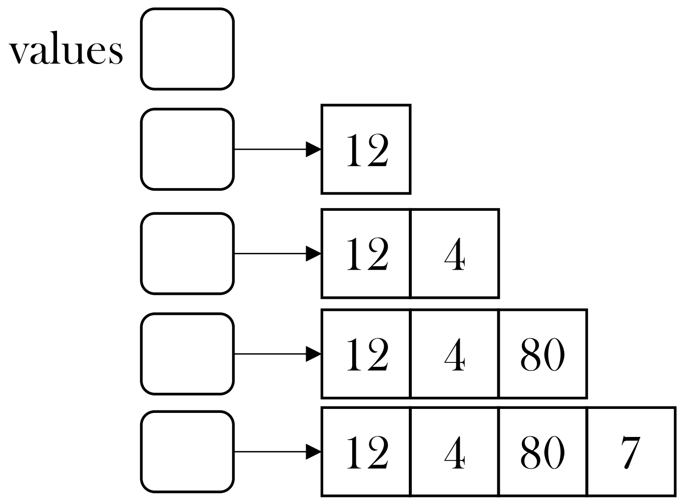
Array Lists

```
import java.util.ArrayList; // import ArrayLists

public class App {
 public static void main(String[] args) {

   // create an array list
   ArrayList<Integer> values = new ArrayList<Integer>();

   values.add(12); // add integer values
   values.add(4);
   values.add(80);
   values.add(7);

   System.out.println(values); // print array list


   values.remove(Integer.valueOf(4)); // remove 4
   System.out.println(values); // print array list
 }
}
```



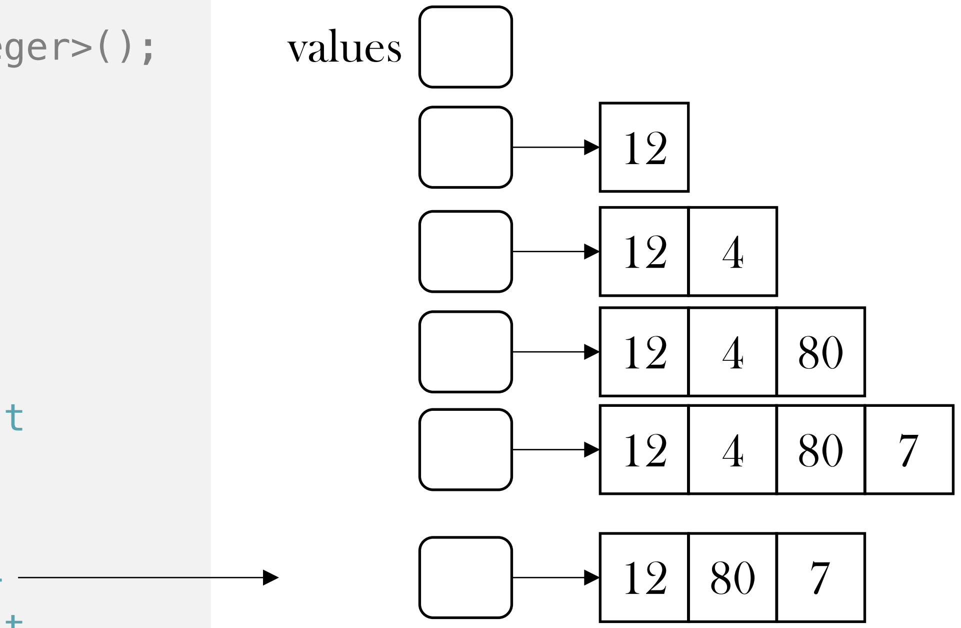
Array Lists

```
import java.util.ArrayList; // import ArrayLists

public class App {
 public static void main(String[] args) {

   // create an array list
   ArrayList<Integer> values = new ArrayList<Integer>();

   values.add(12); // add integer values
   values.add(4);
   values.add(80);
   values.add(7);

   System.out.println(values); // print array list


   values.remove(Integer.valueOf(4)); // remove 4
   System.out.println(values); // print array list
 }
}
```



Array Lists

```java
import java.util.ArrayList; // import ArrayLists

public class App {
 public static void main(String[] args) {

   // create an array list
   ArrayList<Integer> values = new ArrayList<Integer>();

   values.add(12); // add integer values
   values.add(4);
   values.add(80);
   values.add(7);

   System.out.println(values); // print array list


   values.remove(Integer.valueOf(4)); // remove 4
   System.out.println(values); // print array list
 }
}
```



Declaring and Creating Array Lists

- You can declare an array list as shown below:

```
ArrayList<Integer> list1; // declare an array list
ArrayList<Double> list2;
ArrayList<String> list3;
ArrayList<Student> list4;
```

- You can declare and create an array list in a single statement

```
ArrayList<Integer> list1 = new ArrayList<Integer>(); // declare and create an array list
ArrayList<Student> list4 = new ArrayList<Student>();
```

- You can omit the last Integer in array lists during creation:

```
ArrayList<Integer> list1 = new ArrayList<>(); // Integer is omitted here
```

Wrapper Types for Primitive Types

- Array lists can only store objects: You can not store primitive types such as integer and double types in array lists
- Java provides wrapper classes for primitive types, that can be used in array lists
  - Integer: int
  - Double: double
  - Others are: Boolean, Character, Float, Byte, Short, and Long
- When you add an integer to an array list, you actually insert an Integer object, not an int

```
ArrayList<Integer> list1 = new ArrayList<Integer>();
list1.add(13); // here 13 is an Integer object. It is not an int.
```

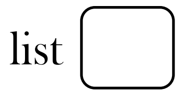
Import Array Lists

- In order to use array lists, import `java.util.ArrayList`

Array List Operations: Add

- Add method appends/adds a new element at the end of an array list

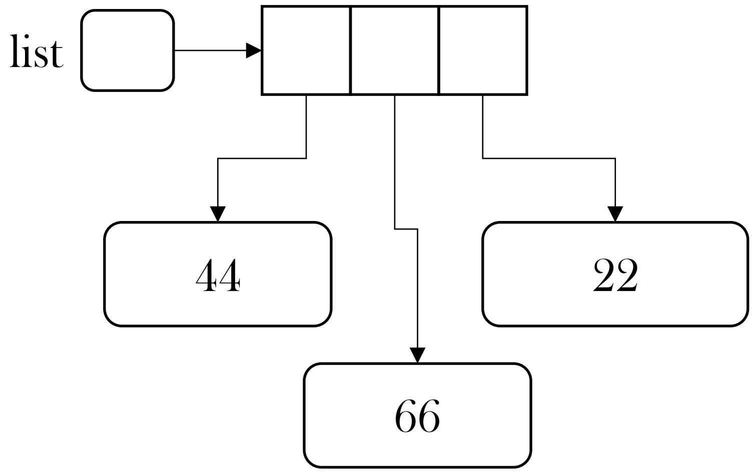
```
ArrayList<Integer> list = new ArrayList<Integer>();
```



Array List Operations: Add

- Add method appends/adds a new element at the end of an array list

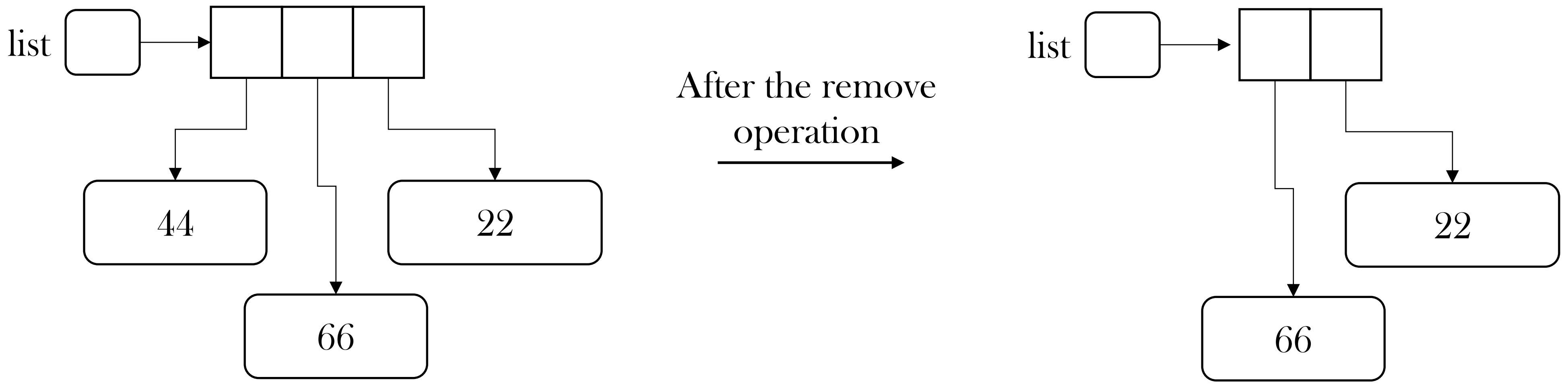
```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add( 44 ); // add integer value 44
list.add( 66 );
list.add( 22 );
System.out.println(“Size of the list: ” + list.size()); // prints 3
```



Array List Operations: Remove

- Add method appends/adds a new element at the end of an array list

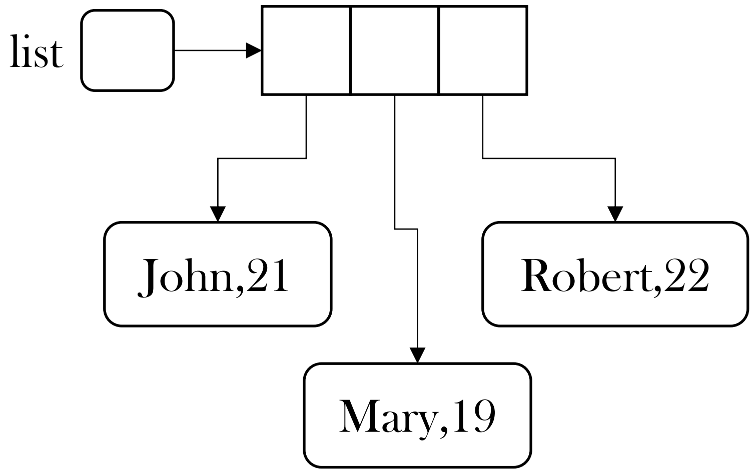
```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add( 44 ); // add integer value 44
list.add( 66 );
list.add( 22 );
list.remove( 0 ); // remove the first item
```



Array List Operations: Add

- Add method appends/adds a new element at the end of an array list

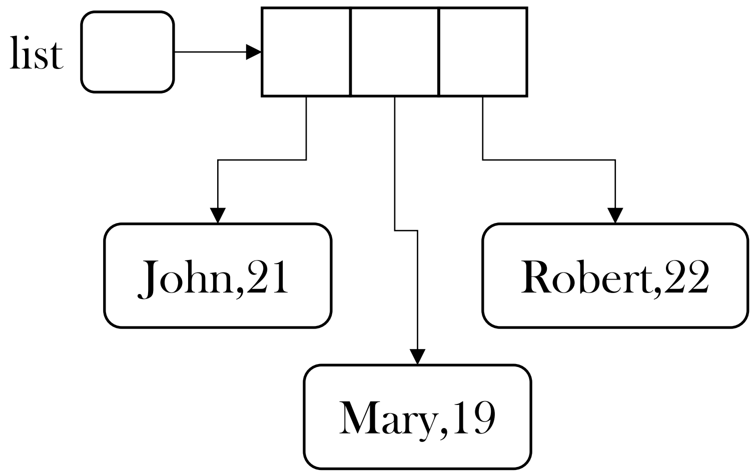
```
ArrayList<Student> list = new ArrayList<Student>();
list.add( new Student(“John”,21) ); // add student John
list.add( new Student(“Mary”,19) ); // add student Mary
list.add( new Student(“Robert”,22) ); // add student Robert
```



Array List Operations: Size

- `size()` method returns the size of an array list

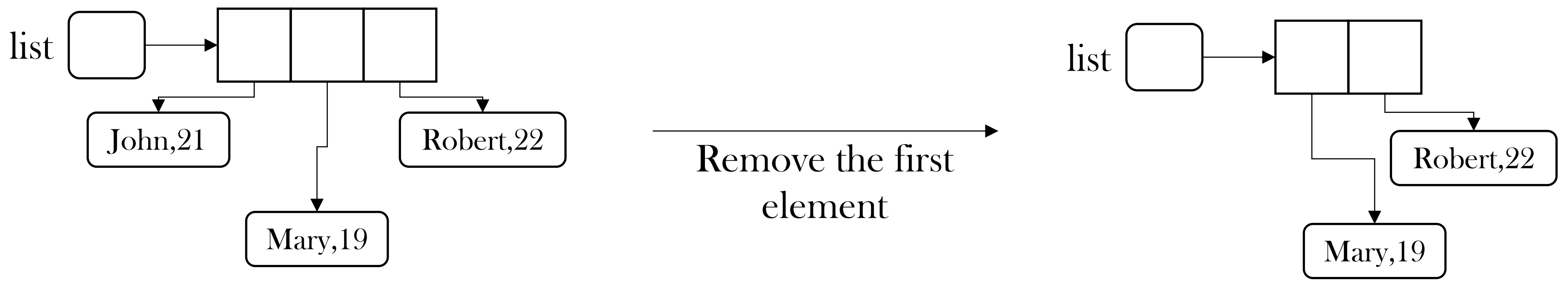
```
ArrayList<Student> list = new ArrayList<Student>();
list.add( new Student(“John”,21) ); // add student John
list.add( new Student(“Mary”,19) ); // add student Mary
list.add( new Student(“Robert”,22) ); // add student Robert
System.out.println(“Size of the list: ” + list.size()); // prints 3
```



Array List Operations: Remove 1/2

- There are two types of remove operations
  - Remove an element by its location, e.g., remove the 1st element
  - Remove an element by its content, e.g., remove student John
- Example: Remove the first student from the list

```
ArrayList<Student> list = new ArrayList<Student>();
list.add( new Student(“John”,21) ); // add student John
list.add( new Student(“Mary”,19) ); // add student Mary
list.add( new Student(“Robert”,22) ); // add student Robert
list.remove(0) // remove the first student
```



Array List Operations: Remove 2/2

- Remove an element by its content
  - Example: An array list stores the names of cities. Remove “Istanbul”

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");
System.out.println(cities);

cities.remove("Istanbul"); // remove “Istanbul”
System.out.println(cities);
```

Program output

```
[New York, Istanbul, Barcelona]
[New York, Barcelona]
```

Removing an Integer by Value

- In an integer array list, how to remove the element containing the value of 0?

```
ArrayList<Integer> myList = new ArrayList<>();
myList.add(2);
myList.add(1);
myList.add(0);
myList.add(3);

//a.remove(0); // removes the first element, not third element with value 0
myList.remove(Integer.valueOf(0)); // removes the third element
```

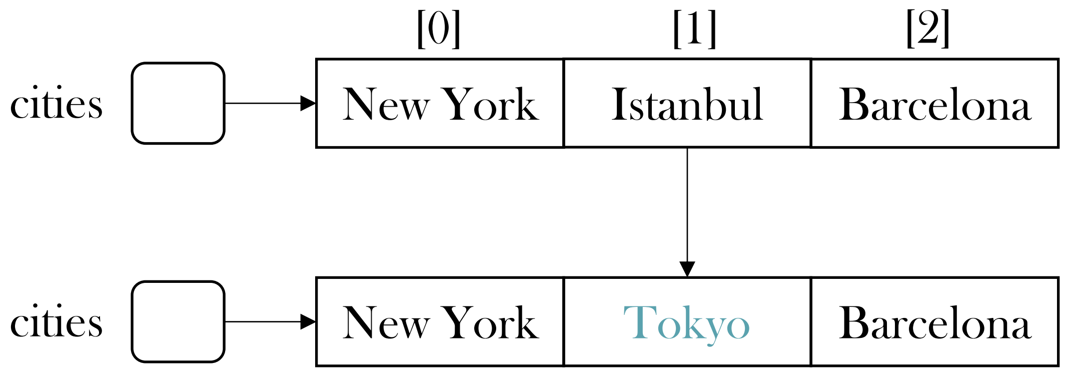
Program output

```
[2,1,3]
```

Array List Operations: Set

- You can set an element by providing its index. Set method overwrites existing value

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");
System.out.println(cities);

cities.set(1,”Tokyo”); // set the second element as Tokyo
System.out.println(cities);
```



Program output

```
[New York, Istanbul, Barcelona]
[New York, Tokyo, Barcelona]
```

Array List Operations: Get

- You can get an element using get() method

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");

String lastCity = cities.get(2); // get the last element
System.out.println(lastCity);
```

Program output

```
[New York, Istanbul, Barcelona]
Barcelona
```

Array List Operations: Add to a Position

- You can insert an element into an array list
- For example, the call `cities.add(1, "Ann")` adds a new element at position 1 and moves all elements with index 1 or larger by one position

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");
System.out.println(cities);

cities.add(1, "Paris"); // insert Paris at position 1
System.out.println(cities);
```

Program output

```
[New York, Istanbul, Barcelona]
[New York, Paris, Istanbul, Barcelona]
```

Array List Operations: Contains

- `contains()` method returns true if the element you search is in the array list. Otherwise, it returns false

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");

boolean result = cities.contains("Istanbul");
System.out.println(result); // prints true
```

Array List Operations: Deleting All Elements

- You can use `clear()` method to delete all elements in an array list

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");
System.out.println(cities);

// clear all elements in the array list
cities.clear();
System.out.println(cities);
```

Program output

```
[New York, Istanbul, Barcelona]
[]
```

Array List Operations: Searching

- `indexOf()` method returns the first position of an element you search for
  - If the element is not found, returns -1
- `lastIndexOf()` method returns the index of the last matching element

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");

int index = cities.indexOf("Barcelona");
System.out.println("Location of Barcelona: " + index);

// search for Berlin
System.out.println("Location of Berlin: " + cities.indexOf("Berlin"));
```

Program output

```
[New York, Istanbul, Barcelona]
Location of Barcelona: 2
Location of Berlin: -1
```

Array List Operations: Is Empty?

- You can check if the array list is empty or not using the `isEmpty()` method

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");

// check if the list is empty or not?
System.out.println("Is the list empty? : " + cities.isEmpty());
```

Program output

```
Is the list empty? : false
```

Printing Array List

- You can print the contents of an array list as shown below

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");
System.out.println(cities); // print the contents of an array list
```

For-each Loop with Array Lists

- You can traverse the array list elements using for-each loop

```
ArrayList<Student> list = new ArrayList<>();
list.add( new Student("John", 21));
list.add( new Student("Mary", 19));
list.add( new Student("Robert", 22));

for (Student student : list)
  System.out.println(student);
```

Copying Array Lists

- You can copy array lists using constructors

```
ArrayList<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Istanbul");
cities.add("Barcelona");
System.out.println(cities);

// copy array list using constructor
ArrayList<String> citiesCopy = new ArrayList<>(cities);

// add a new element to the copy
citiesCopy.add("Lisbon");
System.out.println(citiesCopy);
```

Program output

```
[New York, Istanbul, Barcelona]
[New York, Istanbul, Barcelona, Lisbon]
```

Choosing between Array Lists and Arrays

- Array lists are flexible: they can grow and shrink
- Arrays are faster (especially for primitive types)

- When to choose arrays
  - If the size of a collection never changes, use an array
  - If you have a long sequence of primitive types and you are concerned about efficiency, use an array
- Otherwise, use an array list

Comparing Array and Array List Operations

Operation	Arrays	Array Lists
Creating the collection	`int[] values = new int[10];`	`ArrayList<Integer> values = new ArrayList<>();`
Get an element	`x = values[4];`	`x = values.get(4);`
Replace an element	`values[4] = 35;`	`values.set(4,35);`
Add an element	Not available	`values.add(35);`
Size of the collection	`values.length`	`values.size()`
Removing an element	Not available	`values.remove(index);`
Clearing the collection	Not available	`values.clear();`

Searching for an element	Not available	`values.indexOf(4);`
		`values.lastIndexOf(5);`
		`values.contains(4);`

Examples

Array Lists

Example: Finding Unique Values

- Write a program which gets integers from the user, ending with zero. Display only the unique integers, i.e., the distinct numbers entered
- For example, if the user enters 1, 9, 2, 2, 1, 4, 3, 3, 1, 1, 0, the output should be 1, 9, 2, 4, 3

Source Code

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

public class App {
  public static void main(String[] args) {

    ArrayList<Integer> list = new ArrayList<>();
    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

      // Add integer if it is not in the list
      // Exit from the loop if the input integer is zero
      if (!list.contains(value) && value != 0)
        list.add(value);
    } while (value != 0);

    // Print the distinct numbers
    System.out.println("Distinct numbers are: " + list);
    input.close();
  }
}
```

Example: Find Expensive Products (1/2)

- Amazon stores all of its products in an array list. Find all products which are more expensive than 200TL, and store them in a separate array list

```
import java.util.ArrayList;

public class App {
  public static void main(String[] args) {

    // all products in an array list
    ArrayList<Product> products = new ArrayList<>();
    products.add( new Product("Nike",160));
    products.add( new Product("Adidas",180));
    products.add( new Product("Nike",260));
    products.add( new Product("New Balance",300));
    products.add( new Product("Vans",280));
    products.add( new Product("Vans",380));
    products.add( new Product("Camper",380));
    products.add( new Product("Adidas",120));

    // code continues

    }
}
```

Example: Find Expensive Products (2/2)

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

  // code continues from here

  printProducts(products); // print all products
  int priceThreshold = 200;

  // find expensive products and add them to the new array list
  ArrayList<Product> expensiveProducts = new ArrayList<>();
  for (Product product : products)
    if (product.getPrice() > priceThreshold)
      expensiveProducts.add(product);

  System.out.println("Expensive Products:");
  printProducts(expensiveProducts); // print expensive products
}

// method prints products using toString method
private static void printProducts(ArrayList<Product> products) {
  for (Product p : products)
    System.out.println(p);
}
```

Program output

```
Expensive Products:
[brand=Nike, price=260.0]
[brand=New Balance, price=300.0]
[brand=Vans, price=280.0]
[brand=Vans, price=380.0]
[brand=Camper, price=380.0]
```