Programming Languages - II
Dynamic Arrays

Özgür Koray ŞAHİNGÖZ
Prof.Dr.

Biruni University
Computer Engineering Department

Change Array Size

```
public class Main {
    public static void main(String[] args) {
    int[] numberArray = { 12, 24, 63, 45 };
    System.out.println("Array before ReSize: ");
    for (int i = 0; i < numberArray.length; i++)
        System.out.println(numberArray[i]);

    numberArray = new int[6];
    numberArray[4]=71;
    numberArray[5]=98;

    System.out.println("Array after ReSize: ");

    for (int i = 0; i < numberArray.length; i++)
        System.out.println(numberArray[i]);
    } }
```

Change Array Size

```
public class Main {
  public static void main(String[] args) {
  int[] numberArray = { 12, 24, 63, 45 };
  System.out.println("Array before ReSize: ");
  for (int i = 0; i < numberArray.length; i++)
    System.out.println(numberArray[i]);

  int[] temp = new int[6];
  int length = numberArray.length;

  for (int j = 0; j < length; j++)
    temp[j] = numberArray[j];
  numberArray = temp;
  System.out.println("Array after ReSize: ");

  for (int i = 0; i < numberArray.length; i++)
    System.out.println(numberArray[i]);
  } }
```

Solution

- This is not a dynamic structure

- To use a dynamic (easily resized arrays we need to get a help from LIBRARIES.

■ Vectors	Relatively bad approach
■ ArrayLists	Good Approach

Vectors and arrays

- A Vector is like an array of Objects
- Differences between arrays and Vectors:
  - Arrays have special syntax; Vectors don't
  - You can have an array of any type, but a Vector holds Objects
  - An array is a fixed size, but a Vector expands as you add things to it
    - This means you don't need to know the size beforehand

Creating a Vector

- import java.util.*;
- Vector vec1 = new Vector();
- Vector vec2 = new Vector(***initialSize***);
- Vector vec3 = new Vector(***initialSize***, ***increment***);

Adding elements to a Vector

- boolean add(Object ***obj***)
  - Appends the object ***obj*** to the end of this Vector
  - Always returns true
    - This is for consistency with other, similar classes
- void add(int ***index***, Object ***element***)
  - Inserts the ***element*** at position ***index*** in this Vector
  - The ***index*** must be greater than or equal to zero and less than or equal to the number of elements in the Vector

Removing elements from a Vector

- boolean remove(Object ***obj***)
  - Removes the first occurrence of ***obj*** from this Vector
  - Returns true if an element was removed

- void remove(int ***index***)
  - Removes the element at position ***index*** from this Vector

- void removeAllElements()
  - Removes all elements

Accessing elements of a Vector

- Object elementAt(int ***index***)   *or*
  Object get(int ***index***)
  - Returns the component at position ***index***
  - elementAt is an older method, retained for compatibility with older programs
- Object firstElement()
  - Returns the component at location 0
- Object lastElement()
  - Returns the last component

Searching a Vector

- boolean contains(Object ***element***)
  - Tests if ***element*** is a component of this Vector

- int indexOf(Object ***element***)
  - Returns the index of the first occurrence of ***element*** in this Vector
  - Returns -1 if ***element*** was not found in this Vector

- int indexOf(Object ***element***, int ***index***)
  - Returns the index of the first occurrence of ***element*** in this Vector, beginning the search at ***index***
  - Returns -1 if ***element*** was not found in this Vector

Searching a Vector II

- int lastIndexOf(Object ***element***)
  - Returns the index of the last occurrence of ***element*** in this Vector
  - Returns -1 if ***element*** was not found in this Vector

- int lastIndexOf(Object ***element***, int ***index***)
  - Returns the index of the last occurrence of ***element*** in this Vector, searching backward from ***index***
  - Returns -1 if ***element*** was not found in this Vector

- All searching is done using equals

Getting information about a Vector

- boolean isEmpty()
  - Returns true if this Vector has no elements

- int size()
  - Returns the number of elements currently in this Vector

- Object[ ] toArray()
  - Returns an array containing all the elements of this Vector in the correct order

```
import java.util.*;
public class VectorDemo {

  public static void main(String args[]) {
    // initial size is 3, increment is 2
    Vector v = new Vector(3, 2);
    System.out.println("Initial size: " + v.size());
    System.out.println("Initial capacity: " + v.capacity());

    v.addElement(new Integer(1));
    v.addElement(new Integer(2));
    v.addElement(new Integer(3));
    v.addElement(new Integer(4));
    System.out.println("Capacity after four additions: " +
v.capacity());

    v.addElement(new Double(5.45));
    System.out.println("Current capacity: " + v.capacity());

    v.addElement(new Double(6.08));
    v.addElement(new Integer(7));
    System.out.println("Current capacity: " + v.capacity());

    v.addElement(new Float(9.4));
    v.addElement(new Integer(10));
    System.out.println("Current capacity: " + v.capacity());

    v.addElement(new Integer(11));
    v.addElement(new Integer(12));
    System.out.println("First element: " +
(Integer)v.firstElement());
    System.out.println("Last element: " +
(Integer)v.lastElement());

    if(v.contains(new Integer(3)))
      System.out.println("Vector contains 3.");

    // enumerate the elements in the vector.
    Enumeration vEnum = v.elements();
    System.out.println("\nElements in vector:");

    while(vEnum.hasMoreElements())
      System.out.print(vEnum.nextElement() + " ");
    System.out.println();
  }
}
```

Output

```
Initial size: 0
Initial capacity: 3
Capacity after four additions: 5
Current capacity: 5
Current capacity: 7
Current capacity: 9
First element: 1
Last element: 12
Vector contains 3.

Elements in vector:
1 2 3 4 5.45 6.08 7 9.4 10 11 12
```

Java ArrayList

- The ArrayList class is a resizable array, which can be found in the java.util package.

- The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different:

What is an ArrayList in Java?

- An ArrayList is a dynamic length collection framework in Java that also stores the elements of the same type but here we do not need to mention the size at the time of its creation as the case in arrays.

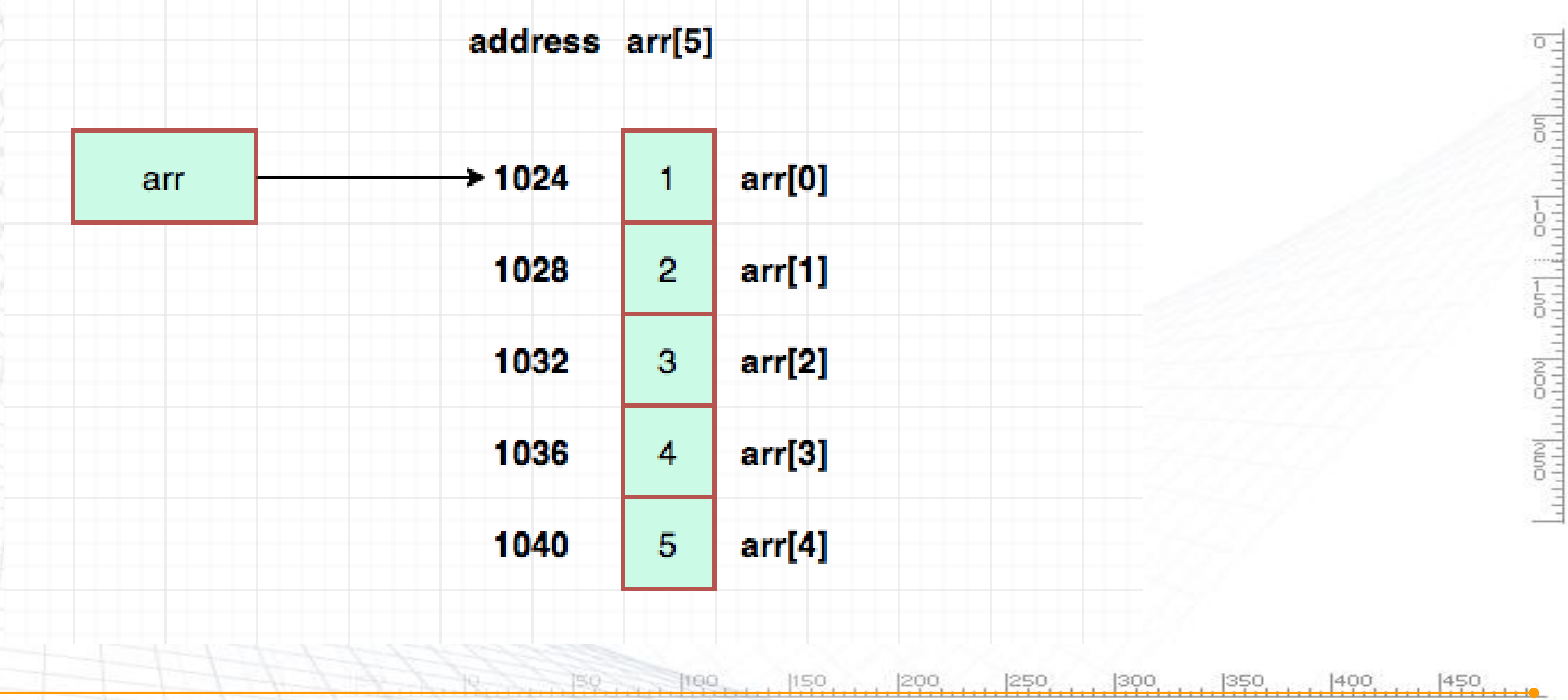
Creating Arrays using ArrayList:

```
ArrayList<data_type> objectName = new ArrayList<data_type>();
```

Example:

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

Array in Memory

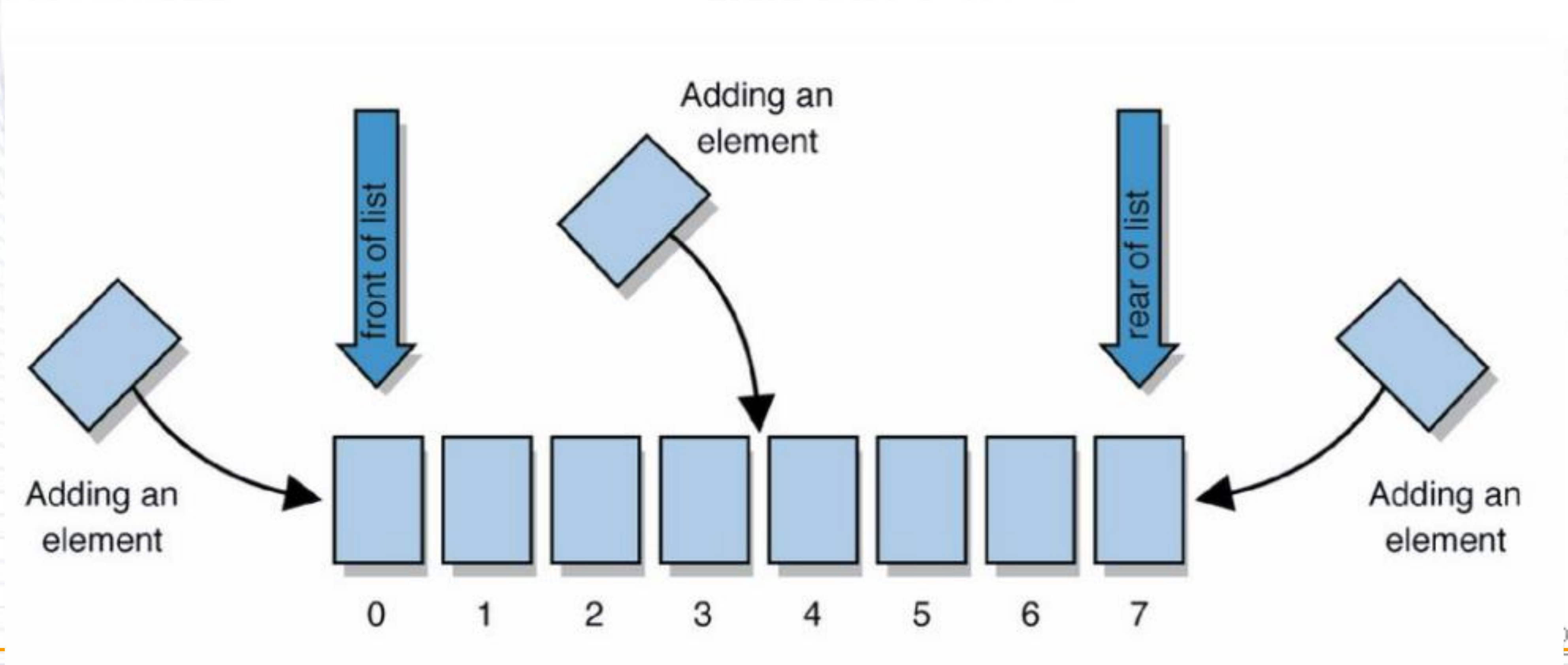


arr

address	arr[5]	
1024	1	arr[0]
1028	2	arr[1]
1032	3	arr[2]
1036	4	arr[3]
1040	5	arr[4]

This is not an array, it is ArrayList

- Each elements are connected with previous ones and next ones.



Example

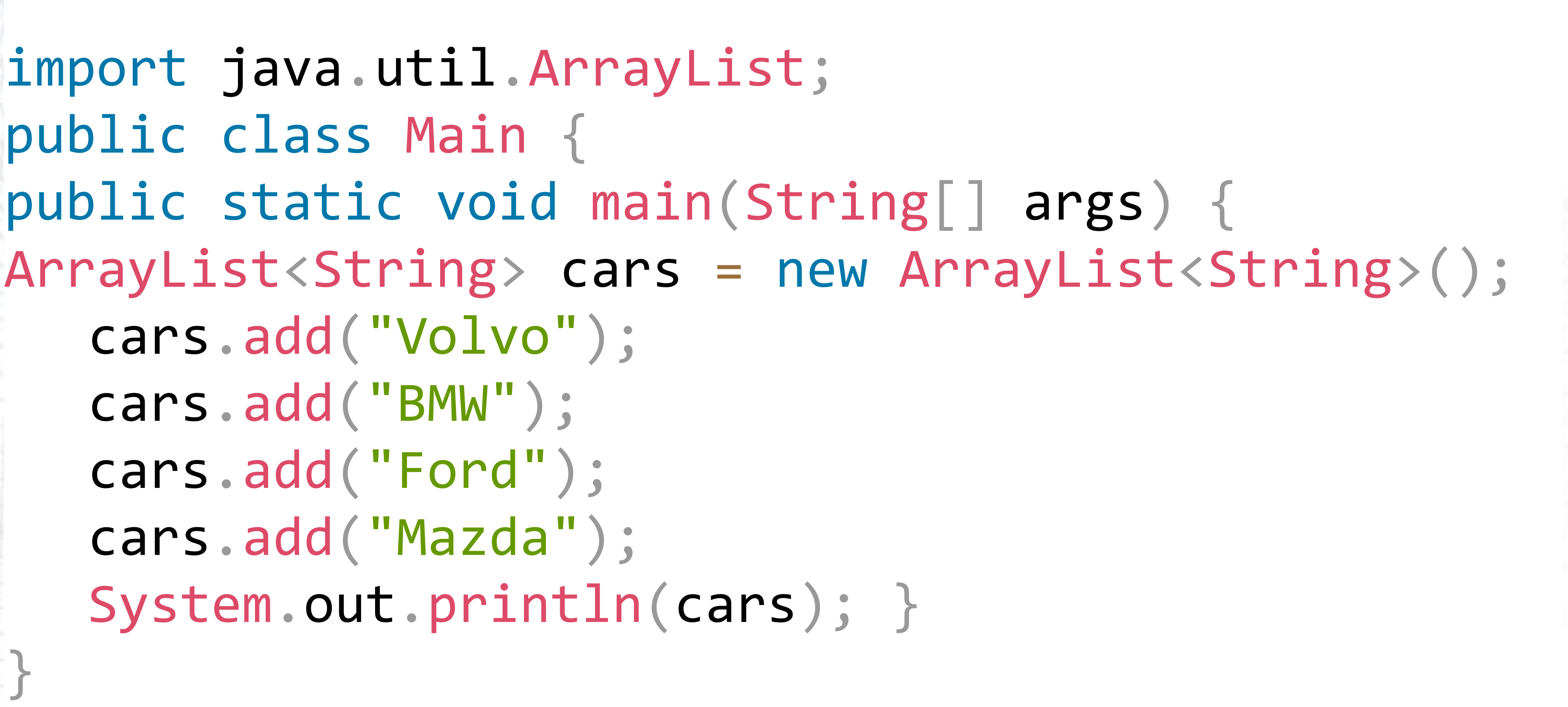
- Create an ArrayList object called cars that will store strings:

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

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

Add Items

The ArrayList class has many useful methods.

For example, to add elements to the ArrayList, use the add() method:



```
import java.util.ArrayList;
public class Main {
public static void main(String[] args) {
ArrayList<String> cars = new ArrayList<String>();
  cars.add("Volvo");
  cars.add("BMW");
  cars.add("Ford");
  cars.add("Mazda");
  System.out.println(cars); }
}
```

Access an Item

To access an element in the `ArrayList`, use the `get()` method and refer to the index number:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars.get(0));
  }
}
```

Change an Item

- To modify an element, use the set() method and refer to the index number:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.set(0, "Opel");
    System.out.println(cars);
  }
}
```

Remove an Item (and clear)

- To remove an element, use the remove() method and refer to the index number:

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.remove(0);
    System.out.println(cars);
  }
}
```

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.clear();
    System.out.println(cars);
  }
}
```

ArrayList Size

- To find out how many elements an ArrayList have, use the size method:

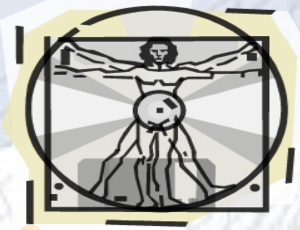
```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars.size());
  }
}
```

Loop Through an ArrayList

- Loop through the elements of an ArrayList with a for loop, and use the size() method to specify how many times the loop should run:

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (int i = 0; i < cars.size(); i++) {
      System.out.println(cars.get(i));
    }
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```



Other Types

- Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent wrapper class: Integer. For other primitive types, use: Boolean for boolean, Character for char, Double for double, etc:

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(10);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(25);
    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Sort an ArrayList

Another useful class in the java.util package is the Collections class, which include the sort() method for sorting lists alphabetically or numerically:

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    Collections.sort(cars);  // Sort cars
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Collections.sort(myNumbers);  // Sort myNumbers

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Difference between Array and ArrayList

Nature

- Arrays in Java are static in nature, i.e we can not change their length. The length of the array is fixed. Once we declare the length at the time of array creation, we can not change its size again.
- On the other hand, ArrayList in Java is dynamic in nature, therefore we also sometimes call it a re-sizeable array or dynamic array. ArrayList can automatically grow in their size if we add more limits beyond its defined capacity, therefore it is dynamic in nature.

Difference between Array and ArrayList

Implementation

- An array is a fundamental feature of Java, while ArrayList is a part of the Collection Framework API in Java. ArrayList in Java is internally implemented using Arrays. ArrayList is a class that carries all the properties of a normal class; we can create objects from it and call methods with the object.
- While an Array is an object in Java but there is no method that we can call using this object. An array just has a single attribute called length that too is constant.

Difference between Array and ArrayList

Performance

- Since ArrayList internally works based on the array, you may think that performance of both of them would be the same.
- Basically, it can be considered true but the performance of ArrayList may be slower as compared to Arrays because it has some extra functionality other than Arrays. The performance of ArrayList affects mainly in terms of CPU time and memory usage.
- Any resize() operation on ArrayList may degrade the performance of ArrayList since it involves the creation of a new array and then copying the content from the old array to the new array. This operation, therefore, slows down the performance of ArrayList.

Difference between Array and ArrayList

Flexibility

- Flexibility is the most important factor that significantly differentiates the array and ArrayList in Java. ArrayList is more flexible as compared to Arrays in Java. This is because ArrayList is dynamic in nature. ArrayList can grow automatically when the elements are being added beyond its capacity, which is not possible with arrays. Moreover, ArrayList also allows us to remove elements from it while it is not possible with arrays. We can’t remove elements from an array after adding them.

Difference between Array and ArrayList

Dimension

This is another significant difference between an Array and an ArrayList.An Array can be single dimensional or multi-dimensional; i.e., it can also have multiple dimensions that help us to represent 2-D and 3-D objects.

On the other hand, ArrayList can not be multi-dimensional, it can only be one or single-dimensional. In fact, it does not allow you to specify the dimension. It is by default one-dimensional in nature.