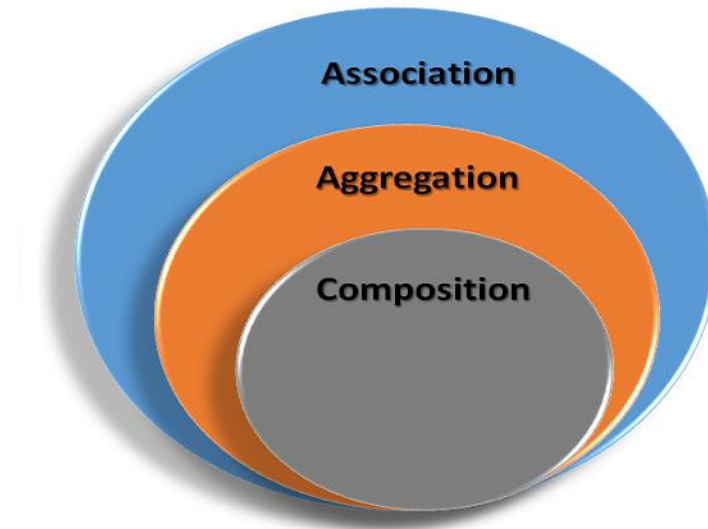




Relations between Classes

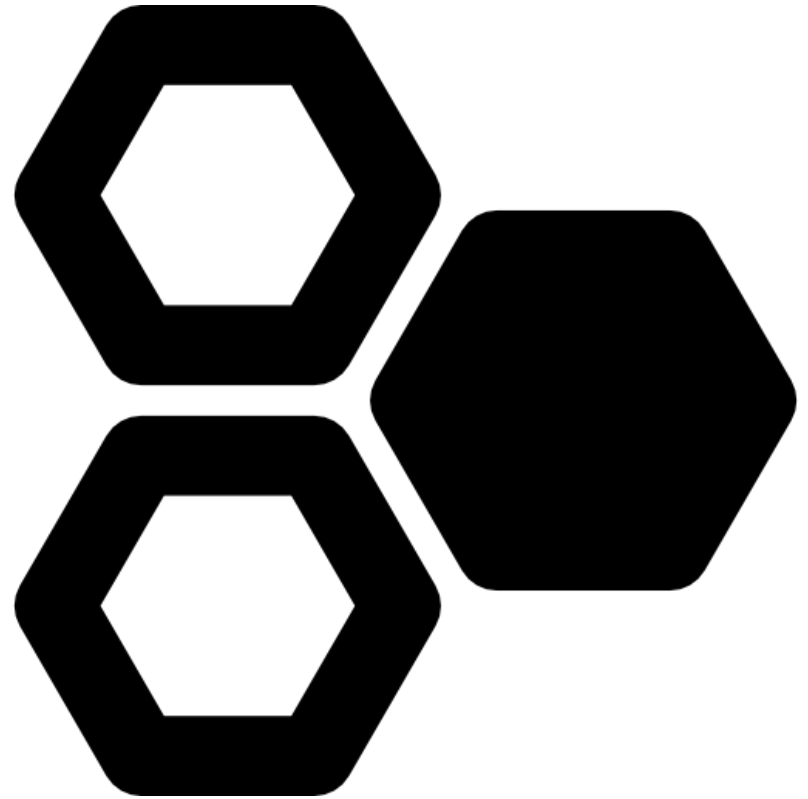


Dr. Abdelkarim Erradi
CSE@QU

Outline

- **Relations between Classes**
- **Introduction to Arrays and Lists**

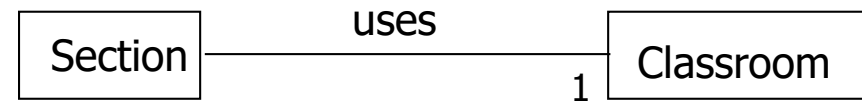
Relations between Classes



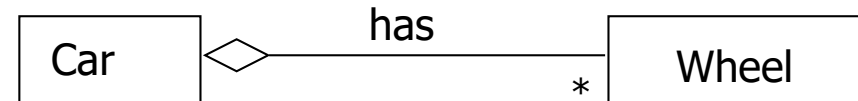
Relations between Classes

- Classes can be related to other classes in 4 ways:

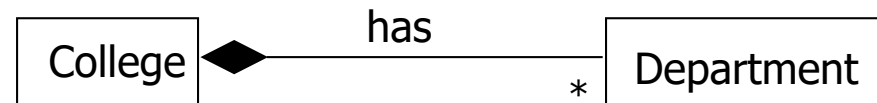
- **Association** (uses without ownership)



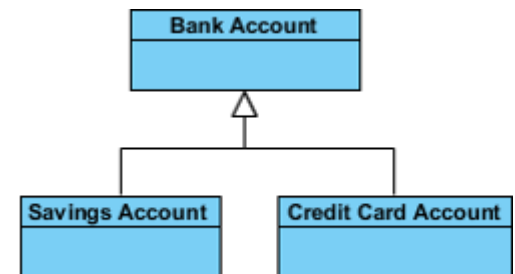
- **Aggregation** (has-a + Whole-Part relationship)



- **Composition** (has-a + Part cannot exist without the Whole)



- **Inheritance** (is-a relation)



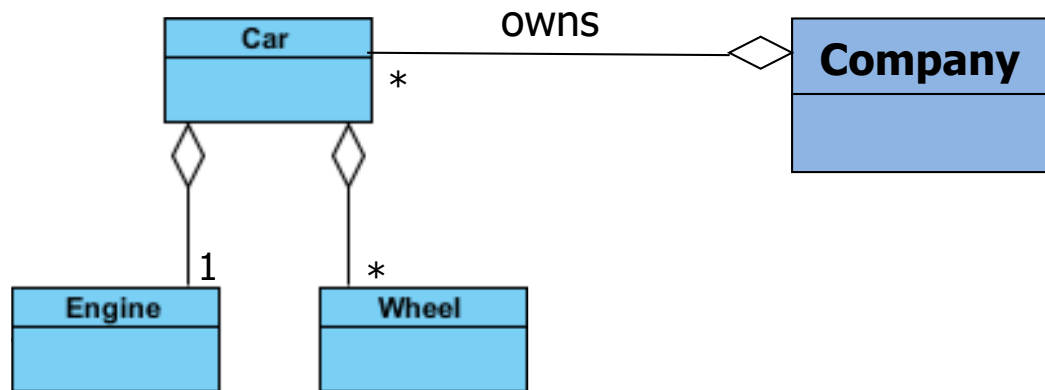
Association

- Association is a very generic relationship used when one class **uses** the functionalities provided by another class
- **No ownership** between the objects and both have their own lifecycle. Both can be created and deleted independently

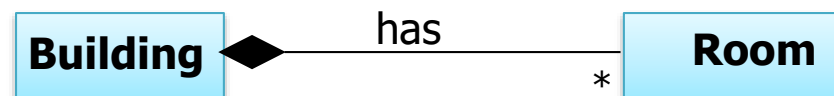
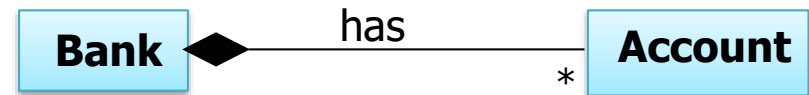
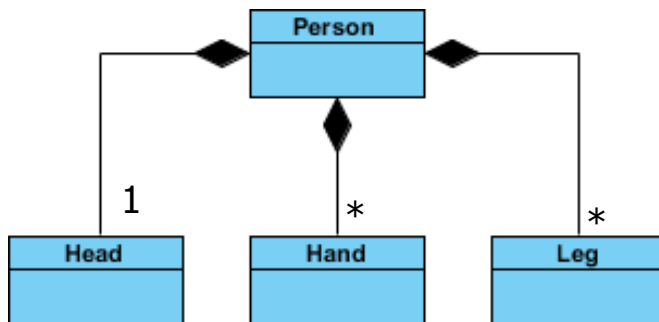


Aggregation vs. Composition

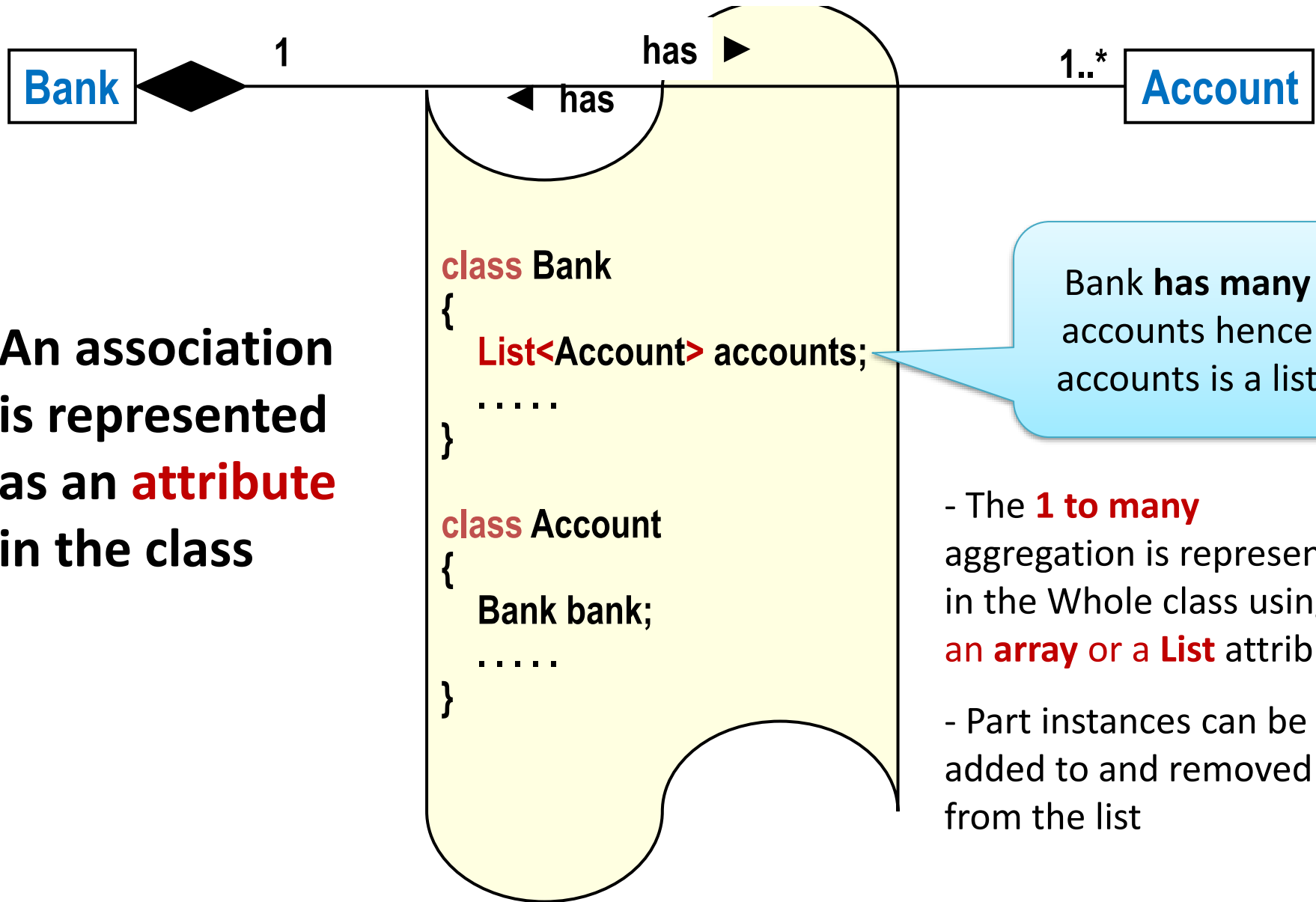
- **Aggregation** = WHOLE-PART relationship. PART can exist without the WHOLE.



- **Composition** = WHOLE-PART relationship. PART cannot meaningfully exist without the WHOLE



Implementation of bidirectional association



```

public class Car {
    private Engine engine;
    private List<Wheel> wheels;

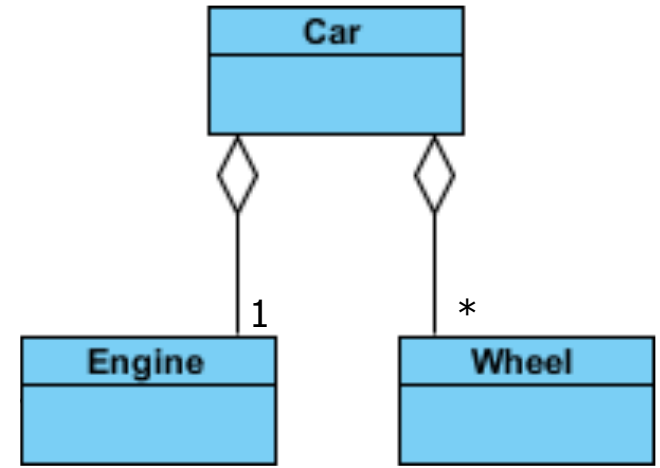
    public Car(Engine engine){
        this.engine = engine;
        this.wheels = new ArrayList<>();
    }

    public addWheel(Wheel wheel){
        wheels.add(wheel);
    }
}

class Engine {
    private String type;
}

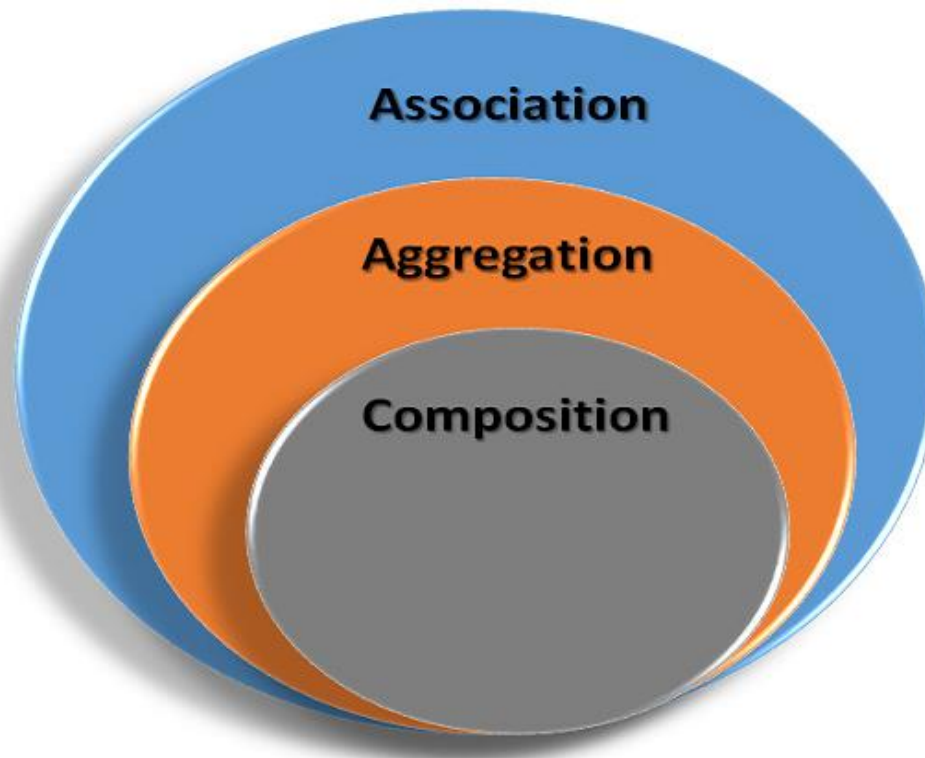
class Wheel {
    private int size;
}

```



Association vs. Aggregation vs. Composition

- A relationship between two classes is referred as an **Association**
- **Aggregation** is a special form of Association
- **Composition** is a strong form of Aggregation



Arrays and Lists



A simple variable stores a single value

MEMORY

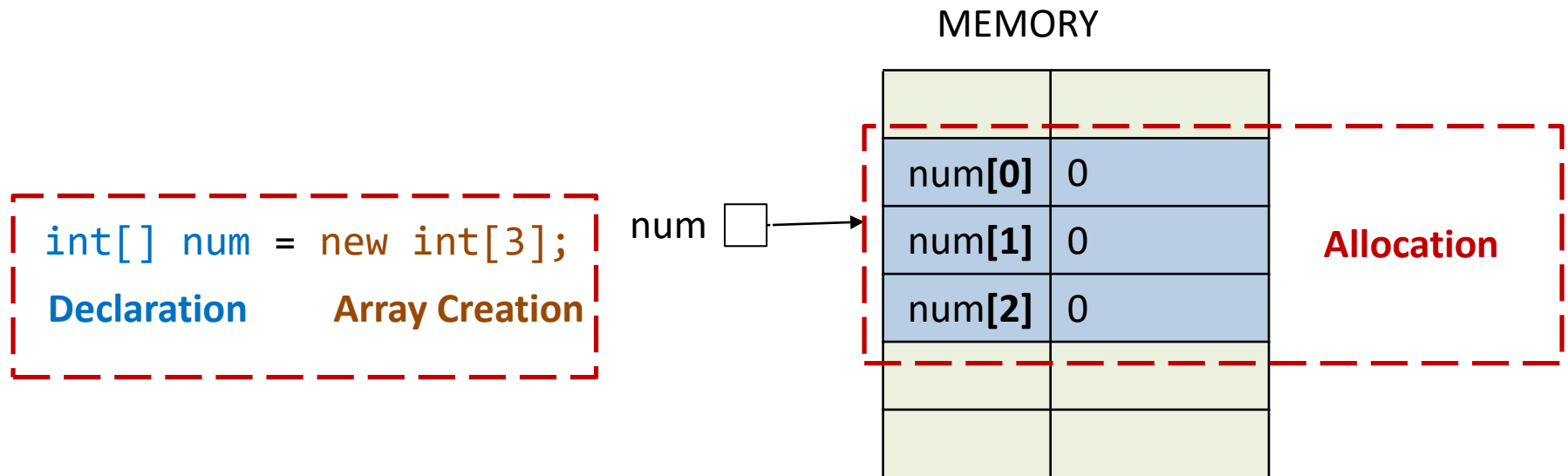
```
int num1 = 10;
```

```
int num2 = 20;
```

```
int num3 = 30;
```

num1	10
num2	20
num3	30

An array object stores multiple values of the same type



- Array elements are **auto initialized** with the type's default value:
 - 0 for the numeric primitive-type elements, false for boolean elements and null for references

Array stores values of the same type

```
int[] number = new int[100];           // stores 100 integers
```

```
double[] salesTax = new double[10];    // stores 10 doubles
```

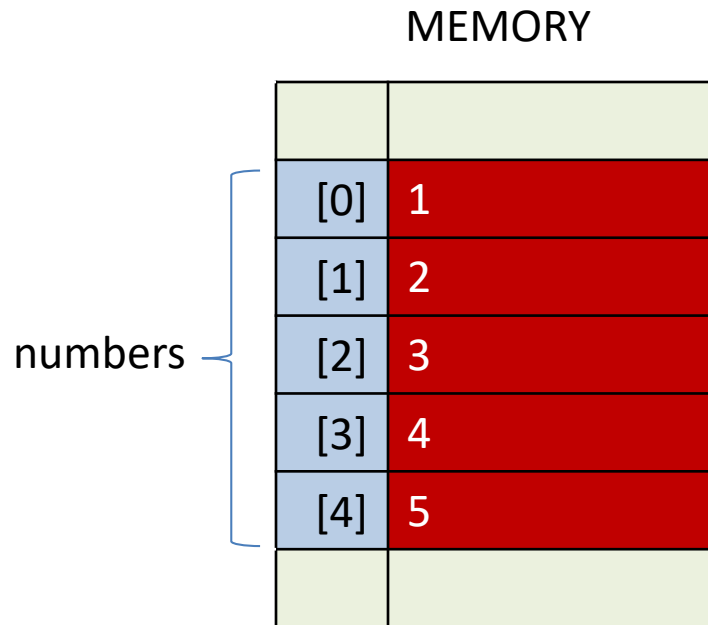
```
char[] alphabet = new char[26];        // stores 26 characters
```

```
Student[] students = new Student[40];  // stores 40 students
```

- The array **size** determines the number of elements in the array.
- The **size** must be specified in the array declaration and it cannot change once the array is created

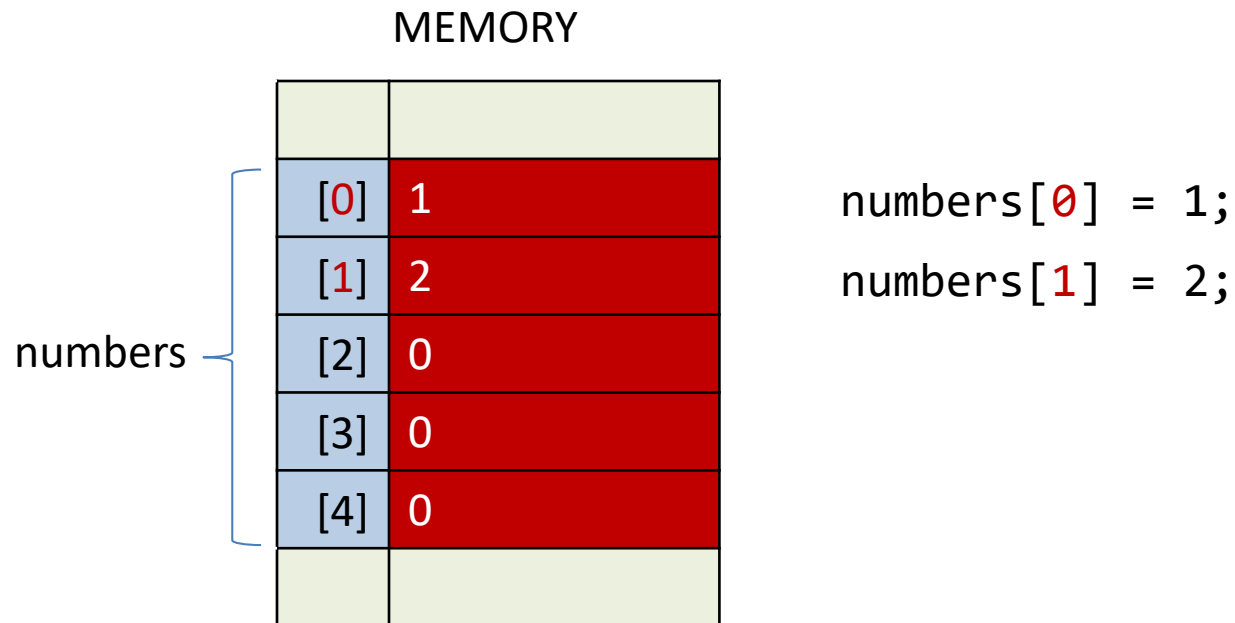
You may initialize an array explicitly

```
int[] numbers = {1, 2, 3, 4, 5}; // Array initializer
```



Array elements are indexed

```
int[] numbers = new int[5];
```



- Array index range is 0 to array size -1

Arrays can be instance variables

```
public class Department {  
    private Employee[] employee;  
    ...  
}
```

Arrays can be local variables

```
public void getHourlyEmployees() {  
    Employee[] hourlyEmployee;  
    ...  
}
```

Arrays can be parameters

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

Arrays can be return values

```
public Employee[] getEmployees() {  
    ...  
}
```


Example - Method that returns an array

```
public int[] initArray(int size, int initValue) {  
    int[] array = new int[size];  
  
    for (int i = 0; i < array.length; i++) {  
        array[i] = initValue;  
    }  
  
    return array;  
}
```

Arrays are objects, thus

```
int[] a = {1, 2, 3};  
int[] b;
```

```
b = a;           // makes b and a refer to the same  
                  // memory location
```

- Arrays are objects so they are **reference types**.
- Elements can be either primitive or reference types.

Arrays are objects, thus

```
int[] a = {1, 2, 3};  
int[] b = {1, 2, 3};
```

```
if (a == b) {...}    // evaluates to false  
                      // since a and b refer to two  
                      // different memory locations
```

Example - Method that tests for array equality

```
public boolean areEqual(int[] array1, int[] array2) {  
    if (array1.length != array2.length) {  
        return false;  
    } else {  
        for(int i = 0; i < array1.length; i++) {  
            if(array1[i] != array2[i])  
                return false;  
        } // end for  
    } // end if  
    return true;  
}
```

Use linear (sequential) search to locate values

MEMORY

[0]	12
[1]	1
[2]	44
[3]	15
[4]	6

an array {

Q: is this the value? A: No

Q: is this the value? A: No

Q: is this the value? A: No

Q: is this the value? A: Yes

Q: is this the value 15 in the array?

Linear Search

```
// Returns true if array contains item, false otherwise.
private boolean contains(String[] items, String item) {
    for(int i = 0; i < items.length; i++) {
        if (items[i].equalsIgnoreCase(item)) {
            return true;
        }
    } // end for
    return false;
}
```

Multidimensional Arrays

- **Two-dimensional arrays** are often used to represent tables of values with data arranged in *rows* and *columns*.
- Example two-dimensional arrays with 3 rows and 4 columns

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Diagram illustrating the indexing of a two-dimensional array. The array is represented as a table with 3 rows and 4 columns. The elements are labeled as `a[row][column]`. The diagram shows the structure of the array and the meaning of the indices:

- Array name: `a`
- Row index: `2`
- Column index: `1`

Multidimensional Arrays (Cont.)

- A multidimensional array **b** with 3 rows and 4 columns

```
int[][] b = new int[3][4];
```

- A two-dimensional array **b** with 2 rows and 3 columns could be declared and initialized with **nested array initializers** as follows:

```
int[][] b = {{1, 2, 9}, {3, 4, 8}};
```

- The initial values are *grouped by row* in braces.
- The number of nested array initializers (represented by sets of braces within the outer braces) determines the number of *rows*.
- The number of initializer values in the nested array initializer for a row determines the number of *columns* in that row.

Lists

- Problem
 - You must know the array size when you create the array
 - Array size cannot change once created.
- Solution:
 - Use **ArrayList**: they stretch as you add elements to them or shrink as you remove elements from them
 - Similar to arrays + **Dynamic resizing**

ArrayList methods

- Create empty list
`new ArrayList<>()`
- Add entry to end
`add(value)` (adds to end)
- Retrieve n^{th} element
`get(index)`
- Check if element exists in list
`contains(element)`
- Remove element
`remove(index)` or `remove(element)`
- Find the number of elements
`size()`
- Remove all elements
`clear()`

ArrayList Example

```
import java.util.*; // Don't forget this import
```

```
public class ListTest2 {  
    public static void main(String[] args) {  
        List<String> entries = new ArrayList<>();  
        double d;  
        while((d = Math.random()) > 0.1) {  
            entries.add("Value: " + d);  
        }  
        for(String entry: entries) {  
            System.out.println(entry);  
        }  
    }  
}
```

This tells Java that
the list will contain
only strings.

Variable-Length Argument Lists

- Variable-length argument lists can be used to create methods that receive an unspecified number of arguments.
 - Parameter type followed by an **ellipsis** (`...`) indicates that the method receives a variable number of arguments of that particular type.
 - The ellipsis can occur only once at the end of a parameter list.

Variable-Length Argument Lists - Example

```
// Variable-Length Argument Lists - Example
public static double average(double... numbers) {
    double total = 0.0;
    for(var num : numbers) {
        total += num;
    }
    return total / numbers.length;
}

public static void main(String[] args) {
    double avg = average(4, 6, 2);
    System.out.println(avg);
}
```

Banking System Example

QuBank 🔍

BankUI
<u>+main(args : String []) : void</u>

**This is the main
class to run the App**

Account
<<Property>> -accountNo : int
<<Property>> -accountName : String
<<Property>> -balance : double
+Account(accountNo : int, accountName : String, balance : double)
+Account(accountNo : int, accountName : String)
+deposit(amount : double) : String
+withdraw(amount : double) : String

Bank has
many
Accounts

Bank
<u>-lastAccountNo : int = 0</u>
<u>-accounts : Account = new ArrayList<>()</u>
<u>+addTestAccounts() : void</u>
<u>+addAccount(account : Account) : void</u>
<u>+getAccount(accountNo : int) : Account</u>
<u>+getBalance(accountNo : int) : double</u>
<u>+deposit(accountNo : int, amount : double) : String</u>
<u>+withdraw(accountNo : int, amount : double) : String</u>
<u>+getFormattedBalance(accountNo : int) : String</u>



Bookstore System example

QuBookstore 🔍

BookStoreUI
<pre>+main(args : String []) : void +addItemsToShoppingCart() : ShoppingCart +displayShoppingCart(shoppingCart : ShoppingCart) : void</pre>

This is the main class to run the App

DataEntryUtils
<pre>+getString(scanner : Scanner, prompt : String) : String +getInt(scanner : Scanner, prompt : String) : int +getInt(scanner : Scanner, prompt : String, min : int, max : int) : int +getDouble(scanner : Scanner, prompt : String) : double +getDouble(scanner : Scanner, prompt : String, min : double, max : double) : double</pre>

This is a utility class to ease data entry

A CartItem is for one particular book

CartItem
<pre><<Property>> -quantity : int <<Property>> -book : Book +CartItem() +CartItem(book : Book, quantity : int) +getTotal() : double +getFormattedTotal() : String</pre>

ShoppingCart
<pre><<Property>> -cartItems : CartItem +ShoppingCart() +addItem(cartItem : CartItem) : void +getTotal() : double +getFormattedTotal() : String</pre>

A Shopping Cart contains 1 or many items

Book
<pre><<Property>> -code : String <<Property>> -description : String <<Property>> -price : double +Book() +Book(code : String, description : String, price : double) +getFormattedPrice() : String</pre>

BookCatalog
<pre>-books : Book = new ArrayList<>() +addTestBooks() : void +getBook(bookCode : String) : Book</pre>

The BookCatlog contains many books