COM2104: Advanced Programming

WK4 LECTURE: AGGREGATION & ARRAYLIST

Objectives

- Know how to use this keyword and how to override equals method.
- Know what is aggregation and how to create class files with aggregated relationship.
- Know how to create array list with different data types and methods about array list
- Know how to sort an array list based on different criteria

this keyword

- The this keyword in Java is a reference variable that refers to the current object.
- It is used within an instance method or a constructor to access members of the current object such as instance variables, methods, and constructors.

Recalling: Overriding toString() method

```java
class Student {
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return id + " " + name;
    }
}
// Driver class to test the Student class
public class Demo {
    public static void main(String[] args) {
        Student s = new Student(101, "James Bond");
        System.out.println("The student details are: "+s);
    }
}
```

Expected output

```
The student details are: 101 James Bond
```

After overriding, we could get expected output about invoking toString() method.

Usage for this keyword

- this keyword is primarily used in the following scenarios:
  - To refer to the current class instance variable.
  - To invoke the current class method.
  - To invoke the current class constructor.

Referencing Instance Variables: Example

```java
public class ThisExample {
    int a;
    int b;

    ThisExample(int a, int b) {
        this.a = a;
        this.b = b;
    }

    void display() {
        System.out.println("a: " + this.a + ", b: " + this.b);
    }

    public static void main(String[] args) {
        ThisExample obj = new ThisExample(10, 20);
        obj.display();
    }
}
```

In this example, the this keyword is used to differentiate between instance variables a and b and the constructor parameters a and b.

Output:

a:10, b:20

Invoking a Method: Example

```java
public class ThisMethodExample {
    void display() {
        System.out.println("Hello, World!");
    }

    void invokeDisplay() {
        this.display();
    }

    public static void main(String[] args) {
        ThisMethodExample obj = new ThisMethodExample();
        obj.invokeDisplay();
    }
}
```

Here, the this keyword is used to invoke the display method from within the invokeDisplay method.

Output:

Hello, World!

Invoking a Constructor: Example

```java
public class ThisConstructorExample {
    int a;
    int b;

    ThisConstructorExample() {
        this(10, 20);
    }

    ThisConstructorExample(int a, int b) {
        this.a = a;
        this.b = b;
    }

    void display() {
        System.out.println("a: " + a + ", b: " + b);
    }

    public static void main(String[] args) {
        ThisConstructorExample obj = new ThisConstructorExample();
        obj.display();
    }
}
```

In this example, the this keyword is used to call another constructor from within a constructor.

Output:

Hello, World!

equals() method

Any class has equals() method by default. Considering the following example:

```
public class LectureE1 {
    private double re, im;
    public LectureE1(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public static void main(String args[]) {
        LectureE1 l1 = new LectureE1(10,15);
        LectureE1 l2 = new LectureE1(10,15);

        if (l1.equals(l2)) {
            System.out.println("Equal");
        }else {
            System.out.println("Not Equal");
        }
    }
}
```

Output:

Not equal

Why l1 and l2 have the same values, they are not same?

Since by default, the equals method will check whether both l1 and l2 refer to same object or not (object variables are always references in Java). l1 and l2 refer to two different objects, hence the value (l1.equals(l2)) is false. Overriding equals method could fix this issue.

Overriding equals() method

```
public class LectureE1 {
    private double re, im;
    public LectureE1(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public boolean equals(LectureE1 o1) {
        boolean check = true;
        if (this.re == o1.re & this.im == o1.im) {
            check = true;
        }else {
            check = false;
        }
        return check;
    }
    public static void main(String args[]) {
        LectureE1 l1 = new LectureE1(10,15);
        LectureE1 l2 = new LectureE1(10,15);

        if (l1.equals(l2)) {
            System.out.println("Equal");
        }else {
            System.out.println("Not Equal");
        }
    }
}
```

Output:

Equal.

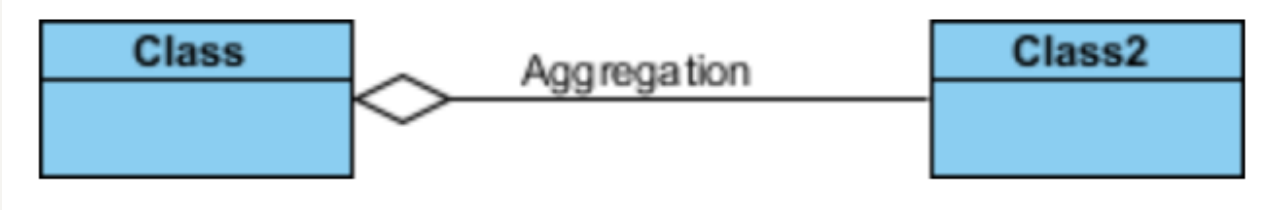
AGGREGATION

What is aggregation?

- An aggregation is a relationship between two classes where one class contains an instance of another class.
- For example, when an object A contains a reference to another object B or we can say Object A has a HAS-A relationship with Object B, then it is termed as Aggregation in Java Programming.

UML diagram for aggregation

Class2 has one instance of Class. We use the rhombus to represent aggregation relationship.



UML diagram for aggregation: Example

Student
-studentName: String -studentID: int
+Student(studentName: String, studentID: int) //constructor +getstudentID():int +getstudentName(): String

Department class contain one attribute which is the instance of Student class.

Department
-deptName: String -students: Student
+Department(deptName: String, students: Student) //constructor +getStudents(): Student +main()

Code for the above example

```java
public class Student {
    // Attributes of Student
    private String studentName;
    private int studentId;

    // Constructor of Student class
    public Student(String studentName, int studentId)
    {
        this.studentName = studentName;
        this.studentId = studentId;
    }

    public int getstudentId() {
      return studentId;
    }

    public String getstudentName() {
      return studentName;
    }
}
```

```java
public class Department {
    // Attributes of Department class
    private String deptName;
    private Student students;

    // Constructor of Department class
    public Department(String deptName, Student students)
    {
        this.deptName = deptName;
        this.students = students;
    }

    public Student getStudents() {
        return students;
    }

    public static void main(String args[]) {
        Student s = new Student("Wen", 111);
        Department dd = new Department("COM", s);
        Student d1= dd.getStudents();
        d1.getstudentId();
        System.out.printf("The name for student is %s. The ID is %d\n",
                d1.getstudentName(), d1.getstudentId());
    }

}
```

```
The name for student is Wen. The ID is 111
```

ARRAYLIST

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.

One example about Arraylist

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

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

Java ArrayList Methods

Methods	Description
add(Object o)	This method is used to append a specific element to the end of a list.
set(index i, object o)	This method is used to replace the element with index of i to o.
get(index i)	This method is used to get the element with index of i.
remove(index i)	This method is used to remove the element with index of i.
size()	This method is used to get the number of elements in the arraylist.
indexof(element)	This method is used to get the index of one specific element in the arraylist.
clear()	This method is used to remove all the elements from any list.

Use 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);
  }
}
```

Use add() method: Cont.

You can also add an item at a specified position by referring 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(0, "Mazda"); // Insert element at the beginning of the list (0)

    System.out.println(cars);
  }
}
```

Use other methods

Access the element with index of 0.

```
cars.get(0);
```

Replace the element with index of 0 to "Opel".

```
cars.set(0, "Opel");
```

Remove the element with index of 0.

```
cars.remove(0);
```

Get the size of the arraylist.

```
cars.size();
```

Remove all elements in the arraylist.

```
cars.clear();
```

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));
    }
  }
}
```

Output

```
Volvo
BMW
Ford
Mazda
```

Loop Through an ArrayList: Cont.

You can also loop through an ArrayList with the for-each loop:

```
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);
    }
  }
}
```

Output

```
Volvo
BMW
Ford
Mazda
```

Other Types of Arraylist

- 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.

An ArrayList to store numbers (Integer)

```java
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);
    }
  }
}
```

Output

```
10
15
20
25
```

An ArrayList to store characters (char)

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Character> myChars = new ArrayList<Character>
();
    myChars.add('a');
    myChars.add('b');
    myChars.add('c');
    myChars.add('d');
    for (char i:myChars) {
      System.out.println(i);
    }
  }
}
```

Output

```
a
b
c
d
```

An ArrayList to store boolean values (Boolean)

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Boolean> myChars = new ArrayList<Boolean>();
    myChars.add(true);
    myChars.add(true);
    myChars.add(false);
    myChars.add(true);
    for (boolean i:myChars) {
      System.out.println(i);
    }
  }
}
```

Output

```
true
true
false
true
```

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.
- Usage:

```
import java.util.Collections;

Collections.sort(an arraylist).
```

- By default, we will sort an arraylist in ascending order.

Sort an ArrayList of Strings

```java
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);
    }
  }
}
```

Output

```
BMW
Ford
Mazda
Volvo
```

This sort method will sort all strings based on their first letter

Sort an ArrayList of Integers

```java
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);
    }
  }
}
```

Output

```
8
12
15
20
33
34
```

Sort all integers in ascending order.

Sort an arraylist in descending order

Syntax:

```
Collections.sort(ArrayList, Collections.reverseOrder());
```

Sort an arraylist of strings in descending order

```java
import java.util.*;

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

        // Get the ArrayList
        ArrayList<String>
            list = new ArrayList<String>();

        // Populate the ArrayList
        list.add("Geeks");
        list.add("For");
        list.add("ForGeeks");
        list.add("GeeksForGeeks");
        list.add("A computer portal");

        // Print the unsorted ArrayList
        System.out.println("Unsorted ArrayList: "
                           + list);

        // Sorting ArrayList in descending Order
        // using Collection.sort() method
        // by passing Collections.reverseOrder() as comparator
        Collections.sort(list, Collections.reverseOrder());

        // Print the sorted ArrayList
        System.out.println("Sorted ArrayList "
                           + "in Descending order : "
                           + list);
    }
}
```

Output:

```
Unsorted ArrayList: [Geeks, For, ForGeeks, GeeksForGeeks, A computer portal]
Sorted ArrayList in Descending order : [GeeksForGeeks, Geeks, ForGeeks, For,
A computer portal]
```

Sort an arraylist of integers in descending order

```java
import java.util.ArrayList;
import java.util.Collections;

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,Collections.reverseOrder());

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

Output

```
34
33
20
15
12
8
```

Get sum and average for an arraylist of integers

```java
// Java program to calculate sum and average of elements in an ArrayList
import java.io.*;
import java.util.ArrayList;

class Main {
    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(45);
        list.add(54);

        // Calculate the sum of elements
        int sum = 0;
        for (int i = 0; i < list.size(); i++)
        {
            sum += list.get(i);
        }
        System.out.println("Sum: " + sum);

        // Calculate the average of elements
        double average = (double)sum / list.size();
        System.out.println("Average: " + average);
    }
}
```

Output

```
Sum: 159
Average: 31.8
```

An ArrayList with class objects as elements

```java
import java.util.ArrayList;
//Class user-defined
class Person {

 // Random properties associated with the person
 // Person name
 String name;
 // Person age
 int age;

 // Constructor for class Person
 // for initializing objects
 Person(String name, int age)
    {
     // This keyword for refering to current object
     this.name = name;
     this.age = age;
    }
}

```

```java
public class LectureE2 {
    // Main driver method
    public static void main(String[] args)
    {
        // Make Person data-type objects
        Person p1 = new Person("Aditya", 19);
        Person p2 = new Person("Shivam", 19);
        Person p3 = new Person("Anuj", 15);
        // Create an ArrayList object
        //(Declaring List of Person type)
        ArrayList<Person> names = new ArrayList<Person>();
        // Adding objects to the ArrayList
        names.add(p1);
        names.add(p2);
        names.add(p3);
        // Print and display the elements of adobe ArrayList
        // using get() method
        System.out.println(names.get(0).name);
        System.out.println(names.get(0).age);
        System.out.println(names.get(1).name);
        System.out.println(names.get(1).age);
        System.out.println(names.get(2).name);
        System.out.println(names.get(2).age);
        // Optional Part for better understanding
        System.out.println("Optional Part Added For Better Understanding");
        // (Optional)
        // Displaying what happens if printed by simply
        // passing List object as parameter
        System.out.println(names);}
}
```

Output:

```
Aditya
19
Shivam
19
Anuj
15
Optional Part Added For Better Understanding
[LabFour.Person@3f8f9dd6, LabFour.Person@aec6354, LabFour.Person@1c655221]
```

3/5/2025

End