Sukkur Institute of Business Administration University

Department of Computer Science

Object Oriented Programming using Java

BS – II (CS/AI/SE)

Spring-2024

Lab # 06: Let's learn about class association and Inheritance

Instructor: Nimra Mughal

Objectives

After performing this lab, students will be able to understand:

- Association in java
- Inheritance in Java

Association in JAVA

https://www.geeksforgeeks.org/association-composition-aggregation-java/

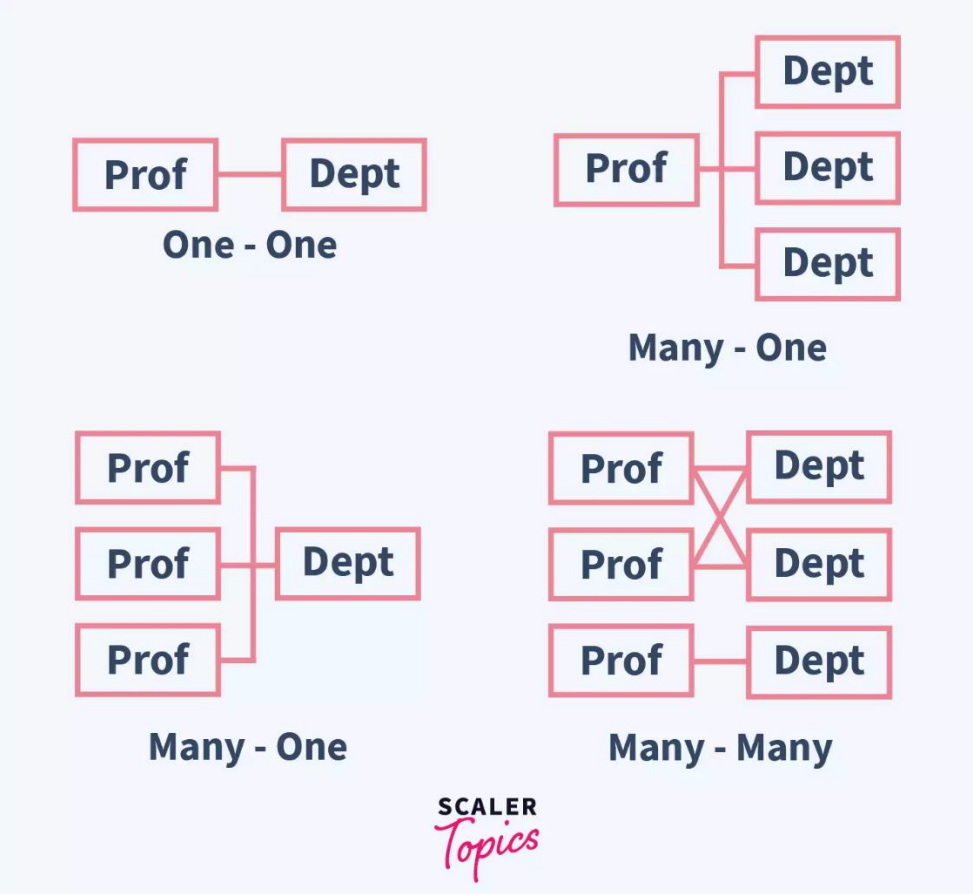
https://medium.com/edureka/association-in-java-3d0facf63d56

https://www.javatpoint.com/association-in-java

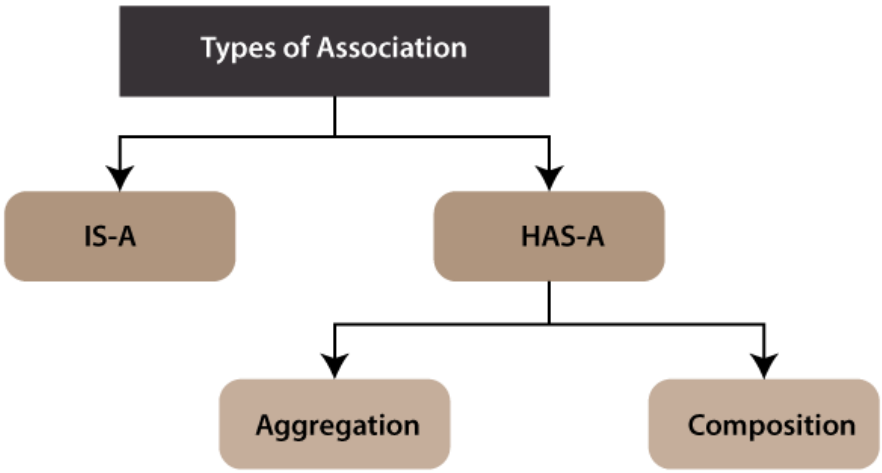
https://www.scaler.com/topics/association-composition-and-aggregation-in-java/

Association, in general terms, refers to the relationship between any two entities. Association in java is the relationship that can be established between any two classes. These relationships can be of four types:

1. One-to-One relation
2. One-to-many relation
3. Many-to-one relation
4. Many-to-many relation



Forms/Types of Association



IS-A Association

This association is also referred to as **Inheritance.**

HAS-A Association

Classified into two parts, i.e**., Aggregation and Composition**. Let's understand the difference between both of them one by one.

Composition

It is a "belongs-to" type of association. It simply means, it is a part or member of the larger object. Alternatively, it is often called a "has-a" relationship.

For example, a building has a room, or in other words, a room belongs to a building. Composition is a strong kind of "has-a" relationship because the objects' lifecycles are tied. It means that if we destroy the owner object, its members also will be destroyed with it.

```
//Car must have Engine
public class Car {
   //engine is a mandatory part of the car
   private final Engine engine;

   public Car () {
     engine = new Engine();

   }
}
//Engine Object
class Engine {}
```

Aggregation

Aggregation is also a "has-a" relationship, but, what distinguishes it from composition, is that the lifecycles of the objects are not tied. Both the entries can survive individually which means ending one entity will not affect the other entity. Both of them can exist independently of each other. Therefore, it is often referred to as week association.

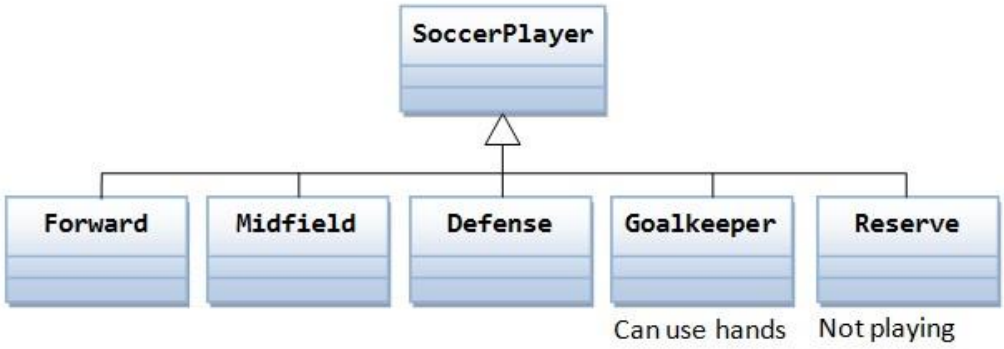
For example: A player who is a part of the team can exist even when the team ceases to exist. The main reason why you need Aggregation is to maintain code reusability.

```
//Team
public class Team {
   //players can be 0 or more
   private int players[];

   public Team () {
      players = new int[10];
   }
Player p=new Player();
}
//Player Object
class Player {}
```

Please visit Elearning for Practice codes……..

Inheritance



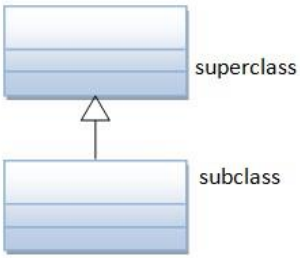

```
class Goalkeeper extends SoccerPlayer {......}

class MyApplet extends java.applet.Applet {.....}

class Cylinder extends Circle {......}
```

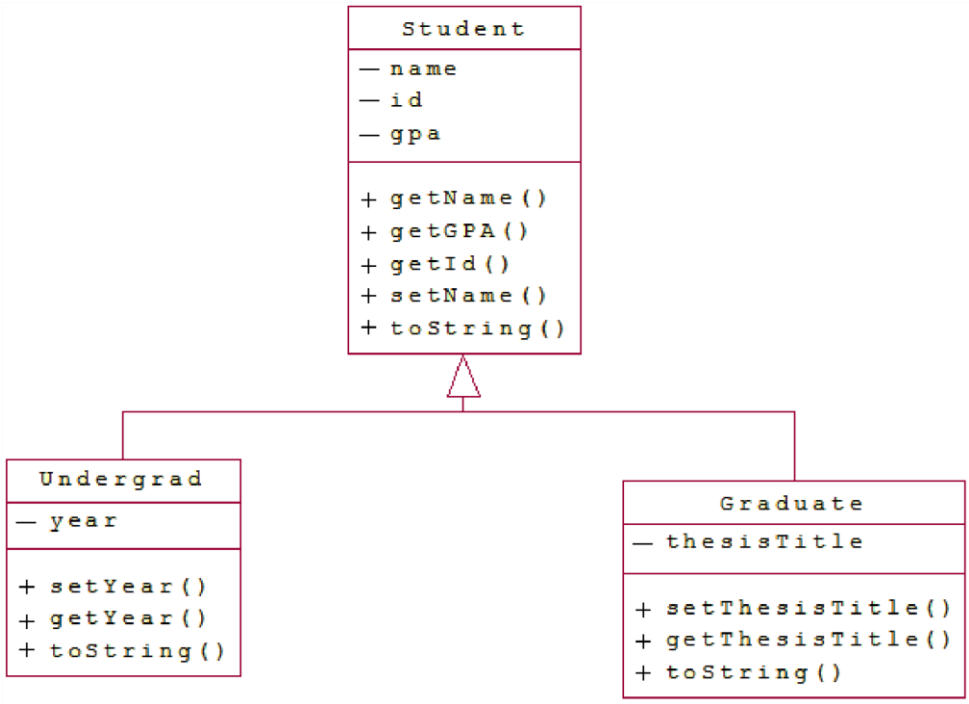
UML Notation:

The UML notation for inheritance is a solid line with a hollow arrowhead leading from the subclass to its superclass. By convention, superclass is drawn on top of its



subclasses as shown.

Example of Inheritance:



The class Student is the parent class. Note that all the variables are private and hence the child classes can only use them through accessor and mutator methods. Also note the use of **overloaded** constructors.

The class Student is the parent class. Note that all the variables are private and hence the child classes can only use them through accessor and mutator methods. Also note the use of **overloaded** constructors.

```
public class Student{
private String name;  private int id;  private double gpa;
      public Student(int id, String name, double gpa) {
             this.id = id;
             this.name = name;
             this.gpa = gpa;
      }

      public Student(int id, double gpa){
             this(id, "", gpa);
      }
      public String getName(){return name;}
      public int getId() {return id;}
      public double getGPA(){return gpa;}
      public void setName(String newName){
             this.name = newName;
      }
      public String toString(){
             return "Student:\nID: "+id+"\nName: "+name+"\nGPA: "+gpa;
      }

 }// Student class ends
```

The class Undergrad extends the Student class. Note the overridden toString() method

```
public class Undergrad extends Student {
       private String year;

       public Undergrad(int id, String name, doubl gpa, String year)  {
              super(id, name, gpa);  // super() can be used to invoke immediate
parent class constructor.
              this.year = year;
       }
       public String getYear() {return year;}
```

```
	public void setYear(String newYear) {this.year = newYear;}


	public String toString() {
		return "Undergraduate "+super.toString()+"\nYear: "+year;  } }
//Undergrad class ends
```

The class Graduate extends the Student class too. Note the overridden toString() method

```
public class Graduate extends Student {
	private String thesisTitle;
	public  Graduate(int  id,  String  name,  double  gpa,  String
thesisTitle)  {
		super(id, name, gpa);
		this.thesisTitle = thesisTitle;
	}
	public String getthesisTitle() {  return thesisTitle;  }
	public void setThesisTitle(String newthesisTitle) {
		this.thesisTitle = newthesisTitle;
	}
	public String toString() {
		return "Graduate " +super.toString()+"\nThesis: "+thesisTitle;
} } // Graduate class ends
```

TestStudents is a driver class to test the above classes

```
public class TestStudents {
	public static void main(String[] args)  {
		Student s1 = new Student(123456, "Aariz", 3.27);
		Student s2 = new Student(234567, 3.22);
		Undergrad u1 = new Undergrad(345678, "Asad", 2.73, "Junior");
		Graduate g1 = new Graduate(456789, "Ahmed", 3.67,  "Algorithms
and Complexity");
		System.out.println(s1);
		System.out.println(s2);
		System.out.println(u1);
		System.out.println(g1);
	 }
}// TestStudents class ends
```

The super keyword

The super keyword is like this keyword. Following are the scenarios where the super keyword is used.

It is used to differentiate the members of superclass from the members of subclass, if they have same names.

It is used to invoke the superclass constructor from subclass.

```
super.variable
super.method();
```

Exercises

Mega Exercise: Solve Exercises from the HackerRank (OOP track topics). As many as you can. I have listed here two basic challenges)

Note: you are supposed to complete two tasks of Inheritance during class

https://www.hackerrank.com/domains/java

Exercise 1(a) (Association & Aggregation) *(Time.java, Passenger.java, Flight.java)*

Based On the class diagram given below, you are required to write complete program for Flight's class, Time's class and Passenger's class with the concept of association and aggregation. Functions information also been given in the table below. Program should display information supplied to flight object.

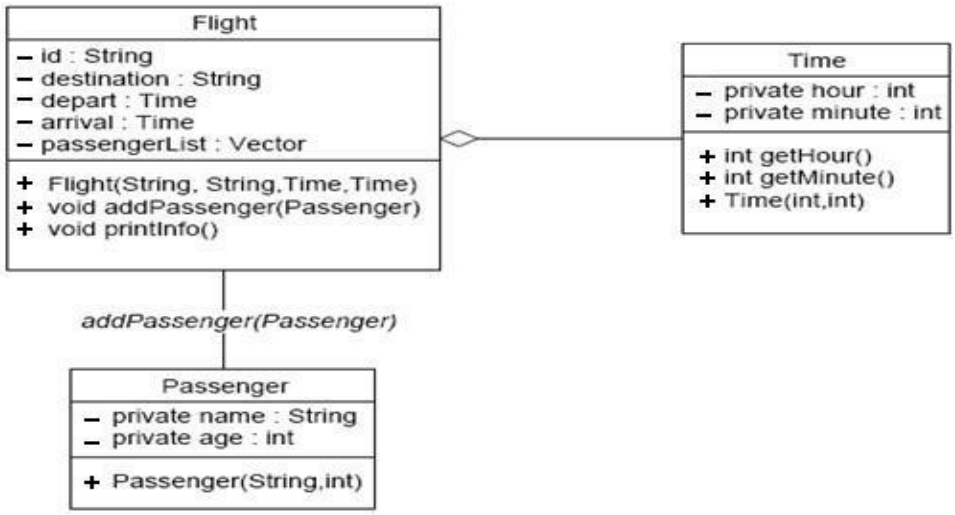


Table of Methods for Flight Class

Method	**Description**
addPassenger(Passenger)	This method will add Passenger's object to vector passengerList.
printInfo()	This method will display all flight information namely ID (Flight number), destination, departure time, arrival time and number of passengers. For Example: Flight no : PK-203 Destination : Sukkur Departure : 8:10 Arrival : 9:00 Number of passenger :3
getHour()	This method will return the value of attribute **hour**
getMinute()	This method will return the value of attribute **minute**

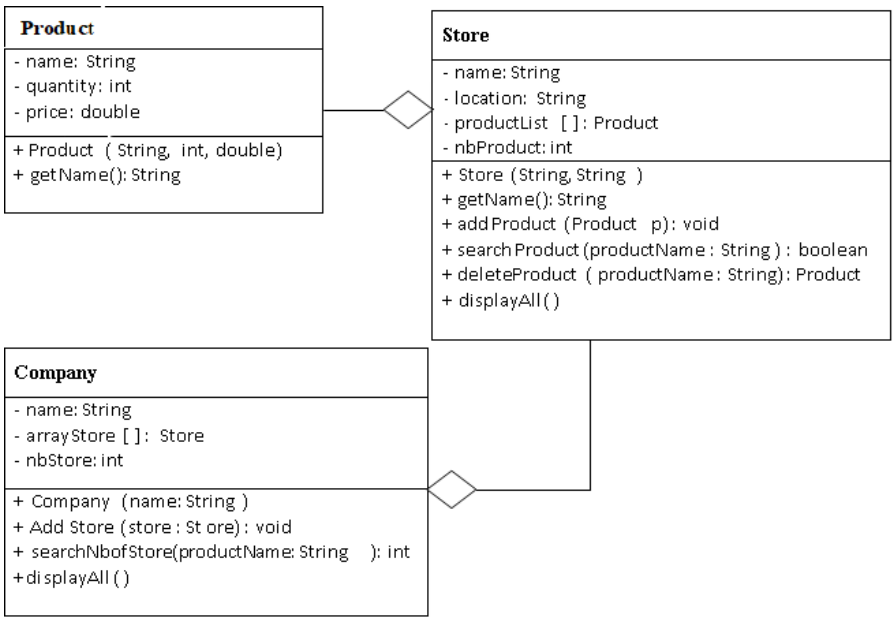
Exercise 1(b) *(FlightTester.java)*

Use the above classes to create the objects in the FlightTester class and call methods:

```
/*
 * FlightTester class
 */
public class FlightTester {
    public static void main(String arg[])
       {
       Time dep=new Time (8,10);
       Time arr=new Time (9,00);
       Flight f = new Flight("PK-303","Lahore",dep,arr);
       Passenger psg1= new Passenger("Aariz", 30);
       Passenger psg2= new Passenger("Arsham", 44);   f.addPassenger(psg1);
       f.addPassenger(psg2);
       f.printInfo();
       }
}
```

Exercise2(a) (Ass: & Agg.) (Product.java, Store.java, Company.java)

A company manages many stores. Each Store contains many Products. The UML diagram of this company system is represented as follow.



Class Store:

Attribute: name, location, productList, nbProduct

Constructor: Store (name: String, location: String):

Method:

addProduct() that adds a new product. Maximum 100 products can be added. searchProduct() that accepts the name of product and return **True** if exist, **False** otherwise. deleteProduct() that accepts the name of product that has to be deleted and returns the deleted object.

displayAll() prints the name of products available in store.

Class Company:

Attribute: name, arrayStore, nbStore

Constructor: Company (name: string):

Method:

addStore() that adds a new Store. Maximum 10 stores can be added.

searchNbofStore() that accepts the name of product and returns the number

of stores containing the product. displayAll() prints the name of stores belongs to company.

Exercise2(b) (TestCompany.java)

Implement Product, Store and Company classes and use the following class to test.

```
public class TestCompany {
    public static void main(String [] args){
        Product p1 = new Product("TV",4,34000);
        Product p2 = new Product("Bicycle", 4, 5500);
        Product p3 = new Product("Oven", 3,70000);

        Store s1 = new Store("Makro", "Karachi");
        Store s2 = new Store("Hypermart","Karachi");
        s1.addProduct(p1);
        s1.addProduct(p2);
        s1.addProduct(p3);
        s1.displayAll();
        Product tempProduct = s1.deleteProduct("Bicycle");
        if (tempProduct!=null)
            System.out.println("Product    "+tempProduct.getName()+    "    is
deleted");
        else
            System.out.println("There is no product to delete");
         s1.displayAll();
         s2.addProduct(p1);
         s2.addProduct(p2);
         s2.addProduct(p3);
         s2.displayAll();
         Company c1 = new Company("Unilever");
         c1.addStore(s1);
         c1.addStore(s2);
         c1.displayAll();
         int n= c1.searchNbofStore("TV");
         System.out.println("Number of stores have TV "+n);

    }
}
```

Exercise 3 (Inheritance)

Define a class named **Person** that contains two instance variables of type String that stores the first name and last name of a person and appropriate accessor and mutator methods.

Also create a method named **displayDetails** that outputs the details of a person.

Next, define a class named **Student** that is derived from **Person**, the **constructor** for which should receive first name and last name from the class Student and also assigns values to student id, course, and teacher name. This class should **redefine** the displayDetails method to person details as well as details of a student. Include appropriate **constructor**(s).

Define a class named **Teacher** that is **derived** from Person. This class should contain instance variables for the subject name and salary. Include appropriate constructor(s). Finally, **redefine** the displayDetails method to include all teacher information in the printout.

Create **a main method** that creates at least two student objects and two teacher objects with different values and calls displayDetails for each.

Exercise4 (Inheritance) Alien.java

The following is some code designed by J. Hacker for a video game. There is an Alien class to represent a monster and an AlienPack class that represents a band of aliens and how much damage they can inflict:

```
class Alien
{
    public static final int SNAKE_ALIEN = 0;
    public static final int OGRE_ALIEN = 1;
    public static final int MARSHMALLOW_MAN_ALIEN = 2;

    public int type; // Stores one of the three above types
    public int health; // 0=dead, 100=full strength
    public String name;
```

```
        public Alien (int type, int health, String name)
        {
            this.type = type;
            this.health = health;
            this.name = name;
        }
    }
    public class AlienPack
    {
        private Alien[] aliens;

        public AlienPack (int numAliens)
        {
            aliens = new Alien[numAliens];
        }
        public void addAlien(Alien newAlien, int index)
        {
            aliens[index] = newAlien;
        }
        public Alien[] getAliens()
        {
            return aliens;
        }
    }
    public int calculateDamage()
    {
        int damage = 0;
        for (int i=0; i < aliens.length; i++)
        {
            if (aliens[i].type==Alien.SNAKE_ALIEN)
            {
                damage +=10;// Snake does 10 damage
            }
            else if (aliens[i].type==Alien.OGRE_ALIEN)
            {
                damage +=6;// Ogre does 6 damage
            }
            else if (aliens[i].type==
            Alien.MARSHMALLOW_MAN_ALIEN)
            {
                damage +=1;
            // Marshmallow Man does 1 damage
            }
        }
        return damage;
    }
}
```

The code is not very object oriented and does not support information hiding in the Alien class. Rewrite the code so that inheritance is used to represent the different types of aliens instead of the "type" parameter. This should result in deletion of the "type" parameter. Also rewrite the **Alien** class to hide the instance variables and create a **getDamage** method for each derived class that returns the amount of damage the alien inflicts. Finally, rewrite the **calculateDamage** method to use **getDamage** and write a main method that tests the code.