Lab 07
Association & Inheritance in JAVA

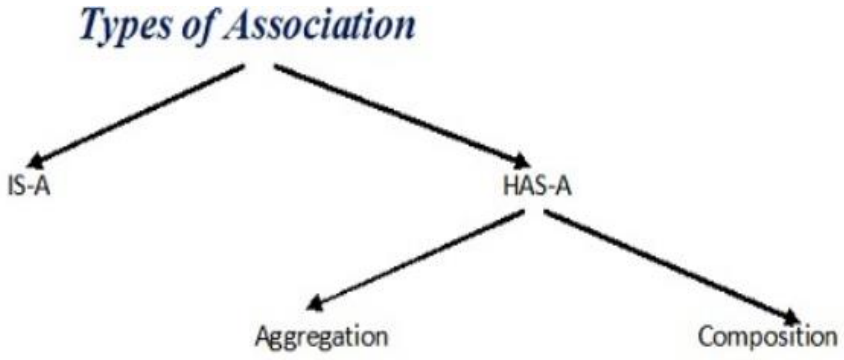
Objective(s):

1. Association in JAVA
2. Inheritance in JAVA
3. Super keyword

1: Association in JAVA

Association establishes relationship between two separate classes through their objects. Association relationship indicates how objects know each other and how they are using each other's functionality. The relationship can be one to one, One to many, many to one and many to many.

In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. **Composition** and **Aggregation** are the two forms of association.



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.

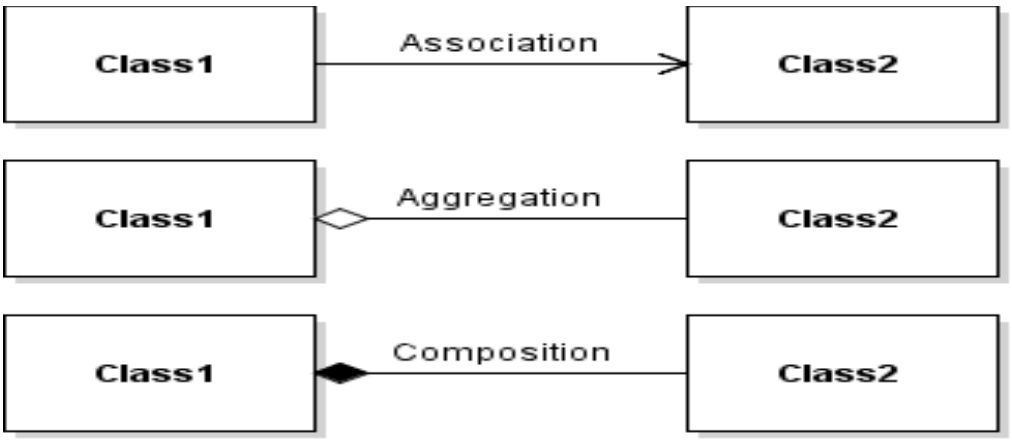
```
//Car must have Engine
public class Car {
   //engine is a mandatory part of the car
   private final Engine engine;

   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 {}
```




```
// Java program to illustrate the concept of Association
import java.io.*;
class Bank
{
	private String name;

	Bank(String name)
	{
		this.name = name;
	}

	public String getBankName()
	{
		return this.name;
	}
}
// employee class
class Employee
{
	private String name;
	// employee name
	Employee(String name)
	{
		this.name = name;
	}

	public String getEmployeeName()
	{
		return this.name;
	}
}
// Association between both the classes in main method
class Association
{
```

```
	public static void main (String[] args)
	{
		Bank bank = new Bank("Axis");
		Employee emp = new Employee("Neha");

		System.out.println(emp.getEmployeeName() +
			" is employee of " + bank.getBankName());
	}
}
```

In above example two separate classes Bank and Employee are associated through their Objects. Bank can have many employees, So it is a one-to-many relationship.

Summary

- Association follows many-to-many relationships.
- Aggregation follows a one-to-one relationship.
- The Composition follows a one-to-many relationship.

2: Inheritance in JAVA

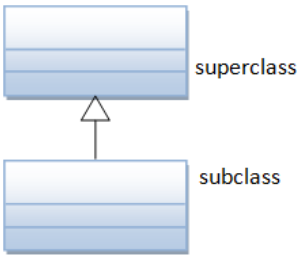
Inheritance is an important object-oriented concept that allows classes to be reused in order to define similar, but distinct, classes. In OOP, classes are organized in *hierarchy* to *avoid duplication and reduce redundancy.*

The classes in the lower hierarchy inherit all the members (variables and methods) from the higher hierarchies, it cannot access those members of the higher hierarchies that have been declared as private. A class in the lower hierarchy is called a *subclass* (or *derived*, *child*, *extended class*). A class in the upper hierarchy is called a *superclass* (or *base*, *parent class*). By pulling out all the common variables and methods into the superclasses, and leave the specialized variables and methods in the subclasses, *redundancy* can be greatly reduced or eliminated as these common variables and methods do not need to be repeated in all the subclasses. ***extends*** *is the keyword used to inherit the properties of a class.*

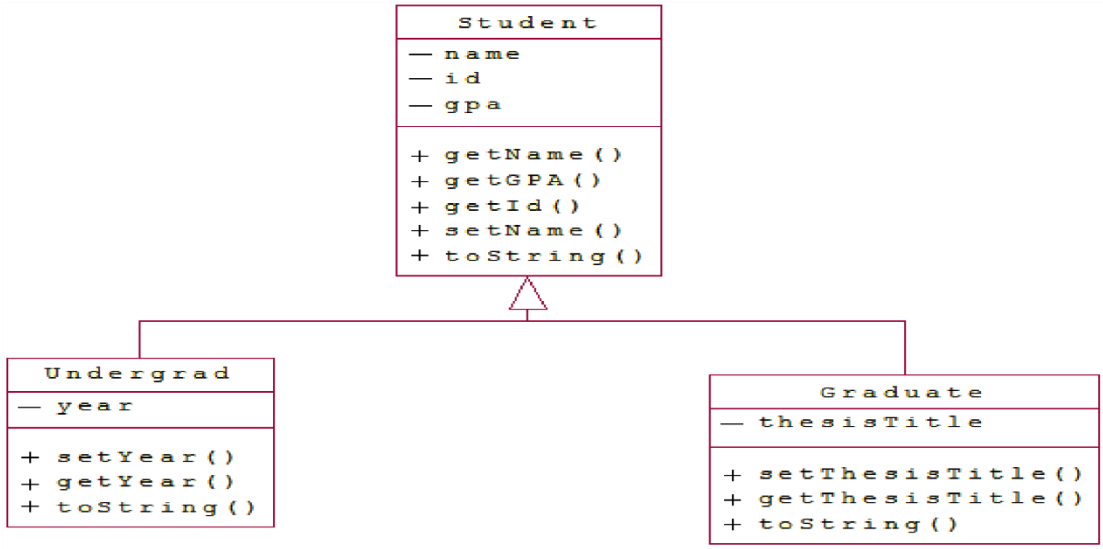
For example:

A subclass inherits all the variables and methods from its superclasses, including its immediate parent as well as all the ancestors. It is important to note that a subclass is not a "subset" of a superclass. In contrast, subclass is a "superset" of a superclass. It is because a subclass inherits all the variables and methods of the superclass; in addition, it extends the superclass by providing more variables and methods.

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:



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();
```

Syntax for using *Super* keyword

```
class Super_class {
   int num = 20;

   // display method of superclass
   public void display() {
      System.out.println("This is the display method of superclass");
   }
}
```

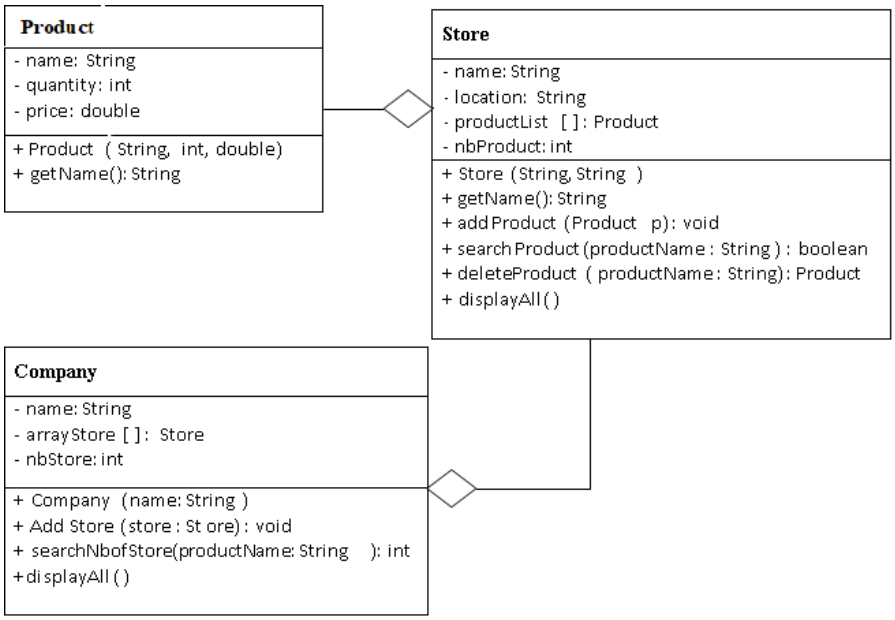
```
public class Sub_class extends Super_class {
   int num = 10;

   // display method of sub class
   public void display() {
      System.out.println("This is the display method of subclass");
   }

   public void my_method() {
      // Instantiating subclass
      Sub_class sub = new Sub_class();

      // Invoking the display() method of sub class
      sub.display();

      // Invoking the display() method of superclass
      super.display();

      // printing the value of variable num of subclass
      System.out.println("value of the variable named num in sub class:"+
sub.num);

      // printing the value of variable num of superclass
      System.out.println("value of the variable named num in super class:"+
super.num);
   }

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

Lab Tasks

Marks : 10, All Questions carry equal marks

Exercise 1 (Ass: & Aggregation) (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.

Exercise 2 *(TestCompany.java)*

```
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);

        }
}
```

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

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.

Exercise 4 (Inheritance)

Define a class named Message that contains an instance variable of type String named text that stores any textual content for the Message. Create a method named toString that returns the text field and also include a method to set this value.

Next, define a class for SMS that is derived from Message and includes instance variables for the recipientContactNo. Implement appropriate accessor and mutator methods. The body of the SMS message should be stored in the inherited variable text. Redefine the toString method to concatenate all text fields.

Similarly, define a class for Email that is derived from Message and includes an instance variable for the sender, receiver, and subject. The textual contents of the file should be stored in the inherited variable text. Redefine the toString method to concatenate all text fields.

Create sample objects of type Email and SMS in your main method. Test your objects bypassing them to the following subroutine that returns true if the object contains the specified keyword in the text property.

```
public static boolean ContainsKeyword(Message messageObject,
String keyword) {
        if (messageObject.toString().indexOf(keyword,0) >= 0)
         return true;
        return false; }
```

Finally, include a method to encode the final message "This is Java" using an encoding scheme, according to which, each character should be replaced by the character that comes after it. For example, if the message contains character B or b, it should be replaced by C or c accordingly, while Z or z should be replaced with an A or a. If the final message is "This is Java", then the encoded message should be "UijtjtKbwb".

Exercise 5
(Inheritance)

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.

Post lab questions to ponder

1. Can class extend more than one classes?
2. Can you restrict a class from inheriting another class?
3. Describe usage of toString() method.

END