Lab 01: single class

Exercise 1

Write a Java code to:

1. Represent a Date class that composed of three attributes: name, month, year and day; so the class contains three Data Members.

2. Create a DisplayMethod to display your name and date as day/ month / year

3. Test the Date class. By creating a date object that contains your date of birth then print it.

Exercise 2:

- Create class Rectangle. It has two data members :length and width of rectangle.

-create method calculateArea will return the area of rectangle.

-Create another class named runner that will create an object of Rectangle class, add values for length and width, then invoke the method calculateArea. * Area of rectangle = width * hight

Exercise 3

Create a new JAVA project and add the following code:

-Write then execute this code

-What is the output of this code?

- Add a new car and display it

```
import javax.swing.JOptionPane;


class CarPart {
private String modelNumber;
private String partNumber;
private String cost;



public void setparameter(String x, String y,String z) {
modelNumber=x;
partNumber=y;
cost=z;
}


public void display()
{
System.out.println("Model Number: " + modelNumber + "  part Number: "+partNumber + " Cost " +
cost);
}
}
public class CarPartRunner {
public static void main(String [] args) {
CarPart car1=new CarPart ();
String x=JOptionPane.showInputDialog("What is Model Number?" );
String y=JOptionPane.showInputDialog("What is Part Number?" );
String z=JOptionPane.showInputDialog("What is Cost?" );
car1.setparameter(x,y,z);
car1.display();
}


}
```

Lab 02: Abstract Classes and Interfaces

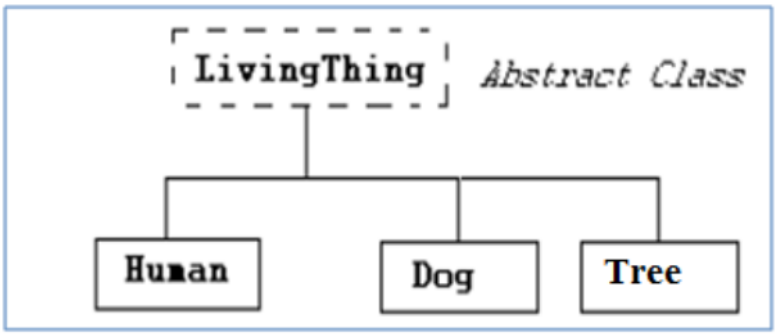
Objective

The purpose of this Lab is to practically familiarize students with the concept of Object Oriented Programming related to Abstract class and Interface. You will learn how to：

- Design and use abstract classes and abstract methods
- Specify common behavior for objects using interfaces
- Define interfaces and define classes that implement interfaces
- Define classes that implement interfaces using the implements keyword．
- Apply polymorphic referencing．
- Invoke methods through polymorphic referencing．

Exercise 1

We want to create a super class named Living Thing. This class has certain methods like breath, eat, sleep and walk. However, there are some methods in this super class wherein we cannot generalize the behavior. Take for example, the walk method. Not all living things walk the same way. Take the humans for instance; we humans walk on two legs, while other living things like dogs walk on four legs and trees don't walk. However, there are many characteristics that living things have in common that is why we want to create a general super class for this.



1. Create a super class named Living Thing, this class must contains common data members and constructors, methods and abstract methods．

2. Create sub-classes humans, dogs, and tree override the walk method by returning appropriate message in the child classes．

3. Create TestLivingThing main class to test your application, use polymorphism concepts while creating objects of sub-classes.

Exercise 2:

In an object-oriented drawing application, you can draw circles, rectangles, lines, curves, and many other graphic objects. These objects all have certain attributes (for example: position, orientation, line color, fill color) and behaviors (for example: moveTo, rotate, resize, draw) in common. Some of these states and behaviors are the same for all graphic objects—for example: position, fill color, and moveTo. Others require different implementations—for example, resize or draw. All Graphic Objects must know how to draw or resize themselves; they just differ in how they do it. This is a perfect situation for an abstract superclass. You can take advantage of the similarities and declare all the graphic objects to inherit from the same abstract parent object— for example, GraphicObject, as shown in the Figure 1 .

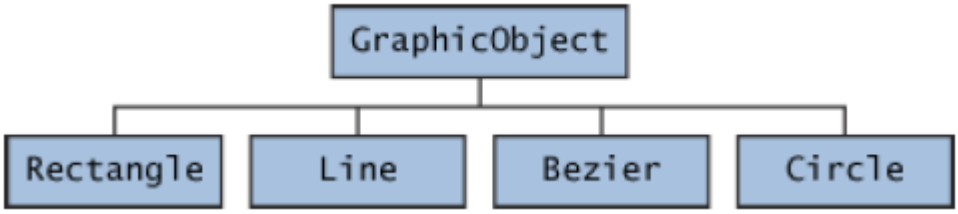


Figure 1: Classes Rectangle, Line, Bezier, and Circle inherit from GraphicObject

1. declare an abstract class, GraphicObject,
    a. provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method.
    b. declare abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways. The GraphicObject class can look something like this：
2. create non-abstract subclass of GraphicObject, such as Circle and Rectangle
    a. provide implementations for the draw and resize methods：
3. Create a main testclass to test your application. Since you are not drawing any figure, in all method you have to simply display the message as per the method, for example method move(4,9) will simply display message “Circle move to 4 , 9 axis” is Circle object invoke move method. In case of resize() method you can display message “Rectangle is Resized!” if Rectangle object invoke the resize() method. And so on …..

Exercise 3:

- **Interface Shape**

1. Create an interface named Shape and have one public abstract methods getName .

- **Point Class**

1. Point Class implements the interface Shape .
2. Class have two instance variables named x and y of data type integer .
3. It has no-argument and an argument constructor .
4. SetPoint method that initialize the instance variables .
5. Class have getter methods to return x and y values
6. Class have method get Name method

- **Circle Class**

1. Circle Class that extends the Point Class .
2. Class has instance variables named radius (data type integer) .
3. It has no-argument and an argument constructor .
4. Class has setter and getter methods for radius instance variable .
5. Class have method get Name method

- **Cylinder Class**

1. Cylinder Class that extends the Circle Class .
2. Class has instance variables named height (data type integer) .
3. It has no-argument and an argument constructor .
4. Class has setter and getter methods for height instance variable .
5. Class have method getName, area and volume

- **Test Application Class**

1. Test Application class.

```
Point: [7, 11]
Circle: Center = [22, 8]; Radius = 3.500000
Cylinder: Center = [10, 10]; Radius = 3.300000; Height = 10.000000

Point: [7, 11]
Area = 0.00
Volume = 0.00

Circle: Center = [22, 8]; Radius = 3.500000
Area = 38.48
Volume = 0.00

Cylinder: Center = [10, 10]; Radius = 3.300000; Height = 10.000000
Area = 275.77
Volume = 342.1
```

Broken Code

Interface Shape

```
// Shape.java
// Definition of interface Shape

public interface Shape
{
   public abstract String getName(); // return shape name
} // end interface Shape
```

Class Point

```
// Point.java
// Definition of class Point

public class Point implements Shape
{
   protected int x, y; // coordinates of the Point

   // no-argument constructor
   public Point()
   {
      setPoint( 0, 0 );
   }
```

```
// constructor
public Point( int xCoordinate, int yCoordinate )
{
   setPoint( xCoordinate, yCoordinate );
}

// Set x and y coordinates of Point
public void setPoint( int xCoordinate, int yCoordinate )
{
   x = xCoordinate;
   y = yCoordinate;
}

// get x coordinate
public int getX()
{
   return x;
}

// get y coordinate
public int getY()
{
   return y;
}

// convert point into String representation
public String toString()
{
   return String.format( "[%d, %d]", x, y );
}

// calculate area
public double area()
{
   return 0.0;
}
```

Broken Code

```
        // calculate volume
        public double volume()
        {
           return 0.0;
        }

        // return shape name
        public String getName()
        {
           return "Point";
        }
     } // end class Point
```

Class Circle

```
     // Circle.java
     // Definition of class Circle

     public class Circle extends Point
     {
        protected double radius;

        // no-argument constructor
        public Circle()
        {
           // implicit call to superclass constructor here
           setRadius( 0 );
        }
```

```
   // constructor
   public Circle( double circleRadius, int xCoordinate, int yCoordinate )
   {
      super( xCoordinate, yCoordinate ); // call superclass constructor

      setRadius( circleRadius );
   }
   // set radius of Circle
   public void setRadius( double circleRadius )
   {
      radius = ( circleRadius >= 0 ? circleRadius : 0 );
   }

   // get radius of Circle
   public double getRadius()
   {
      return radius;
   }

   // calculate area of Circle
   public double area()
   {
      return Math.PI * radius * radius;
   }

   // convert Circle to a String represention
   public String toString()
   {
      return String.format( "Center = %s; Radius = %f",
         super.toString(), radius );
   }

   // return shape name
   public String getName()
   {
      return "Circle";
   }
} // end class Circle
```

Broken Code

Class Cylinder

```
// Cylinder.java
// Definition of class Cylinder.

public class Cylinder extends Circle
{
   protected double height; // height of Cylinder

 // no-argument constructor
   public Cylinder()
   {
      // implicit call to superclass constructor here
      setHeight( 0 );
   }

   // constructor
   public Cylinder( double cylinderHeight, double cylinderRadius,
      int xCoordinate, int yCoordinate )
   {
      // call superclass constructor
      super( cylinderRadius, xCoordinate, yCoordinate );

      setHeight( cylinderHeight );
   }

   // set height of Cylinder
   public void setHeight( double cylinderHeight )
   {
      height = ( cylinderHeight >= 0 ? cylinderHeight : 0 );
   }
```

```
    // get height of Cylinder
    public double getHeight()
    {
       return height;
    }

    // calculate area of Cylinder (i.e., surface area)
    public double area()
    {
       return 2 * super.area() + 2 * Math.PI * radius * height;
    }

    // calculate volume of Cylinder
    public double volume()
    {
       return super.area() * height;
    }

    // convert Cylinder to a String representation
    public String toString()
    {
       return String.format( "%s; Height = %f",
          super.toString(), height );
    }

    // return shape name
    public String getName()
    {
       return "Cylinder";
    }
 } // end class Cylinder
```

Broken Code

Class Test

```
// Test.java
// Test Point, Circle, Cylinder hierarchy with interface Shape.

public class Test
{
   // test Shape hierarchy
   public static void main( String args[] )
   {
      // create shapes
      Point point = new Point( 7, 11 );
      Circle circle = new Circle( 3.5, 22, 8 );
      Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );

      Cylinder arrayOfShapes[] = new Cylinder[ 3 ]; // create Shape array

      // aim arrayOfShapes[ 0 ] at subclass Point object
      arrayOfShapes[ 0 ] = ( Cylinder ) point;

      // aim arrayOfShapes[ 1 ] at subclass Circle object
      arrayOfShapes[ 1 ] = ( Cylinder ) circle;

      // aim arrayOfShapes[ 2 ] at subclass Cylinder object
      arrayOfShapes[ 2 ] = ( Cylinder ) cylinder;

      // get name and String representation of each shape
      System.out.printf( "%s: %s\n%s: %s\n%s: %s\n", point.getName(),
         point, circle.getName(), circle, cylinder.getName(), cylinder );

      // get name, area and volume of each shape in arrayOfShapes
      for ( Shape shape : arrayOfShapes )
      {
         System.out.printf( "\n\n%s: %s\nArea = %.2f\nVolume = %.2f\n",
            shape.getName(), shape, shape.area(), shape.volume() );
      } // end for
   } // end main
} // end class Test
```

Lab 02: Inheritance and Polymorphism

Objective

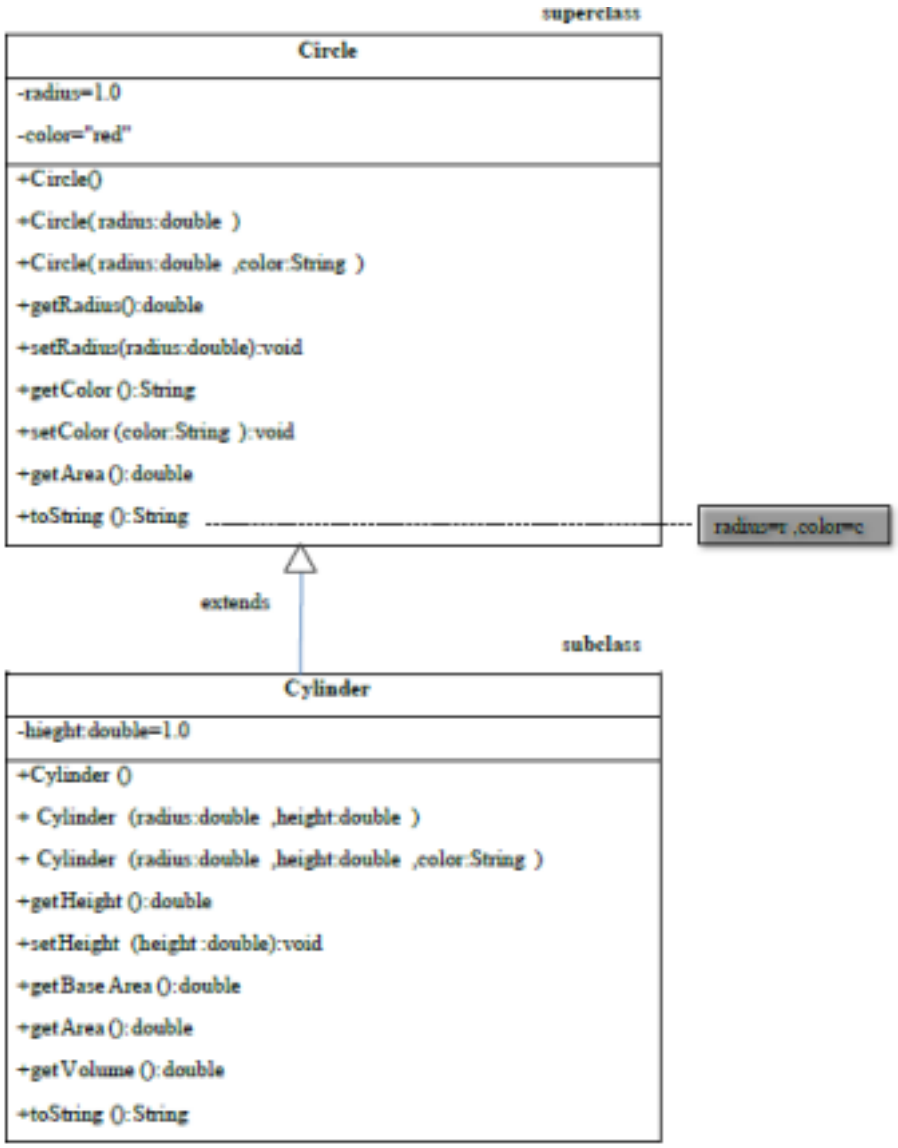
The purpose of this Lab is to practically familiarize students with the concept of Inheritance and Polymorphism. You will learn:

- Using is-a relationship.
- Emphasizing on software reusability promoted by inheritance.
- Clarifying the notions of superclasses and subclasses.
- To use keyword extends to create a class that inherits attributes and behaviors from another class.
- To access superclass members with 'super' keyword.
- Clarifying the role of constructors in inheritance hierarchies.
- Introducing the concept of method overriding.

Exercise 1:

- Create Cylinder Class as described in the UML.
- Add all the required codes to complete the relationship.
- Use the **super** identifier to apply the inheritance relationship.
- Add the toString method to the cylinder class to display the cylinder information as shown in the sample output.
- Override the getArea method inside the Cylinder class to get the cylinder surface area instead of the circle area.
- Create Main Class and name it "TestCylinder".
- Circle area= base_area=radius * radius * 3.14
- Cylinder surface area=2* 3.14 *radius*height + 2*base_area
- Cylinder volume= base_area*height

```
***Cylinder***
--------------
Please, enter the cylinder's base radius:5.0
--------------
Please, enter the cylinder's height:2.0 --------------
Cylinder information: base radius=5.0 ,color=red ,height=2.0 ,base area=78.53981633974483
,Surface area=219.9114857512855 ,volume=157.07963267948966
```


superclass
Circle
-radius=1.0
-color="red"
+Circle()
+Circle(radius:double)
+Circle(radius:double ,color:String)
+getRadius():double
+setRadius(radius:double):void
+getColor ():String
+setColor (color:String):void
+getArea ():double
+toString ():String
radius=r ,color=c
extends
subclass
Cylinder
-hieght:double=1.0
+Cylinder ()
+ Cylinder (radius:double ,height:double)
+ Cylinder (radius:double ,height:double ,color:String)
+getHeight ():double
+setHeight (height :double):void
+getBase Area ():double
+getArea ():double
+getVolume ():double
+toString ():String

Template

```
// Save as "Circle.java" public class Circle {
 // private instance variables
   private double radius;
   private String color;

// Constructors
   public Circle() {  this.radius = 1.0;      this.color = "red";  }
   public Circle(double radius) {  this.radius = radius;      this.color = "red"; }
   public Circle(double radius, String color) {   this.radius = radius;      this.color = color;}

 // Getters and Setters
    public double getRadius() {      return this.radius;   }
    public String getColor() {      return this.color; }
    public void setRadius(double radius) {      this.radius = radius; }
    public void setColor(String color) {  this.color = color;   }

   // Describle itself
    public String toString() { return "radius=" + radius + " ,color=" + color ;   }
  // Return the area of this Circle
     public double getArea() {      return radius * radius * Math.PI;    }
}
//-----------------------------------------------------------------------
// Save as "Cylinder.java"
public class Cylinder          ..................  {
                               ..................  }
//-----------------------------------------------------------------------
// save as "TestCylinder.java"
public class TestCylinder      {
                               ..................
                               }
```

Implement the TestCylinder Class to test your Cylinder class.
Copie the following code in `TestCylinder.java`

```
1
 /*
2
  * A test driver for the Cylinder class.
3
  */
4
 public class TestCylinder {
5
  public static void
main(String[] args) { 6
  Cylinder cy1 = new Cylinder();
7
  System.out.println("Radius is " +
cy1.getRadius() 8
```

```
   + " Height is " + cy1.getHeight()
 9
   + " Color is " + cy1.getColor()
10
   + " Base area is " +
cy1.getArea() 11
   + " Volume is " +
cy1.getVolume()); 12

13
   Cylinder cy2 = new
Cylinder(5.0, 2.0); 14
   System.out.println("Radius is " +
cy2.getRadius() 15
   + " Height is " + cy2.getHeight()
16
   + " Color is " + cy2.getColor()
17
   + " Base area is " + cy2.getArea()
```



```
18
   + " Volume is " +
cy2.getVolume()); 19
   }
20
  }
```

Lab 04: Java Collection Framework

• **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.

• **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.

• **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

HashSet Exercises

1. Write a Java program to append the specified element to the end of a hash set

Sample Output:

```
The Hash Set: [Red, White, Pink, Yellow, Black, Green]
```

2. Write a Java program to convert a hash set to a List/ArrayList.

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow,
Black, Green]
ArrayList contains: [Red, White, Pink, Yellow,
Black, Green]
```

3. Write a Java program to compare two sets and retain elements which are same on both sets.

Sample Output:

```
Frist HashSet content: [Red, White, Black, Green]
Second HashSet content: [Red, Pink, Black, Orange]
HashSet content:
[Red, Black]
```

TreeSet Exercises

4. Write a Java program to get the number of elements in a tree set.

Sample Output:

```
Original tree set: [Black, Green, Pink, Red,
orange]
Size of the tree set: 5
```

5. Write a Java program to get the element in a tree set which is strictly greater than or equal to the given element.

Sample Output:

```
Strictly greater than 76 : 82
Strictly greater than 31 : 36
```

HashMap Exercises

6. Write a Java program to associate the specified value with the specified key in a HashMap.

Sample Output:

```
1 Red
2 Green
3 Black
4 White
5 Blue
```

7. Write a Java program to test if a map contains a mapping for the specified value.

Sample Output:

```
The Original map: {1=Red, 2=Green, 3=Black, 4=White, 5=Blue}
1. Is value 'Green' exists?
Yes!

2. Is value 'orange' exists?
No!
```

TreeMap Exercises

8. Write a Java program to get the first (lowest) key and the last (highest) key currently in a map.

Sample Output:

```
Orginal TreeMap content: {C1=Green, C2=Red, C3=White,
C4=Black}
Greatest key: C1
Least key: C4
```

9. Write a Java program to get the portion of this map whose keys are less than (or equal to, if inclusive is true) a given key.

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 40=Black, 50=White,
60=Pink
}
Checking the entry for 10:
Key(s): {10=Red}
Checking the entry for 20:
Key(s): {10=Red, 20=Green}
Checking the entry for 70:
Key(s): {10=Red, 20=Green, 40=Black, 50=White, 60=Pink}
```

ArrayList Exercises

10. Write a Java program to print all the elements of a ArrayList using the position of the elements.

Sample Output:

```
Original array list: [Red, Green, Black, White, Pink]

Print using index of an element:
Red
Green
Black
White
Pink
```

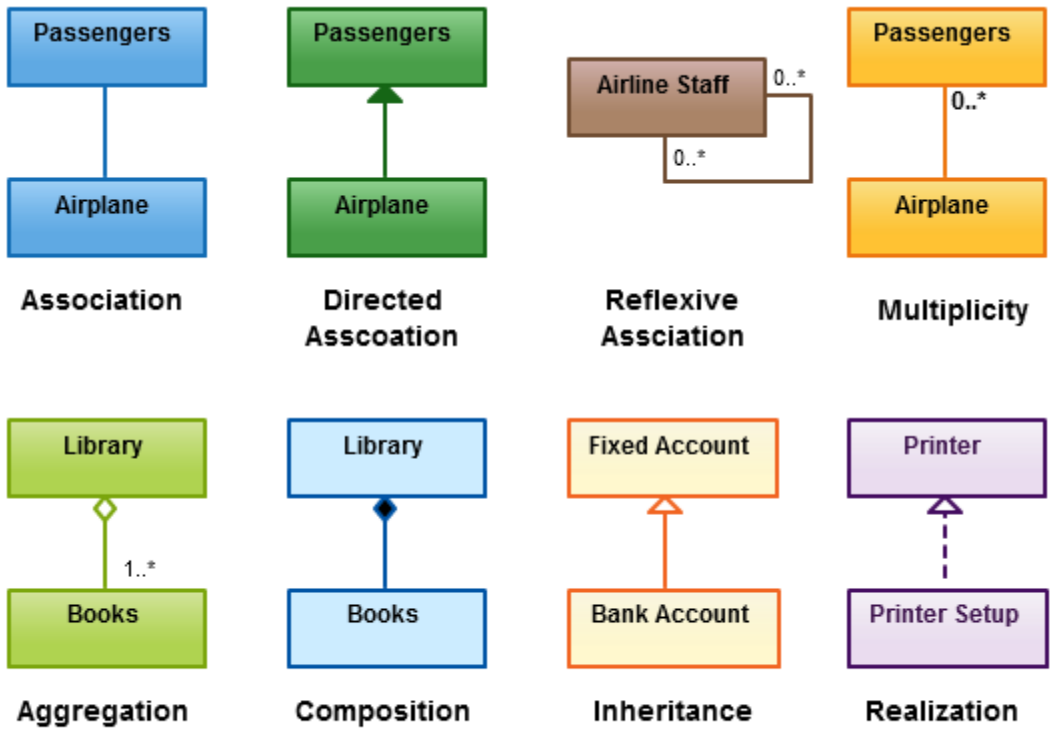
LinkedList Exercises

11. Write a Java program to shuffle the elements in a linked list.

Sample Output:

```
Linked list before shuffling:
[Red, Green, Black, Pink, orange]
Linked list after shuffling:
[orange, Pink, Red, Black, Green]
```

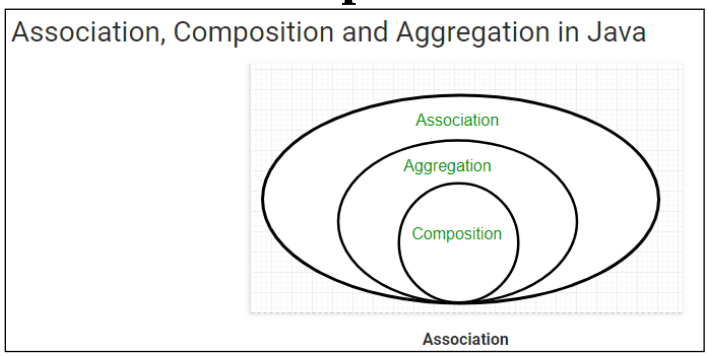
Lab 05: Class to JAVA(Java relationships)



Relationships in UML class diagrams

This lab explores three relationships of class including: **association, aggregation, composition**

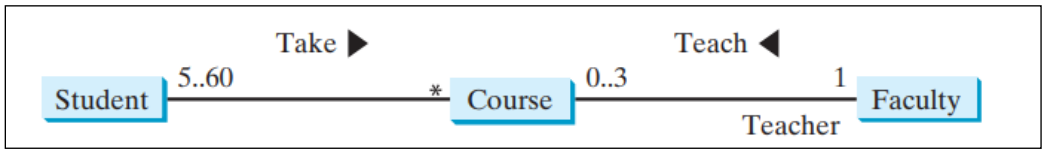
General Concept:



	Association	Aggregation	Composition
Related to Association		Special type of Association.	Special type of Aggregation
		Weak association	Strong association
Relation		**Has-A**	**Owns**
		one object is the owner of another object.	one object is contained in another object
Owner	No owner	Single owner	Single owner
Life-cycle	own life-cycle	own life-cycle	owner life-cycle
Child object	independent	belong to single parent	belong to single parent

Exercise 1: (Association)

Convert the schema of the following class diagram to java code



Add the following attribute to the **Student class** :
Add the following attributes to the **Student class** :

- number (type: int)
- name (type: string)
- age (type: int)

Add the following methods to the **Student class** :

- getNumber()
- ….

Add the following attributes to the **course class** :

- courseName (type: string)
- noOfStudents (type: string)
- teacher (type: string)

Add the following methods to the **course class** :

- getCourseName()
- ….
- setCourseName()
- ….

printStudents() : to print all students
Course() : default constructor
Course(para1, para2, para3)

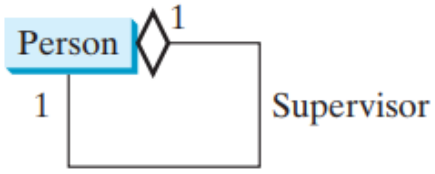
- Para1 : noOfStudents(type int)

- Para2 : courseName(type string)
- Para3 : teacher (type string)

Write a main program that offers a course entitled: "Object Oriented Programming" for his teacher "LeBron James". Add three students to this course.

Exercise 2: (Aggregation)

Convert the schema of the following class diagram to java code. Add the necessary attributes and methods



Exercise 3: (Composition)

Convert the schema of the following class diagram to java code. Add the necessary attributes and methods