CEGEP VANIER COLLEGE
CENTRE FOR CONTINUING EDUCATION
Advanced Programming in Java
420-984-VA

Teacher: Samir Chebbine **Lab 3** **Jul 11, 2022**

Lab 3: Abstract Classes, Interface, Composition in Java

Create and Submit a Word file ***Lab3OOPProgramminIIYourName.doc*** which contains Answers of Book Exercises and output screenshots for every Java Project. Submit the Java projects too.

1. **Abstract methods and Abstract Classes**
   Create the Project **AbstractShapeProject** of Figure 1. *Shape* class is called the Abstract *Superclass* and *RectangleShape*, *BoxShape* classes are the *Subclasses.*
   Complete all these following programs as explained in my **Lab 3 YouTube Video 1.** Notice all *missing* coding statements are presented in this video with explanation.

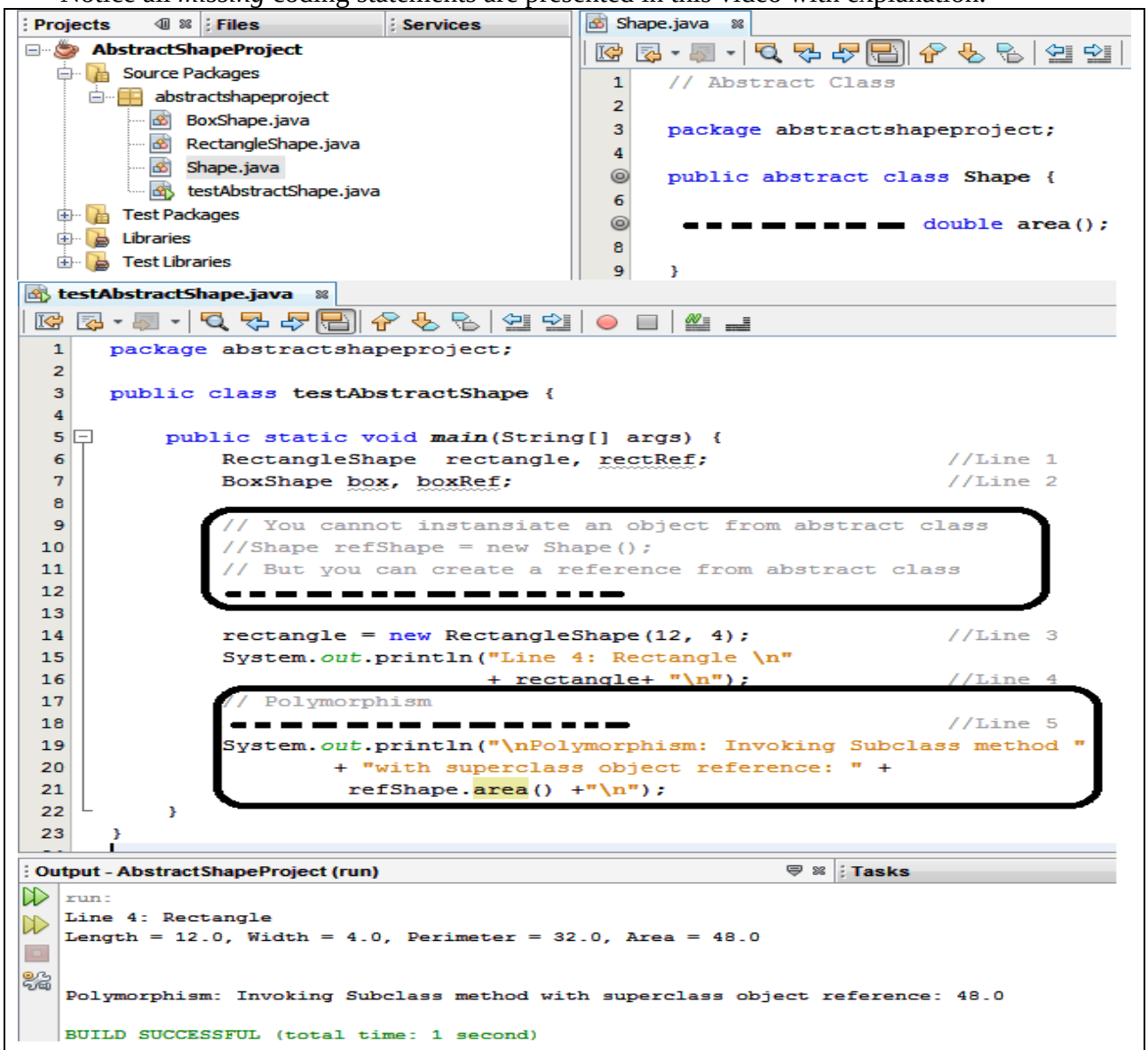


Figure 1

2. Interface Classes

Create the Project **InterfaceShapeProject** of Figure 2. *AngleShape* class is called the *interface class* and *RectangleShape* class is the *Subclass.*

RectangleShape.java

```
package geometryproject;
public  class RectangleShape extends Shape
                                 implements AngleShape
```

Projects | Files

- InterfaceShapeProject
  - Source Packages
    - interfaceshapeproject
      - AngleShape.java
      - BoxShape.java
      - RectangleShape.java
      - Shape.java
      - testInterfaceShape.java

AngleShape.java

```
1  //Interface class
2
3  package interfaceshapeproject;
4
6  public interface AngleShape {
       ------------------ 45;
       public void printShapeInfo();
```

(Testing the Shape, *RectangleShape* and *BoxShape* classes) Create the Java Program *TestInterfaceShape.java* of Figure 2 to test how to call a method declared in an interface.

Start Page | AngleShape.java | BoxShape.java | RectangleShape.java | testInterfaceShape.java

Source | History

```
11
12      public static void main(String[] args) {
13          RectangleShape  rectangle, rectRef;
14          BoxShape box, boxRef;
25
26          // You cannot instansiate an object from Interface
27          // AngleShape angleShapeRef = new AngleShape();
28          // But you can create a reference from abstract class
29          ------------------
30
31          rectangle = new RectangleShape(12, 4);              //Line 6
32          System.out.println("Line 6: Rectangle \n"
33                          + rectangle+ "\n");                 //Line 7
34          // Polymorphism  via interface
35          ------------------                                  //Line 8
36
37          // Calling method defined in Interface AngleShape
38          System.out.println("Calling method defined in interface AngleShape:");
39          angleShapeRef.printShapeInfo();
40
41
42          boxRef = new BoxShape(10,2,3);                      //Line 9
43          System.out.println("Line 6: boxRef \n"
44                          + boxRef+ "\n");                    //Line 10
45          // Polymorphism interface
46          ------------------                                  //Line 11
47          // Calling method defined in Interface AngleShape
48          System.out.println("Calling method defined in interface AngleShape:");
49          angleShapeRef.printShapeInfo();
50      }
51  }
```

Projects | Files

- InterfaceShapeProject
  - Source Packages
    - interfaceshapeproject
      - AngleShape.java
      - BoxShape.java
      - RectangleShape.java
      - Shape.java
      - testInterfaceShape.java

Output - InterfaceShapeProject (run)

```
run:
Line 4: Rectangle
Length = 12.0, Width = 4.0, Perimeter = 32.0, Area = 48.0


Polymorphism: Invoking Subclass method with superclass object reference: 48.0

Calling method defined in interface AngleShape:
Length = 12.0, Width = 4.0, angle shape = 45.0
Line 6: Rectangle
Length = 12.0, Width = 4.0, Perimeter = 32.0, Area = 48.0

Calling method defined in interface AngleShape:
Length = 12.0, Width = 4.0, angle shape = 45.0
Line 6: boxRef
Length = 10.0, Width = 2.0, Height = 3.0, Surface Area = 112.0, Volume = 60.0

Calling method defined in interface AngleShape:
Length = 10.0, Width = 2.0, angle shape = 22.5
BUILD SUCCESSFUL (total time: 2 seconds)
```

Figure 2

3. Composition and aggregation

Create the Project **CompositionShapeProject** of Figure 3. Object from *Circle is a member of RectangleShape* object and *BoxShape* object. *So*, object *RectangleShape is* ***composed of*** object from Circle class type.

Circle.java

```java
package compositionshapeproject;

public class Circle {

    private double radius;

      //Method to set the radius of the
      //Sets the radius to the value spe
    public void setRadius(double r)
    {
        radius = r;
    }

      //Method to return the radius of t
      //Returns the radius of the circle
    public double getRadius()
    {
        return radius;
    }
}
```

BoxShape.java

```java
package instanceofproject;
public class BoxShape exten
{
    private double height;


    public BoxShape()
        {
            super();
            height = 0;
        }
```

Projects | Files

- CompositionShapeProject
  - Source Packages
    - compositionshapeproject
      - AngleShape.java
      - BoxShape.java
      - Circle.java
      - RectangleShape.java
      - Shape.java
      - testCompositionShape.java

RectangleShape.java

```java
package compositionshapeproject;

public  class RectangleShape extends Shape
        implements AngleShape
{
    protected double length;
    protected double width;
    public Circle circle1;
```

(Testing the Shape, *RectangleShape* and *BoxShape* classes) Create the Java Program *testCompositionShape.java* of Figure 3 to test how to call a method declared in an interface.

testCompositionShape.java

```java
package compositionshapeproject;
public class testCompositionShape {

    public static void main(String[] args) {
        RectangleShape  rectangle, rectRef;                    //Line 1
        BoxShape box, boxRef;                                  //Line 2
        // You cannot instansiate an object from abstract class
        //Shape refShape = new Shape();
        // But you can create a reference from abstract class
        Shape refShape;

        rectangle = new RectangleShape(12, 4);                 //Line 3
        System.out.println("Line 4: Rectangle \n"
                           + rectangle+ "\n");                 //Line 4
        // Polymorphism
        refShape = rectangle;                                  //Line 5
        System.out.println("\nPolymorphism: Invoking Subclass method "
                + "with superclass object reference: " +
                 refShape.area() +"\n");
         // Calling method defined in Interface CourseNight
        System.out.println("Calling method defined in interface AngleShape:
        rectangle.printShapeInfo();
         // Using Composition  of the class Circle
        ----------------------------
        System.out.println("Using Composition, the raduis Subject:" +
                          ---------------------------- );
    }
}
```

Output - CompositionShapeProject (run) | Tasks

```
run:
Line 4: Rectangle
Length = 12.0, Width = 4.0, Perimeter = 32.0, Area = 48.0


Polymorphism: Invoking Subclass method with superclass object reference: 48.0

Calling method defined in interface AngleShape:
Length = 24.0, Width = 12.0
Using Composition, the raduis Subject:45.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 3

4. Complete Project SportProject from Lab 1:

a) (**Polymorphism**)

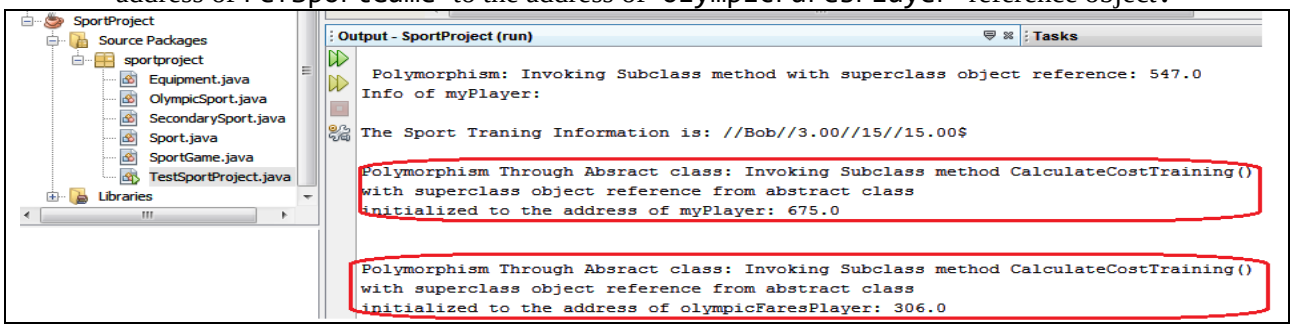
- Within **`TestSport`**, call the method implementing the cost of training `CalculateCostTraining()` of the sub class **`OlympicSport`** with the super class object `yourPlayer` as shown hereafter.
- Test the polymorphism through other sub class objects.

```
The Sport Training Information is : Irena//2.00//18//15.00$//7.00$


Polymorphism: Invoking Subclass method with superclass object reference: 547.0

BUILD SUCCESSFUL (total time: 10 seconds)
```

b) (**Abstract Class**)

Define a new abstract class called **`SportGame`**, which specifies the implemented overriding method called **`CalculateCostTraining ()`** (method already implemented in super class Sport and sub class OlympicSport).

1) You have to make **`Sport`** as a sub class of the new abstract class **`SportGame`**.
2) Within **`TestSport`**, call the method implementing the cost of training **`CalculateCostTraining()`** of the sub class **`Sport`** with the super class object `refSportGame` from **`SportGame`** abstract class data type as shown hereafter. Initialize the address of `refSportGame` to the address of `myPlayer` reference object.
3) Within **`TestSport`**, call the method implementing the cost of training **`CalculateCostTraining()`** of the sub class **`OlympicSport`** with the super class object `refSportGame` from **`SportGame`** abstract class data type as shown hereafter. Initialize the address of `refSportGame` to the address of `olympicFaresPlayer` reference object.



c) (**Interface**)

Define an interface class called **`SecondarySport`**, which specifies a given method heading called *SumPro* that returns the sum of its data member's `cost_year1` = 2013 and `cost_year2`= *2014*.

1) You have to implement **`OlympicSport`** on top of `SecondarySport` interface.
2) Within `TestSport`, call the method `SumPro()` with the sub class object `olympicSarahPlayer`.

```
Polymorphism Through Absract class: Invoking Subclass method CalculateCostTraining()
with superclass object reference from abstract class
initialized to the address of olympicFaresPlayer: 306.0

Calling method defined in interface SecondarySport SumPro return: 4027.0

Using Composition, the Equipment Title: Apparatus For Body Building
BUILD SUCCESSFUL (total time: 12 seconds)
```

d) (**Composition**)

Define a new class **`Equipment`** that includes two private data members `equi_Title` of type `String` and `equi_Price` of type `double`. Add an object of type new class called `Equipment` as public member of **`OlympicSport`**. Within `TestSport`, set `equi_Title` of object `olympicSarahPlayer` to `"Apparatus For Body Building".` Call the method `setEqui_Title ()` of `Equipment` class with a sub class object `olympicSarahPlayer` and display the following output.

5. Add Java Statements if required. Using your own wording, answer the following questions:

a) What is the purpose of `Abstract class, Interface, and Composition`*?*

b) Assume two classes `Book` and `Chapter`. Are we implementing Inheritance or Composition? Why?

c) T/F. You cannot define method with body in `abstract class`.

d) T/F. You can define method with body in `interface type`*?*

e) T/F. You can instantiate an object of `Abstract class type`*?*

f) Give an example of super `Abstract class` and concrete sub class in **your own** stated project (other than Geometry, Sport).

g) Add an `abstract method` to the specified super `Abstract class`.

h) Provide detail implementation of the method to be defined in the `sub class`. Write then a Java statement to instantiate an object from sub class.

i) Apply polymorphism with the super abstract class reference object from `question (f)` to invoke the **overriding** method of sub class defined in `question (h)`. Write then the appropriate Java statements on how to use **polymorphism via Abstract class**.

j) Give an example of an `Interface` where concrete sub class defined in `question (h)` will be implemented on top of that `interface`.

k) Add an interface method heading to the specified `Interface`.

l) Apply **polymorphism** with the interface reference object from `question (j)` to invoke the method of sub class defined in `question (k)`. Write then the appropriate Java statements on how to use **polymorphism via Interface**.

m) Give an example of a `class` where its instantiated object will be an *inner* object of the *outer* object from the sub class type defined in `question (h)`.

n) Define then **private** data attributes of the specified *inner* class type. (give at least two data attributes)

o) Apply **composition** with the super abstract class reference object from `question (f)` to display the values of data attributes defined in (n) of a given object from sub class type defined in `(h)`.

p) Assume two classes `Home` and `Room`. Are we implementing Inheritance or Composition? Why?

q) Assume two classes `Account` and `InvestmentAccount`. Are we implementing Inheritance or Composition? Why?

r) Assume two classes `Computer` and `RAM`. Are we implementing Inheritance or Composition? Why?

s) Assume two classes `Employee` and `PartTimeEmployee`. Are we implementing Inheritance or Composition? Why?