Lab 3: Abstract Classes, Interface, Composition in Java

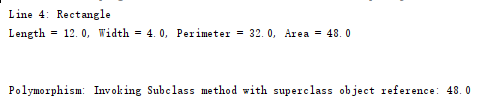
1. Abstract methods and Abstract Classes

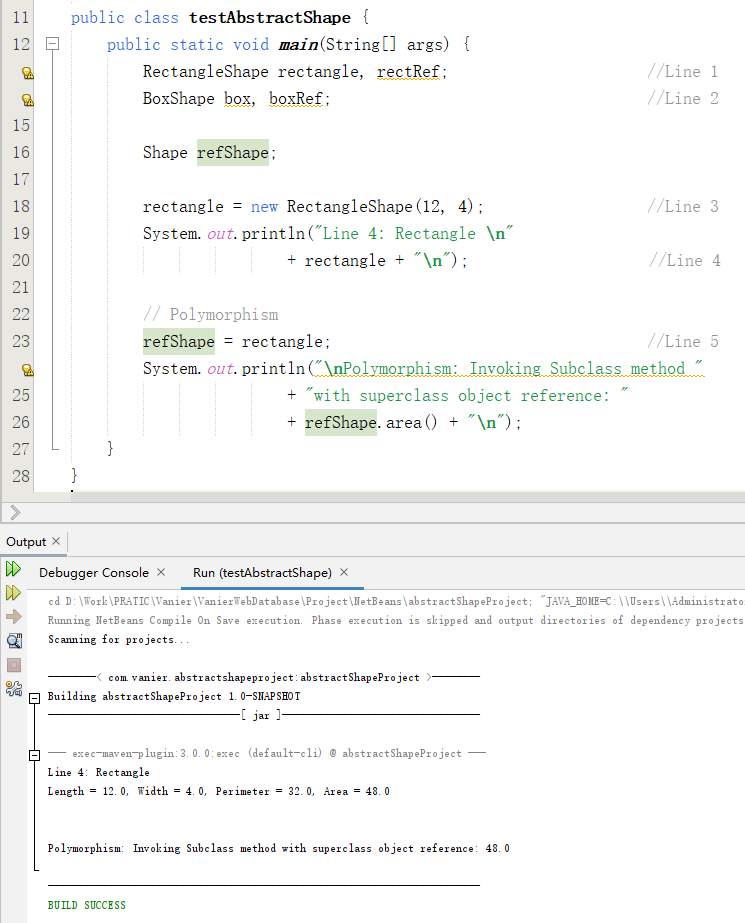
Shape.java

1. **public** **abstract** **class** Shape {
2. //area is overloading method in BoxShape and RectangleShape
3. **public** **abstract** **double** area();
4. }

testAbstractShape.java

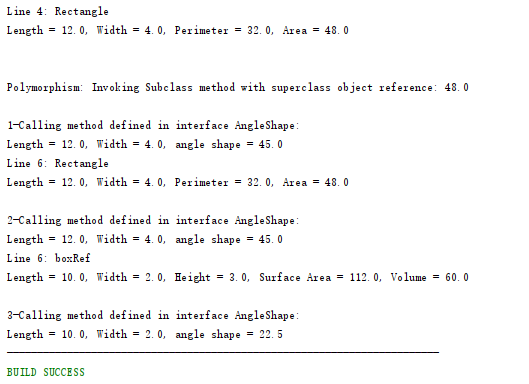
1. **public** **class** testAbstractShape {
2. **public** **static** **void** main(String[] args) {
3. RectangleShape rectangle, rectRef;                      //Line 1
4. BoxShape box, boxRef;                                   //Line 2
6. Shape refShape;
8. rectangle = **new** RectangleShape(12, 4);                  //Line 3
9. System.out.println("Line 4: Rectangle \n"
10. + rectangle + "\n");                    //Line 4
11. // Polymorphism
12. refShape = rectangle;                                   //Line 5
13. System.out.println("\nPolymorphism: Invoking Subclass method "
14. + "with superclass object reference: "
15. + refShape.area() + "\n");
16. }
17. }





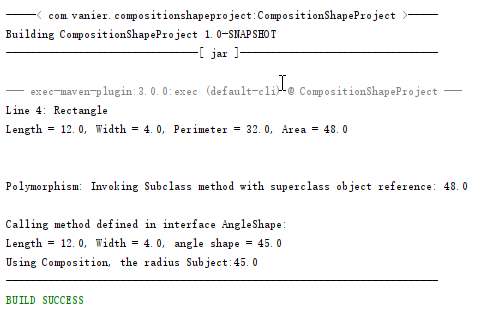
2. Interface Classes

1. **public** **class** testInterfaceShape {
2. **public** **static** **void** main(String[] args) {
3. RectangleShape rectangle, rectRef;
4. BoxShape box, boxRef;
6. // Create a reference from abstract class
7. rectangle = **new** RectangleShape(12, 4);                    //Line 3
8. System.out.println("Line 4: Rectangle \n"
9. + rectangle + "\n");                      //Line 4
11. System.out.println("\nPolymorphism: Invoking Subclass method "
12. + "with superclass object reference: "
13. + rectangle.area() + "\n");
14. AngleShape angleShapeRef;
15. // Calling method defined in Interface AngleShape
16. angleShapeRef = rectangle;
17. System.out.println("Calling method defined in interface AngleShape:");
18. angleShapeRef.printShapeInfo();
19. rectangle = **new** RectangleShape(12, 4);                    //Line 6
20. System.out.println("Line 6: Rectangle \n"
21. + rectangle + "\n");                //Line 7
22. // Polymorphism via interface
23. boxRef = **new** BoxShape(10, 2, 3);
24. angleShapeRef = boxRef;                                //Line 8
25. // Calling method defined in Interface AngleShape
26. System.out.println("Calling method defined in interface AngleShape:");
27. angleShapeRef = rectangle;
28. angleShapeRef.printShapeInfo();
29. boxRef = **new** BoxShape(10, 2, 3);                        //Line 9
30. System.out.println("Line 6: boxRef \n"
31. + boxRef + "\n");                       //Line 10
32. // Polymorphism interface
33. angleShapeRef = boxRef;                                 //Line 11
34. // Calling method defined in Interface AngleShape
35. System.out.println("Calling method defined in interface AngleShape:");
36. angleShapeRef.printShapeInfo();
37. }
38. }



**3. Composition and aggregation**

1. **public** **class** testCompositionShape {
2. **public** **static** **void** main(String[] args) {
3. RectangleShape rectangle, rectRef;                      //Line 1
4. BoxShape box, boxRef;                                   //Line 2
5. Shape refShape;
7. rectangle = **new** RectangleShape(12, 4);                  //Line 3
8. System.out.println("Line 4: Rectangle \n"
9. + rectangle + "\n");                    //Line 4
11. // Polymorphism
12. refShape = rectangle;                                   //Line 5
13. System.out.println("\nPolymorphism: Invoking Subclass method "
14. + "with superclass object reference: "
15. + refShape.area() + "\n");
17. // Calling method defined in Interface AngleShape
18. System.out.println("Calling method defined in interface AngleShape:");
19. rectangle.printShapeInfo();
21. // Using Composition of the class Circle
22. rectangle.circle1.setRadius(45.0);
23. System.out.println("Using Composition, the radius Subject:"
24. + rectangle.circle1.getRadius());
25. }
27. }



**4. Complete Project SportProject from Lab 1:**

**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?

**Abstract** classes provide a simple and easy way to version the components.

An **Interface** is also a user-defined type that is syntactically similar to a class. It can have a collection of field constants and method signatures that will be overridden by interface implementing classes.

And another technique for software reuse is **Composition**. By Composition, in our classes under development, we explicitly declare data members as objects of some existing classes.

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

Composition is better, because the Book has a Chapter.

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

False

An abstract class could have abstract methods or not, and only the abstract method cannot have body. If an abstract class has the normal methods, these methods should have bodies.

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

False

All methods in the Interface must be abstract methods while Interface is not Class.

And after Java 8, non-abstract methods can be modified with the default keyword in the interface.

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

False

Abstract classes cannot be instantiated.

And we could indirectly instantiate the super abstract class by pointing to the instance of the sub class through the application of the super class, because the sub class must instantiate its super class before instantiating it. In this way, a sub class object that inherits the abstract class is created, and its super class (abstract class) is instantiated.

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

public abstract class Book {

public String bookname;

public abstract void buy();

public abstract void read();

}

class eBook extends Book {

@Override

public void buy() {

System.out.println("Paid");

System.out.println("Download the eBook to my computer");

}

@Override

public void read() {

System.out.println("Click the book’s icon");

System.out.println("Showing the chapter through the link in the content");

}

}

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

public abstract class Book {

public String bookname;

public abstract void buy();

public abstract void read();

public abstract void on\_shelf();

}

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.

public class eBook extends Book {

@Override

public void buy() {

System.out.println("Paid");

System.out.println("Download the eBook to my computer");

}

@Override

public void read() {

System.out.println("Click the book’s icon");

System.out.println("Showing the chapter through the link in the content");

}

@Override

public void on\_shelf() {

System.out.println("Upload the file of the eBook.");

}

}

eBook eDataStructures = new eBook();

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.

class PaperBook extends Book {

@Override

public void buy() {

System.out.println("Paid");

System.out.println("Waiting till the books is delivered to my house");

}

@Override

public void read() {

System.out.println("Open the book");

System.out.println("Flip the pages and read");

}

}

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

public interface iBook {

void buy();

void read();

}

public class PaperBook implements iBook {

public String bookname;

public void PaperBook(String name) {

bookname = name;

}

@Override

public void buy() {

System.out.println("Paid");

System.out.println("Waiting till the books is delivered to my house");

}

@Override

public void read() {

System.out.println("Open the book");

System.out.println("Flip the pages and read");

}

}

PaperBook computer(“Data and Structures”) = new PaperBook;

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

public interface iGift {

void setMemorialDay();

void setGreetings();

}

public BookAsGift implements iBook, iGift {

}

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.

public BookAsGift implements iBook, iGift {

public String bookname;

private String strMemorialDay;

private String strGreetings;

@Override

public void buy() {

System.out.println("Paid & deliver");

}

@Override

public void read() {

System.out.println("Open the book & read");

}

@Override

public void setMemorialDay(String date) {

strMemorialDay = date;

}

@Override

public void setGreetings(String words) {

strGreetings = words;

}

}

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).

public class eBook extends Book {

public class Download {

public Boolean downloaded;

}

Download download = new Download;

download.downloaded = true;

}

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

public class eBook extends Book {

public class Download {

public Boolean downloaded;

private int filesize;

private int downloadTimes;

public int getFilesize {

return filesize;

}

public getDownloadTimes {

return downloadTimes;

}

}

Download download = new Download;

download.downloaded = true;

}

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).

public abstract class Book {

}

public class eBook extends Book {

}

eBook eDataStructures = new eBook();

System.out.println(“The eBook ” + eDataStructures.bookname

” file size is ” + eDataStructures.getFilesize());

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

Inheritance is better, because Home is a kind of Room.

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

Inheritance is better, because InvestmentAccount is a kind of Account.

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

Composition is better, because Computer has a RAM.

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

Inheritance is better, because PartTimeEmployee is a kind of Employee.