Uppgift

Lös koppling mellan programdelar (klasser) eftersträvas ofta i objektorienterad programmering. Om klassen A beror av klassen B, hur kan man lösa upp detta beroende och åstadkomma en lösare koppling mellan klasserna A och B?

Svaret kan ges i Javakod (eller i UML-klassdiagram).

```
public class A {
  B minB;
  ...
}
public class B {
   <B:s många variabler och metoder>
}
```

Lösning:

Enklaste lösningen (men flera alternativ finns) är att införa ett interface som A är beroende av och som B får implementera. Det är "lösare" att bero av ett interface än en klass med implementerade metoder.

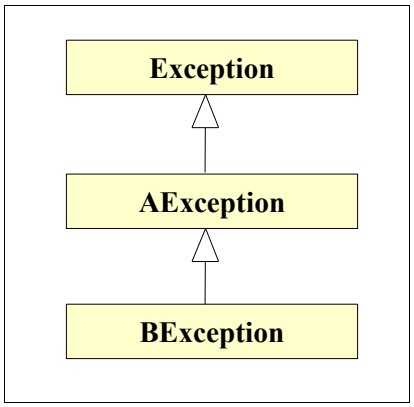
```
public class A {
    BI minB;
    .....
}

public interface BI {
   // skriv huvuden för alla metoder som B ska ha
}

public class B implements BI {
  // implementera alla metoder som står i BI
}
```

Uppgift

Antag att klasserna AException och BException är relaterade enligt UML-diagrammet till höger. Vad skrivs ut i nedanstående program?




```
public class ExceptionTester {
    public static void main (String[] args) {
        Catcher theCatcher = new Catcher();
        for( int val = -10; val <= 10; val += 10 ) {
            try {
                theCatcher.catchIt(val);
            }
            catch( BException e ) {
                System.out.println( "main caught an exception" );
            }
        }
    }
}
public class Catcher {
    public void catchIt(int send) throws BException {
        Pitcher aPitcher = new Pitcher();
        try {
            aPitcher.throwIt (send);
        }
        catch (AException e) {
            System.out.println( "catchIt caught an exception" );
        }
    }
}
public class Pitcher {
    public void throwIt(int a) throws AException, BException {
        if( a < 0 )
            throw new AException();
        else if (a == 0)
            throw new BException();
        else
            System.out.println( "In throwIt a is: " + a );
    }
}
```

Lösning:

```
catchIt caught an exception
catchIt caught an exception
In throwIt a is: 10
```

Uppgift

Förklara varför nedanstående klasser bryter mot Liskov Substitution Principle.

```
public abstract class Payment {
   /**
       @pre amt >= 0
   **/
   public void setPaymentAmount(int amt) { . . . }
}
public class CreditCardPayment extends Payment {
   /**
       @pre  amt > = 250
   **/
   public void setPaymentAmount(int amt) { . . . }
}
public class CashPayment extends Payment { . . . }
```

Lösning:

```
Payment p = new CashPayment();
p .setPaymentAmount(5);
p. = new CrediteCardPayment();
p.setPaymentAmount(5); //oops! Strider mot förvillkoret!!
```

Uppgift

Betrakta nedanstående klasser:

```
public class FamilyTester {
   public static void main(String[] args) {
      Grandparent person = new Grandparent();
      person.whoAmI();
      person = new Parent();
      person.whoAmI();
      person = new Child();
      person.whoAmI();
   }
}

public class Grandparent {
   public void whoAmI() {
      System.out.println("I'm a Grandparent");
   }
}

public class Parent extends Grandparent {
   public String toString() {
      return "I'm a Parent";
   }
}

public class Child extends Parent {
   public void whoAmI() {
      System.out.println("I'm a Child");
   }
}
```

Vad blir utskriften när main-metoden exekveras?

Lösning:

I’m a Grandparent
I’m a Grandparent
I’m a Child

Uppgift

Antag att klassen B är intresserad av alla tillståndsförändringar som sker i A. När instansvariabeln i A ändras skall B skriva ut det nya värdet på konsolen. Komplettera koden så att designmönstret Observer realiseras.

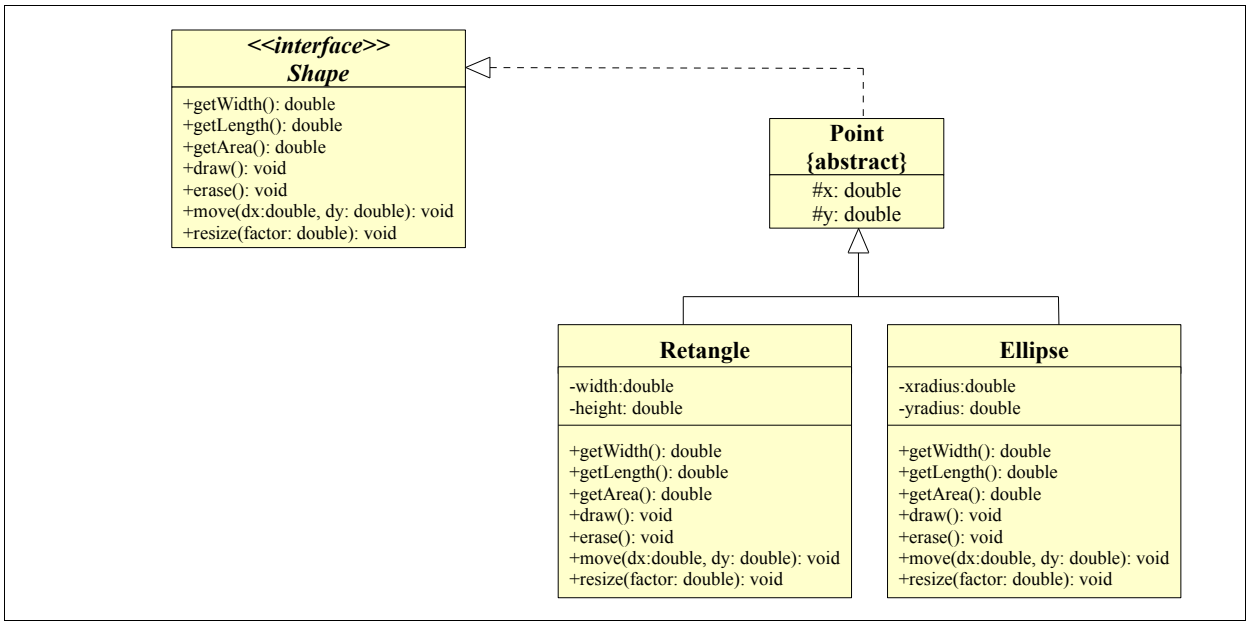
```
public class A {
    private int value = 0;
    public void compute(int x) {
        value += x;
        //Some code that you have to write
    }
    public int getValue() {
        return value;
    }
}
public class B {
    private A theAObject;
    public B(A anAObject) {
        theAObject = anAObject;
        //Some code that you have to write
    }
    //Some method you have to write
}
```

Lösning:

```
public class A extends Observable {
    private int value = 0;
    public void compute(int x) {
        value += x;
        setChanged();
        notifyObservers();
    }
    public int getValue() {
        return value;
    }
}
public class B implements Observer {
    private A theAObject;
    public B(A anAObject) {
        theAObject = anAObject;
        theAObject.addObserver(this);
    }
    public void update(Observable o, Object arg) {
        System.out.println(theAObject.getValue());
    }
}
```

Uppgift

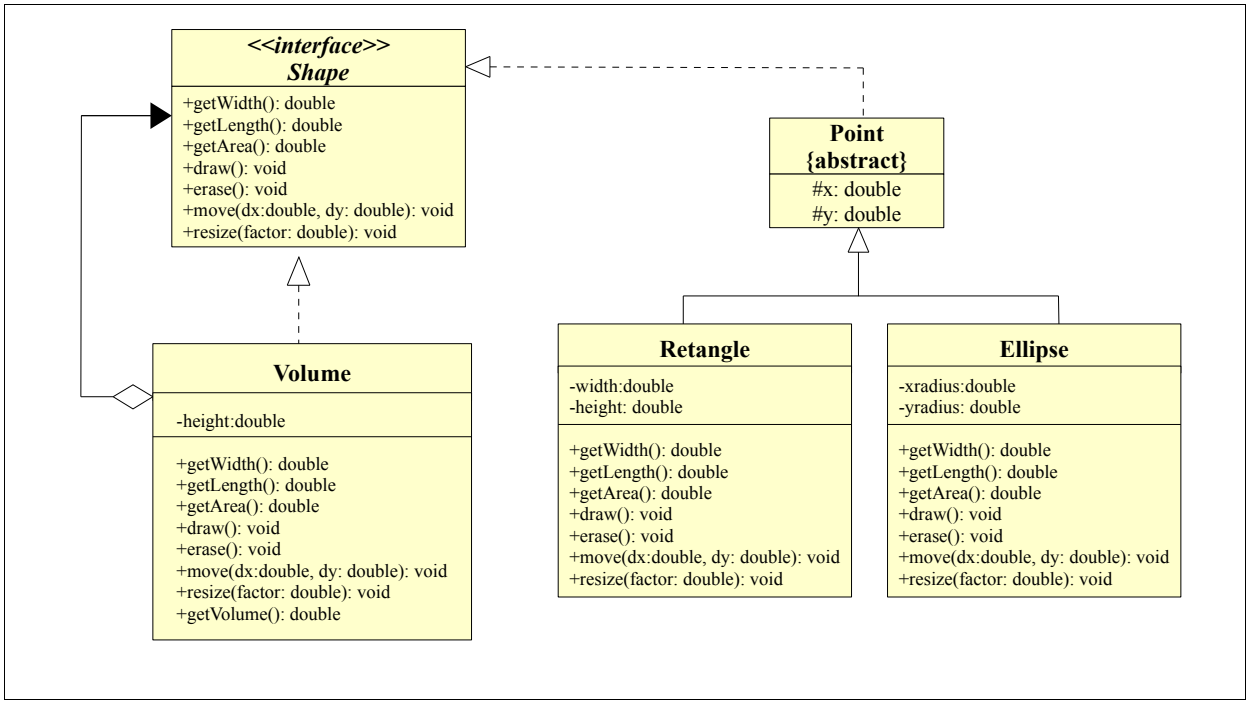
Antag att man har tillgång till följande klasser för att representera geometriska figurer



Klasserna är givna och får inte ändras. Antag att man vill kunna addera en tredje dimension till retangel- och ellipsobjekt så att de får en volym. De skall alltså även ha en höjd. Inför klassen Volume. Klassen skall bl.a ha metoden

public double getVolume()

För att göra detta skall du använda designmönstret Decorator enligt figuren nedan:



a) Implementera klassen Volume i Java.

b) Ge ett kodexempel som skapar en kub och skriver ut dess volym.

Lösning:

a)

```
public class Volume implements Shape {
    private Shape obj;
    private double height;
    public Volume(Shape object, double height) {
        this.obj = obj;
        this.height = height;
    }
    public double getWith() {return obj.getWith(); }
    public double getLength() {return obj.getLength(); }
    public double getArea() {return obj.getArea(); }
    public void draw() { obj.draw(); }
    public void erase() { obj.erase(); }
    public void move(double dx, double dy) { obj.move(dx, dy); }
    public void resize(double factor) {
        height = heigth*factor;
        obj.resize(factor);
    }
    public double getVolume() {
        return obj.getArea()*height;
    }
}
```

b)

```
Volume cub = new Volume( new Rectangle(0 ,0,10, 10), 10);
System.out.println(cube.getVolume());
```