Violation of Liskov

abstract class Bird {

    abstract void fly() ;

}

class Eagle extends Bird {

    @Override

    public void fly() {

        sout(" Eagles fly");

    }

}

class Ostrich  extends Bird {

    @Override

    public void fly() { // dummy implentation

        sout("cant fly high but It lays big egg");

    }

}

class Driverclass{

    psvm(String[] args){

    }

}

Code:

abstract class Bird {  
 abstract void layEggs();  
}  
  
interface Flyable {  
 void fly();  
}  
  
class Eagle extends Bird implements Flyable {  
 @Override  
 public void fly() {  
 System.*out*.println("Eagles fly high");  
 }  
  
 @Override  
 void layEggs() {  
 System.*out*.println("Eagle lays eggs");  
 }  
}  
  
class Ostrich extends Bird {  
 @Override  
 void layEggs() {  
 System.*out*.println("Ostrich lays big eggs");  
 }  
}  
public class Driverclass {  
 public static void main(String[] args) {  
 Bird eagle = new Eagle();  
 Bird ostrich = new Ostrich();  
  
 eagle.layEggs();  
 ostrich.layEggs();  
  
 Flyable flyingBird = new Eagle(); // works  
 flyingBird.fly();  
  
 // Flyable nonFlyingBird = new Ostrich(); // compile error — and that’s GOOD!  
 }  
}

output:

Eagle lays eggs

Ostrich lays big eggs

Eagles fly high

Process finished with exit code 0

Task 02:

Implementation of Liskov

abstract class BirdsthatFly {

    abstract void fly() ;

}

abstract class BirdsthatDontFly {

    abstract void Speciality() ;

}

class Eagle extends BirdsthatFly {

    @Override

    public void fly() {

        sout(" Eagles fly");

    }

}

class Ostrich  extends BirdsthatDontFly {

    @Override

    public void Speciality() {

        sout("It lays big egg");

    }

}

class Driverclass{

    psvm(String[] args){

    }

}

Code:

// Abstract base class for birds that can fly  
abstract class BirdsThatFly {  
 abstract void fly();  
}  
  
// Abstract base class for birds that can't fly  
abstract class BirdsThatDontFly {  
 abstract void speciality();  
}  
  
// Concrete subclass representing a flying bird  
class Eagle extends BirdsThatFly {  
 @Override  
 public void fly() {  
 System.*out*.println("Eagles fly high");  
 }  
}  
  
// Concrete subclass representing a non-flying bird  
class Ostrich extends BirdsThatDontFly {  
 @Override  
 public void speciality() {  
 System.*out*.println("Ostrich lays big eggs");  
 }  
}  
  
// Driver class to test functionality  
class DriverClass {  
 public static void main(String[] args) {  
 BirdsThatFly eagle = new Eagle();  
 eagle.fly();  
  
 BirdsThatDontFly ostrich = new Ostrich();  
 ostrich.speciality();  
 }  
}

Output:

Eagles fly high

Ostrich lays big eggs

Process finished with exit code 0

Task 03:

Violation of Interface Segregation principle

interface ICalcShapesArea {

    calcArea();

    calcVolume();

}

class Circle implements ICalcShapesArea {

    calcArea() { sout()};

    calcVolume() { sout()}; // dummy implentation as it been forced

}

class Sphere implements ICalcShapesArea {

    calcArea() { sout()};

    calcVolume() { sout()};

}

class Driverclass {

    psvm(  ,...  ) {

    }

}

Code:

package July26;  
  
interface IArea {  
 void calcArea();  
}  
  
interface IVolume {  
 void calcVolume();  
}  
class Circle implements IArea {  
 @Override  
 public void calcArea() {  
 System.*out*.println("Calculating area of a circle");  
 }  
}  
  
class Sphere implements IArea, IVolume {  
 @Override  
 public void calcArea() {  
 System.*out*.println("Calculating surface area of a sphere");  
 }  
  
 @Override  
 public void calcVolume() {  
 System.*out*.println("Calculating volume of a sphere");  
 }  
}  
public class DriverClass1 {  
 public static void main(String[] args) {  
 IArea circle = new Circle();  
 circle.calcArea();  
  
 Sphere sphere = new Sphere();  
 sphere.calcArea();  
 sphere.calcVolume();  
 }  
}

output:

Calculating area of a circle

Calculating surface area of a sphere

Calculating volume of a sphere

Process finished with exit code 0

Task 04:

Implementation of Interface Segregation Principle

interface ICalcArea {

    calcArea(); calcPerimeter();

}

interface ICalcVolume {

    calcVolume();

}

class Circle implements ICalcArea {

    @Override

    calcArea() { sout()};

}

class Sphere implements ICalcArea, ICalcVolume {

     @Override

     calcArea() { sout()};

     calcVolume() { sout()};

}

class Driverclass {

    psvm(  ,...  ) {

    }

}

Code:

interface ICalcArea {

void calcArea();

void calcPerimeter();

}

interface ICalcVolume {

void calcVolume();

}

class Circle implements ICalcArea {

@Override

public void calcArea() {

System.out.println("Calculating area of a circle");

}

@Override

public void calcPerimeter() {

System.out.println("Calculating perimeter of a circle");

}

}

class Sphere implements ICalcArea, ICalcVolume {

@Override

public void calcArea() {

System.out.println("Calculating surface area of a sphere");

}

@Override

public void calcPerimeter() {

System.out.println("No perimeter for 3D shapes");

}

@Override

public void calcVolume() {

System.out.println("Calculating volume of a sphere");

}

}

public class DriverClass {

public static void main(String[] args) {

ICalcArea circle = new Circle();

circle.calcArea();

circle.calcPerimeter();

Sphere sphere = new Sphere();

sphere.calcArea();

sphere.calcPerimeter();

sphere.calcVolume();

}

}

Output:

Calculating area of a circle

Calculating perimeter of a circle

Calculating surface area of a sphere

No perimeter for 3D shapes

Calculating volume of a sphere

Process finished with exit code 0

Task5:

public class Clothes {

    void seeRating() {

    }

    void viewSample() {

    }

}

public class Cupboard { // high level class

// completly depending on low level class

//called clothes, books, vessels etc..

    Clothes cobj;

    void addClothes(Clothes cobj) {

    }

    void CustomizeClothes() {

    }

}

customer asks to add books in the code:

public class Books {

    void seeRating() {

    }

    void readSample() {

    }

}

Now DIP implementation:

Implementing Dependency Inversion Principle

public interface IProduct {

    void SeeReviews();

    void getSample();

}

public class Clothes implements IProduct {

    @Override

    public void SeeReviews() {

    }

    @Override

    public void getSample() {

    }

}

public class Books implements IProduct {

    @Override

    public void SeeReviews() {

    }

    @Override

    public void getSample() {

    }

}

Code:

// Abstraction

public interface IProduct {

void seeReviews();

void getSample();

}

// Low-level modules implementing the abstraction

public class Clothes implements IProduct {

@Override

public void seeReviews() {

System.out.println("Clothes rating: 4.5 stars");

}

@Override

public void getSample() {

System.out.println("Showing fabric swatch");

}

}

public class Books implements IProduct {

@Override

public void seeReviews() {

System.out.println("Books rating: 4.8 stars");

}

@Override

public void getSample() {

System.out.println("Showing first 5 pages");

}

}

public class Cupboard {

IProduct product; // depends on abstraction, not concrete class

public void addProduct(IProduct product) {

this.product = product;

}

public void customizeProduct() {

product.seeReviews();

product.getSample();

}

}

public class DriverClass {

public static void main(String[] args) {

Cupboard cupboard1 = new Cupboard();

cupboard1.addProduct(new Clothes());

cupboard1.customizeProduct();

System.out.println("---");

Cupboard cupboard2 = new Cupboard();

cupboard2.addProduct(new Books());

cupboard2.customizeProduct();

}

}

Output:

Clothes rating: 4.5 stars

Showing fabric swatch

---

Books rating: 4.8 stars

Showing first 5 pages

Process finished with exit code 0