학번: \_\_\_\_\_2016112158\_\_\_\_\_\_\_ 이름: \_\_\_김희수\_\_\_\_\_\_\_\_\_



객체지향 설계와 패턴

**LAB ASSIGNMENT #3: 객체지향 설계 원리(SOLID)의 이해**



**마감: 2021. 3. 28. 자정**

문제 #1: 다음 AreaCalculator 프로그램을 작성하는 과정이다. 처음 Rectangle 모양에 대한 정의와 면적 계산 기능을 작성하였다.

public class Rectangle{

public double length;

public double width;

}

public class AreaCalculator{

public double calculateRectangleArea(Rectangle rectangle){ return rectangle.length \*rectangle.width;

}

}

Circle 모양을 추가하기 위하여 다음과 같이 AreaCalulator를 수정하였다.

public class Circle{

public double radius;

}

public class AreaCalculator{

public double calculateRectangleArea(Rectangle rectangle){ return rectangle.length \*rectangle.width;

}

public double calculateCircleArea(Circle circle){

return (22/7)\*circle.radius\*circle.radius;

}

}

1. 위 프로그램이 어떤 문제점이 있는지 설명하라.

Circle이 추가되었다고 AreaCalculator클래스에 새로운 메소드가 추가되었다. 다른 도형(삼각형)이 추가된다면 또 AreaCalculator를 수정해야한다.

1. 객체지향 설계의 어떤 원리를 위배한 것인가?

개방폐쇄의 원리 위배

1. 위 프로그램을 (2)의 원리를 적용하여 수정하라.

공통적인 기능인 면적 계산 작업을 인터페이스로 분리하여 Calculable인터페이스를 만든 후,

Calculable인터페이스를 구현한 RectAreaCalculator, CircleAreaCalculator클래스를 정의했다. 각 Cacluator클래스는 클래스에 따라 Rectangle이나 Circle을 멤버로 사용해서 calculateArea()를 다르게 정의할 수 있다. 만약 삼각형을 추가한다해도 Calculable 인터페이스는 변경되지 않을 것이다.

public interface Calculable {  
 public double calculateArea();  
}

public class RectAreaCalculator implements Calculable {  
 private Rectangle rect;  
  
 public RectAreaCalculator(Rectangle rect) {  
 this.rect = rect;  
 }  
  
 @Override  
 public double calculateArea() {  
 return rect.length \* rect.width;  
 }  
}

public class CircleAreaCalculator implements Calculable {  
 private Circle circle;  
  
 public CircleAreaCalculator(Circle circle) {  
 this.circle = circle;  
 }  
  
 @Override  
 public double calculateArea() {  
 return (22/7) \* circle.radius \* circle.radius;  
 }  
}

객체지향 설계와 패턴 *Page 1! of 6!* 동국대학교

학번: \_\_\_\_\_\_2016112158\_\_\_\_\_\_\_ 이름: \_\_\_\_\_\_\_김희수\_\_\_\_\_\_

문제 #2:

다음과 같은 스스로 인쇄가능한 Text 클래스가 있다.

class Text {

String text;

String author;

int length;

String getText() { ... }

void setText(String s) { ... }

String getAuthor() { ... }

void setAuthor(String s) { ... }

int getLength() { ... }

void setLength(int k) { ... }

/\*methods that change the text\*/

void allLettersToUpperCase() { ... }

void findSubTextAndDelete(String s) { ... }

/\*method for formatting output\*/

void printText() { ... }

}

1. 위 Text 클래스의 문제점은 무엇인가? 또한 어떤 객체설계 원리를 위배한 것인가?

너무 많은 기능을 Text클래스 혼자서 담당하고 있다. 이는 단일책임의 원칙을 위배한것이라 생각된다

1. 위 문제를 해결하기 위한 Java프로그램을 작성하라.

public class Text {  
 String text;  
 String author;  
 int length;  
  
 public String getText() {*//...}* public void setText(String text) {...}  
  
 public String getAuthor() {*//... }* public void setAuthor(String author) {... }  
  
 public int getLength() {... }  
  
 public void setLength(int length) {...}  
   
}

public class TextChanger {  
 void allLettersToUpper(Text txt) {...}  
 void findSubTextAndDelete(Text txt, String s){...}  
}  
  
public class Printer{  
 void printText(Text txt){...}  
}

객체지향 설계와 패턴 *Page 2! of 6!* 동국대학교

학번: \_\_\_\_\_\_\_\_\_2016112158\_\_\_\_\_\_\_ 이름: \_\_\_\_\_김희수\_\_\_\_\_\_\_

문제 #3:

다음은 전등을 켜고 끄는 Power switch클래스 프로그램이다.

public class LightBulb {

public void turnOn() {

System.out.println("LightBulb: Bulb turned on...");

}

public void turnOff() {

System.out.println("LightBulb: Bulb turned off...");

}

}

public class ElectricPowerSwitch {

public LightBulb lightBulb;

public boolean on;

public ElectricPowerSwitch(LightBulb lightBulb) {

this.lightBulb = lightBulb;

this.on = false;

}

public boolean isOn() {

return this.on;

}

public void press(){

boolean checkOn = isOn();

if (checkOn) {

lightBulb.turnOff();

this.on = false;

} else { lightBulb.turnOn(); this.on = true;

}

}

}

1. 위 프로그램의 설계에 어떤 문제점이 있는가?

ElecticPowerSwitch클래스가 의존관계를 맺을 때 구체적인 클래스(LightBulb)에 의존하고 있다. 이는 의존관계 역전의 원리에 위배된다

1. 위 프로그램을 개선하여 다시 작성하라. LightBulb 이외에 Fan 같은 클래스가 추가되었을 때 어떻게 되는지 보여라.

turnOn, turnOff의 공통된 기능을 가진 것들을 묶은 Switchable 인터페이스를 정의한 후, LightBulb같은 구체 클래스가 아닌 인터페이스에 의존하게 EletricPowerSwitch 클래스를 정의한다

public interface Switchable {  
 public void turnOn();  
 public void turnOff();  
}

public class LightBulb implements Switchable {  
 @Override  
 public void turnOn() {  
 System.*out*.println("LightBulb turn on");  
 }  
  
 @Override  
 public void turnOff() {  
 System.*out*.println("LightBulb turn off");  
 }  
}

public class Fan implements Switchable {  
 @Override  
 public void turnOn() {  
 System.*out*.println("Fan turn on");  
 }  
  
 @Override  
 public void turnOff() {  
 System.*out*.println("Fan turn off");  
 }  
}

public class ElectricPowerSwitch {  
 public Switchable switchable;  
 public boolean on;  
  
 public ElectricPowerSwitch(Switchable switchable) {  
 this.switchable = switchable;  
 this.on = false;  
 }  
  
 public boolean isOn() {  
 return this.on;  
 }  
  
 public void press() {  
 boolean checkOn = isOn();  
 if (checkOn) {  
 switchable.turnOff();  
 this.on = false;  
 } else {  
 switchable.turnOn();  
 this.on = true;  
 }  
 }  
}

객체지향 설계와 패턴 *Page 3! of 6!* 동국대학교

학번: \_\_\_\_\_\_\_\_2016112158\_\_\_\_\_\_\_\_ 이름: \_\_\_\_\_김희수\_\_\_\_\_\_\_\_

문제 #4

여러 가지 Toy를 구축하는 프로그램을 작성한다고 하자. 각 toy는 가격(price)과 색상(color)이 있다. toy car와 toy train은 toy 에 move 기능을 추가하여 만들고 toy plane은 move 기능과 fly를 추가하여 만든다. Toy의 동작을 정의하는 인터페이스를 다음 과 같이 정의하였다.

public interface Toy {

void setPrice(double price);

void setColor(String color);

void move();

void fly();

}

이렇게 설계하면 만일 Toyhouse가 필요하다면 다음과 같이 move 기능과 fly 기능이 없더라도 구현하여야 한다.

public class ToyHouse implements Toy {

double price;

String color;

@Override

public void setPrice(double price) {

this.price = price;

}

@Override

public void setColor(String color) {

this.color=color;

}

@Override

public void move(){}

@Override

public void fly(){}

}

1. 위와 같은 문제점이 생기게 되는 원인은 무엇인가? 어떤 설계 원리를 위배하여 생기는 것인가?

ToyHouse클래스에는 필요없는 move, fly메소드가 Toy 인터페이스에 선언되어있기 때문. 이는 인터페이스 격리의 원리에 위배된다.

1. 위에서 지적한 문제점을 해결하여 Toy House, Toy Car, Toy Plane을 구현하라.

ToyHouse, ToyCar, ToyPlane 은 모두 Toy의 한 종류들이다(is a kind of 관계). 즉, 이 세개의 공통된 멤버와 메소드(price와 color, setPrice, setColor)를 가지고 있는 Toy클래스를 정의한 후, Toy를 베이스로 하는 ToyHouse,Car,Plane클래스를 정의할 것이다. ToyCar에는 move()메소드를 추가할 것이고, ToyPlane에는 fly()메소드를 추가할 것이다.

public class Toy {  
 double price;  
 String color;  
  
 void setPrice(double price){  
 this.price = price;  
 }  
 void setColor(String color){  
 this.color = color;  
 }  
}

public class ToyCar extends Toy {  
 public void move() {}  
  
}

public class ToyHouse extends Toy {  
  
}

public class ToyPlane extends Toy {  
 public void fly(){}  
}

객체지향 설계와 패턴 *Page 4! of 6!* 동국대학교

학번: \_\_\_\_\_2016112158\_\_\_\_\_\_\_\_\_ 이름: \_\_\_\_\_\_\_\_김희수\_\_\_\_\_\_\_

문제 #5

다음 프로그램을 읽고 물음에 답하라.

public class Rectangle {

private int height;

private int width;

public Rectangle(int height, int width) {

this.height = height;

this.width = width;

}

public int getHeight() {

return this.height;

}

public void setHeight(int height) {

this.height = height;

}

public int getWidth() {

return this.width;

}

public void setWidth(int width) {

this.width = width;

}

}

public class Square extends Rectangle {

public Square(int size) {

super(size, size);

}

~~public int getHeight() {~~

~~return super.height;~~

~~}~~

~~public int getWidth() {~~

~~return super.width;~~

~~}~~

}

1. 위 프로그램이 다음과 같은 코드로 실행되었을 때 결과와 문제점은?

Rectangle r = new Square(3);

r.setHeight(5);

r.setWidth(6);

텍스트이(가) 표시된 사진

자동 생성된 설명

문제가 너무 많다. 일단 이 문제를 낸 의도는 SOLID원칙중에서 어떤 원리를 위배했는지를 물어보는거 같은데, 당장 주어진 코드를 실행하면 실행되지 않는다. 왜냐하면 Rectangle r = new Square();에서 파라미터가 없기 때문에 컴파일 에러가 발생한다. 이 문제를 해결할거면 r = new Square(3)로 바꾸거나 디폴트 생성자가 선언해야한다. 두번째로 Square 클래스 정의할 때 getHeight(), getWidth()에서 베이스의 private 멤버 height, width를 바로 super로 가져온다는건 불가능하다. Protected로 선언한다면 가능하겠지만 주어진 코드론 불가능하다. 또한, 베이스클래스인 Rectangle에 이미 getHeight(), getWeight()메소드가 존재하는데 굳이 파생클래스에서 또 한번 getHeight, getWeight()를 정의할 필요가 있을까?

위 코드를 실행시켜보면 Square객체의 세로=5, 가로=6이 되어 정사각형이 아니게 된다.

1. 위와 같은 문제가 발생한 이유는 무엇인가? 어떤 설계 원리를 위배하였는가?

Square클래스가 정사각형을 의도한다면 height와 width는 항상 동시에 설정되어야지 위와 같이 별개로 설정되선 안된다. 정사각형의 height와 width가 달라지는 경우가 생겨선 안된다. 이는 리스코프 교체의 원리에 위배된다.

1. 위 프로그램을 올바로 고치시오.

Setter에 의해서 문제가 발생하므로 setter를 없애서 문제를 해결할 수 있다.

public class Rectangle {  
 private int height;  
  
 private int width;  
  
 public Rectangle(int height, int width) {  
 this.height = height;  
 this.width = width;  
 }  
  
 public int getArea() {  
 return width \* height;  
 }  
  
 public int getHeight() { return this.height;}  
 public int getWidth() {  
 return this.width;  
 }

@Override

public String toString() {  
 return "h : "+height+", w : "+width;  
 }

}

public class Square extends Rectangle{  
  
 public Square(int length) {  
 super(length, length);  
 }  
}

public class main {  
 public static void main(String[] args) {  
 Rectangle r = new Square(3);  
  
 System.*out*.println(r.toString());

}  
}

또는 상속대신 composition을 이용해서 문제를 해결할 수도 있다.

public interface Shape {  
 int getArea();  
}

public class Rectangle implements Shape{  
 private int height;  
  
 private int width;  
  
 public Rectangle(int height, int width) {  
 this.height = height;  
 this.width = width;  
 }  
  
 public int getArea() {  
 return width \* height;  
 }  
  
 public int getHeight() { return this.height;}  
 public int getWidth() {  
 return this.width;  
 }

@Override

public String toString() {  
 return "h : "+height+", w : "+width;  
 }

}

public class Square implements Shape{  
 private Rectangle rect;  
 public Square(int length) {  
 setLength(length);  
 }  
  
 public void setLength(int length) {  
 this.rect = new Rectangle(length, length);  
 }  
  
 public int getArea() {  
 return rect.getArea();  
 }

@Override

public String toString() {  
 return "h : "+height+", w : "+width;  
 }

}

public class main {  
 public static void main(String[] args) {  
 Rectangle r = new Square(3);  
  
 System.*out*.println(r.toString());  
 }  
}

객체지향 설계와 패턴 *Page 5! of 6!* 동국대학교

학번: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 이름: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**문제 6. 인사 관리 소프트웨어 시스템은 새로운 종류의 휴가 수당을 추가해야 한다. 기능을 지원하려면 원본 코드를 크게 수정해야 하였다면 이것은 어떤 SOLID 설계 원칙을 위반한 것일까? 그 이유는?**

**개방폐쇄의 원리 위반. 수당이란 인터페이스를 선언하고 “새로운 휴가 수당”이 수당 인터페이스를 구현했다면 원본코드를 크게 수정할 필요없이 확장가능 할 것이다. 즉, 주어진 시스템은 확장엔 닫혀있고 변경에 열려 있다.**

**문제 7. 파생 클래스는 UnuspportedOperationException을 throw하여 재정의 된 메서드를 구현하였다. 이것은 어떤 SOLID 설계 원칙을 위반한 것인가? 그 이유를 설명하라.**

**사용하지 않을 메소드를 구현하였다. 이는 인터페이스 격리의 원리에 위배된다**

**문제 8. 클래스의 메서드에 선택적 경로가 많으면 테스트하기가 어렵다. 이것은 어떤 SOLID 설계 원칙을 위반한 것일까? 그 이유는?**

**개방폐쇄의 원리 위반. 선택적 경로가 많다는 건 새로운 경로를 추가할 때마다 코드를 수정해야한다는 뜻이다. 즉 변경에 닫혀있지 않다**

**문제 9. 다음 코드 조각은 어떤 SOLID 원칙을 위반한 것일까? 그 이유는?**

**public interface I {**

**void drink();**

**void eat();**

**void move();**

**void rent();**

**void block();**

**void run();**

**void purchase();**

**void packForTrip();**

**};**

**하나의 인터페이스에 너무 많은 책임을 갖고 있어서 이 인터페이스를 구현한 클래스는 저 메소드를 전부 오버라이딩 해야할 것이다. 즉, 사용하지 않을 메소드에 의존하는일이 생길 수 있다. 이는 인터페이스 격리의 원리를 위반한 것이다.**

**문제 10. 다음 코드를 DIP 원칙을 따르도록 수정하라.**

public interface Driveable {  
 void accelerate();  
  
 void steerRight();  
  
 void steerLeft();  
  
 void steerStraight();  
  
 void brake();  
}

public class SelfDrivingCar implements Driveable{  
 @Override  
 public void accelerate() { }  
  
 @Override  
 public void steerRight() { }  
  
 @Override  
 public void steerLeft() { }  
  
 @Override  
 public void steerStraight() { }  
  
 @Override  
 public void brake() { }  
}

public class Automobile {  
 Driveable dr;  
 public Automobile(Driveable dr) {  
 this.dr = dr;  
 }  
  
 public void accelerate() {  
 dr.accelerate();  
 }  
  
 public void steerRight() {  
 dr.steerRight();  
 }  
  
 public void steerLeft() {  
 dr.steerLeft();  
 }  
  
 public void steerStraight() {  
 dr.steerStraight();  
 }  
  
 public void brake() {  
 dr.brake();  
 }  
}

**제출물**

제출하여야 할 것:

1. 모든답은빈칸에쓸것
2. UML 다이어그램은 StarUML을 이용하여 그린 후 복사할 것
3. 실행 결과는 이클립스 실행 결과를 캡처해도 좋음.

**평가**

과제 #3은 다음과 같은 기준으로 평가할 것임:

1. 원시코드의 정확성
2. 설계의 타당성
3. 설계, 구현 결과물에서 볼 수 있는 설계에 대한 이해 수준

객체지향 설계와 패턴 *Page 6! of 6!* 동국대학교