دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

درس آز مهندسی نرم‌افزار

استاد سیدحسن میریان حسین‌آبادی

گزارش آزمایش اول

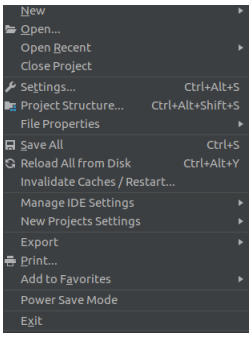
حسین ذاکری‌نیا، مصطفی قدیمی

سؤال ۱. نوشتن تست

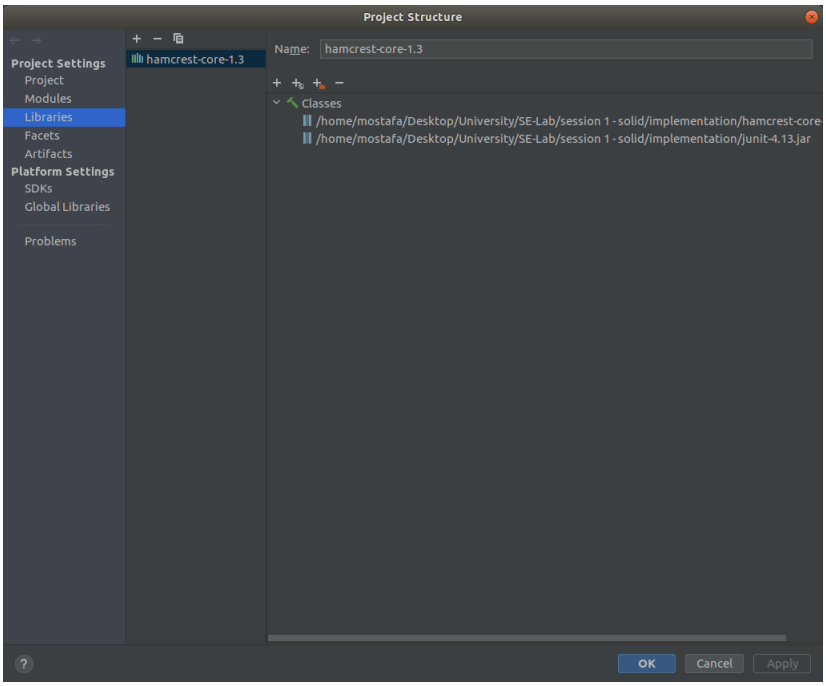
برای نوشتن تست در زبان برنامه‌نویسی جاوا، از junit استفاده می‌کنیم. برای راه‌اندازی و استفاده از آن در محیط برنامه‌نویسی نرم‌افزار Intellij باید به صورت زیر عمل کنیم:

۱. دو فایل hamcrest و junit با فرمت .jar را دانلود می‌کنیم.

۲. از زبانه‌ی File گزینه‌ی Project Structure را انتخاب می‌کنیم.



۳. در قسمت libraries در منوی ظاهر شده، دو فایل دانلود شده را اضافه می‌کنیم.



حال تست‌ها را شروع به نوشتن می‌کنیم. دو عملیات برای اشکال مربع و مستطیل مد نظرمان می‌باشد:

- محاسبه مساحت
- تغییر اضلاع

برای هر کدام از شکل‌ها باید تستی متناظر با آن را بنویسیم و مقداری که از خروجی آن انتظار می‌رود را امتحان کنیم. لذا مقدار مورد انتظار را با استفاده از متد assertEquals از کلاس Assert به کار می‌بریم.

تست مستطیل

```
public class RectangleJUnit {
    @Test
    public void computeArea(){
        Rectangle rectangle = new Rectangle(2, 3);
        double area = rectangle.computeArea();
        Assert.assertEquals(6.0, area, ;(0.0
    }

    @Test
    public void changeWidth() {
        Rectangle rectangle = new Rectangle(3, 4);
        rectangle.setWidth(5);
        Assert.assertEquals(5, rectangle.getWidth(), ;(0.0
    }

    @Test
    public void changeHeight() {
        Rectangle rectangle = new Rectangle(3, 4);
        rectangle.setHeight(6);
        Assert.assertEquals(6, rectangle.getHeight(), ;(0.0
    }
}
```

تست مربع

```
public class SquareJUnit {
        @Test
        public void computeArea(){
                Shape square = new Square(5);
                double area = square.computeArea();
                Assert.assertEquals(25.0, area, ;(0.0
        }

        @Test
        public void changeParameter() {
                Square square = new Square(2);
                square.setSide(4);
                Assert.assertEquals(4.0, square.getSide(), ;(0.0
        }
}
```

سؤال ۲. اضافه کردن کد اصلی برنامه

با توجه به این‌که هر دو شکل مربع و مستطیل یک کارایی را برای محاسبه مساحت در نظر دارند؛ بنابراین، یک Interface را در نظر گرفتیم که این متد رو به صورت Abstract دارد. اگر هر کدام را به عنوان پدر دیگری انتخاب می‌کردیم، از لحاظ اصول اساسی شی‌گرا مشکل پیش می‌آمد و برای برقرار اصل DIP این عمل را انجام داده‌ایم.

کد Shape

با توجه به توضیحات داده شده، یک Interface با نام Shape در نظر گرفته‌ایم که تنها متد آن computeArea است.

```
public interface Shape {
    double computeArea();
}
```

کد Rectangle

```
public class Rectangle implements Shape {
    private double width;
    private double height;

    public void setWidth(double width) {
        this.width = width;
    }

    public void setHeight(double height) {
        this.height = height;
    }

    public double getWidth() {
        return width;
    }

    public double getHeight() {
        return height;
    }

    public Rectangle(double width, double height) {
        this.setWidth(width);
        this.setHeight(height);
    }

    public double computeArea(){
        return this.getWidth() * this.getHeight();
    }

}
```

کد Square

```
public class Square implements Shape {
    private double side;
```

```
    public double getSide() {
        return side;
    }

    public void setSide(double side) {
        this.side = side;
    }

    public Square(double side){
        this.setSide(side);
    }

    public double computeArea() {
        return this.getSide() * this.getSide();
    }

}
```

سؤال ۳. رعایت اصول SOLID

- **Single Responsibility Principle**: این اصل برقرار است. زیرا هر کدام از کلاس‌های مستطیل و مربع فقط به یک کنش‌گر جواب‌گو هستند؛ به عبارت دیگر، از هر کدام از متدها فقط یک کنش‌گر استفاده می‌کند.
- **Open Closed Principle**: با توجه به این‌که هر دو کلاس مستطیل و مربع نسبت به تغییرات در منطقشان بسته هستند و در صورتی که بنا به دلایلی در آینده به ارث بخواهند برسند، امکان اضافه کردن قابلیت به آن‌ها وجود دارد.
- **Liskov Substitution Principle**: اگر در این قسمت Interface را به عنوان کلاس پدر در نظر بگیریم، هر دو کلاس مربع و مستطیل قابلیت جای‌نشینی را دارند. زیرا preconditionها قوی‌تر نشده است و هم‌چنین postconditionهای آن ضعیف‌تر نشده است.
- **Interface Segregation Principle**: به دلیل این‌که در این قسمت فقط یک Interface داریم، این اصل نیز رعایت می‌شود.
- **Dependency Inversion Principle**: همان‌طور که قبلا توضیح داده شد، به دلیل اشتراکاتی که کلاس مربع و کلاس مستطیل دارند، به جای ارث‌بری از یک‌دیگر، از یک واسط برای آن‌ها استفاده کردیم و این اصل رعایت شد.