به نام خدا

گزارش آزمایش پنجم

محمد حیدری و محمد حسین قیصریه

ابتدا کلاس Rectangle را پیاده‌سازی می‌کنیم:

```
public class Rectangle {

    4 usages
    private double width;
    4 usages
    private double height;

    3 usages
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    6 usages
    public double computeArea() {
        return width * height;
    }
```

به همراه Getter و Setter ها:

```java
public double getWidth() {
    return width;
}

3 usages
public double getHeight() {
    return height;
}

5 usages  1 override
public void setWidth(double width) {
    this.width = width;
}

5 usages  1 override
public void setHeight(double height) {
    this.height = height;
}
```

حال یک تست می‌نویسیم که درستی این محاسبات را بررسی کند:

```java
@Test
public void areaTest() {
    Rectangle rectangle = new Rectangle( width: 10,  height: 12);
    Assert.assertEquals( expected: 120, rectangle.computeArea(),  delta: 0);

    Random random = new Random();
    for (int i = 0; i < 10; i++) {
        double width = random.nextDouble();
        double height = random.nextDouble();
        rectangle.setHeight(height);
        rectangle.setWidth(width);
        Assert.assertEquals( expected: width * height, rectangle.computeArea(),  delta: 0);
    }
}
```

تست بالا درستی محاسبات مربوط به مساحت را انجام می‌دهد و تست پایین تست مربوط به مقدار دهی ها را بررسی می کند.

```
@Test
public void getterSetterTest(){
    Rectangle rectangle = new Rectangle( width: 0,  height: 0);

    Random random = new Random();
    for (int i = 0; i < 10; i++) {
        double width = random.nextDouble();
        rectangle.setWidth(width);
        Assert.assertEquals(width, rectangle.getWidth(), delta: 0);

        double height = random.nextDouble();
        rectangle.setHeight(height);
        Assert.assertEquals(height, rectangle.getHeight(),  delta: 0);
    }
}
```

حال می‌خواهیم کلاس Square را بسازیم که از کلاس Rectangle ارث‌بری می کند:

```
public class Square extends Rectangle {
    2 usages
    public Square(double side) {
        super(side, side);
    }


}
```

و بلافاصله برای آن تست می‌نویسیم:

```java
@Test
public void areaTest() {
    Square square = new Square( side: 12);
    Assert.assertEquals( expected: 144, square.computeArea(),  delta: 0);

    Random random = new Random();
    for (int i = 0; i < 10; i++) {
        double height = random.nextDouble();
        square.setHeight(height);
        Assert.assertEquals( expected: height * height, square.computeArea(),  delta: 0);

        double width = random.nextDouble();
        square.setWidth(width);
        Assert.assertEquals( expected: width * width, square.computeArea(),  delta: 0);

    }
}
```

حال باید این تست ها را اجرا کنیم و ببینیم چه مقدار از آن ها پاس می‌شوند. طبق انتظارمان همه fail می‌شوند و باید به پیاده سازی Square برگردیم و شروع به پیاده‌سازی کنیم به نحوی که تست های مورد انتظار را نهایتا پاس کند.

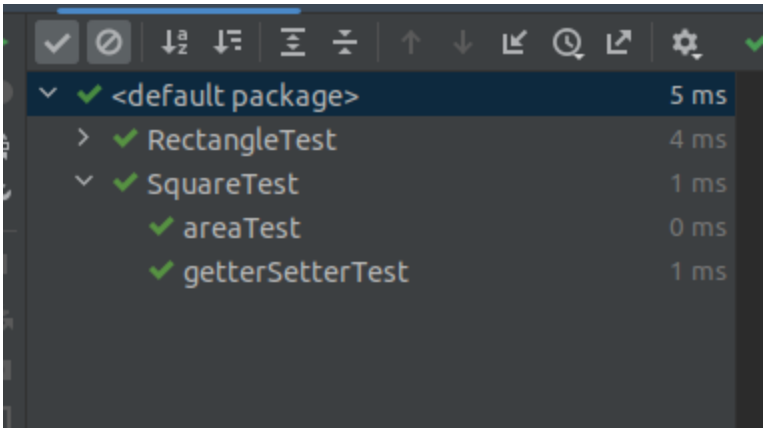
پیاده سازی تمام می‌شود:

```java
public class Square extends Rectangle {
    2 usages
    public Square(double side) {
        super(side, side);
    }

    4 usages
    public void setSideLength(double size) {
        super.setWidth(size);
        super.setHeight(size);
    }

    1 usage
    public double getSideLength(){
        return this.getHeight();
    }

    5 usages
    @Override
    public void setWidth(double width) {
        this.setSideLength(width);
    }

    5 usages
    @Override
    public void setHeight(double height) {
        this.setSideLength(height);
    }
}
```

حال تمامی تست ها را مجددا اجرا می‌کنیم و مطمئن می‌شویم که همگی پاس می‌شوند. از طرفی تعدادی تست هم برای بررسی صحت مقدار دهی ابعاد مربع اضافه می‌کنیم.

```
@Test
public void getterSetterTest() {
    Square square = new Square( side: 0);

    Random random = new Random();
    for (int i = 0; i < 10; i++) {
        double width = random.nextDouble();
        square.setWidth(width);
        checkSideSize(square, width);

        double height = random.nextDouble();
        square.setHeight(height);
        checkSideSize(square, height);

        double side = random.nextDouble();
        square.setSideLength(side);
        checkSideSize(square, side);
    }
}

3 usages
public void checkSideSize(Square square, double expectedSize){
    Assert.assertEquals(expectedSize, square.getSideLength(),  delta: 0);
    Assert.assertEquals(expectedSize, square.getWidth(),  delta: 0);
    Assert.assertEquals(expectedSize, square.getHeight(),  delta: 0);
}
```

نتیجه اجرای تست ها:


<default package> 5 ms
RectangleTest 4 ms
SquareTest 1 ms
areaTest 0 ms
getterSetterTest 1 ms

پرسش‌ها

- هر یک از پنج اصل SOLID را در دو الی سه خط توضیح دهید.

مفهوم اصول SOLID در مهندسی نرم افزار به منظور طراحی و توسعه نرم‌افزارهای انعطاف‌پذیر، قابل‌تغییر و قابل‌نگهداری به کار می‌رود. پنج اصل SOLID عبارتند از:

۱. SRP (Single Responsibility Principle): هر کلاس باید فقط یک مسئولیت داشته باشد و تغییرات در آن باید به یک دلیل واحد باشند.

۲. OCP (Open/Closed Principle): کلاس‌ها باید باز برای توسعه و بسته برای تغییر باشند. بدین معنی که باید بتوانیم بدون تغییر کد موجود، کلاس را گسترش دهیم.

۳. LSP (Liskov Substitution Principle): باید بتوانیم هر شیء مشتق شده از یک کلاس را به جای آن کلاس در برنامه استفاده کنیم بدون اینکه عملکرد برنامه تغییر کند.

۴. ISP (Interface Segregation Principle): باید برای هر کاربرد، یک رابط کاربری خاص و مشخص داشته باشیم و هیچ کلاسی نباید به چیزی نیاز داشته باشد که برایش لازم نیست.

۵. DIP (Dependency Inversion Principle): باید برای کاهش وابستگی‌ها، به جای وابستگی به کلاس‌های پایین‌تر، به کلاس‌هایی که به آن‌ها وابستگی کمتری دارند، وابستگی داشت.

- اصول SOLID در کدام یک از گام‌های اصلی ایجاد نرم‌افزار (تحلیل نیازمندی‌ها، طراحی، پیاده‌سازی، تست و استقرار) استفاده می‌شوند؟ توضیح دهید.

اصول SOLID در مهندسی نرم‌افزار در مرحله طراحی و پیاده‌سازی استفاده می‌شوند.

در مرحله طراحی، اصول SOLID به منظور طراحی سیستم‌های انعطاف‌پذیر، قابل‌تغییر و قابل‌نگهداری به کار می‌روند. در این مرحله، طراحی کلاس‌ها، رابط‌ها و ارتباطات بین آنها بر اساس اصول SOLID انجام می‌شود.

- در مرحله پیاده‌سازی، اصول SOLID به منظور پیاده‌سازی کدی که با توجه به اصول طراحی شده استفاده می‌شوند. با رعایت اصول SOLID، کدی با انعطاف‌پذیری بالا، قابل‌تغییر و قابل‌نگهداری تولید می‌شود.

هیچ تناقضی بین روش TDD و تست پس از پیاده‌سازی وجود ندارد، به طوری که هر دو این روش‌ها بهبود کیفیت نرم‌افزار را هدف می‌گیرند. در واقع تست نوشتن در روش TDD قبل از پیاده‌سازی به تست‌های ابتدایی که برای برنامه نویسی مشخص شده‌اند، کمک می کند تا تیم مهندسی نرم‌افزار، نیازمندی‌های مشتری را درک بهتری داشته و همچنین با انجام تست‌های اولیه، برخی از مشکلات پیشگیری و درمان شوند.

بنابراین، انجام تست‌ها پس از پیاده‌سازی نیز برای بالا بردن کیفیت نرم‌افزار لازم است. به این ترتیب، تست در هر دو روش TDD و تست پس از پیاده‌سازی برای بهبود کیفیت و عملکرد نرم‌افزار بسیار مفید است.

- فرض کنید در آزمایش بالا نیازی به تغییر ابعاد مستطیل نداشتیم. در این حالت طراحی مدل‌ها چه تفاوتی می کند؟

بهترین کاری که می‌توانستیم انجام دهیم این بود که تمام field ها را final کنیم و مساحت را همان جا در ابتدای کار محاسبه کنیم و در یک field مخصوص قرار دهیم و در صورت صدا شدن تابع مساحت آن مقدار را پس بدهیم.