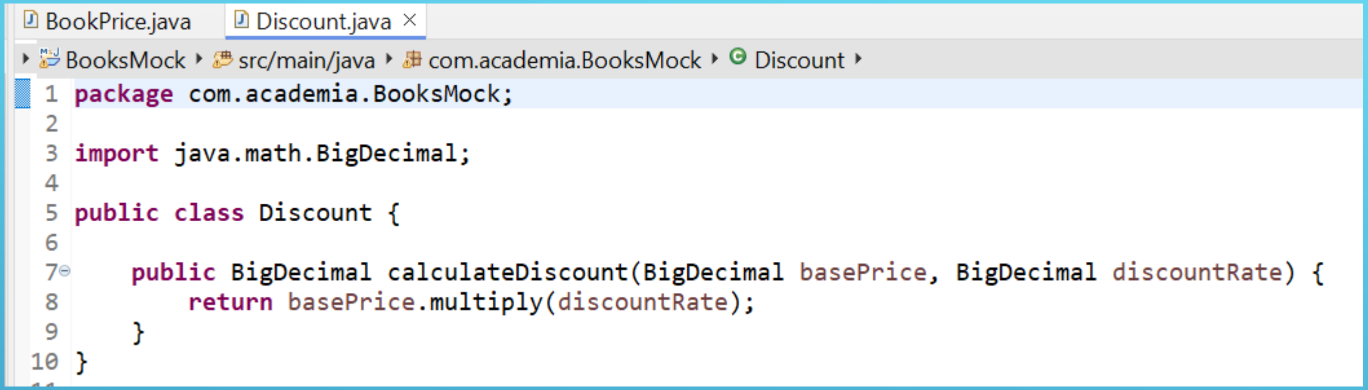
BOOKSMOCK

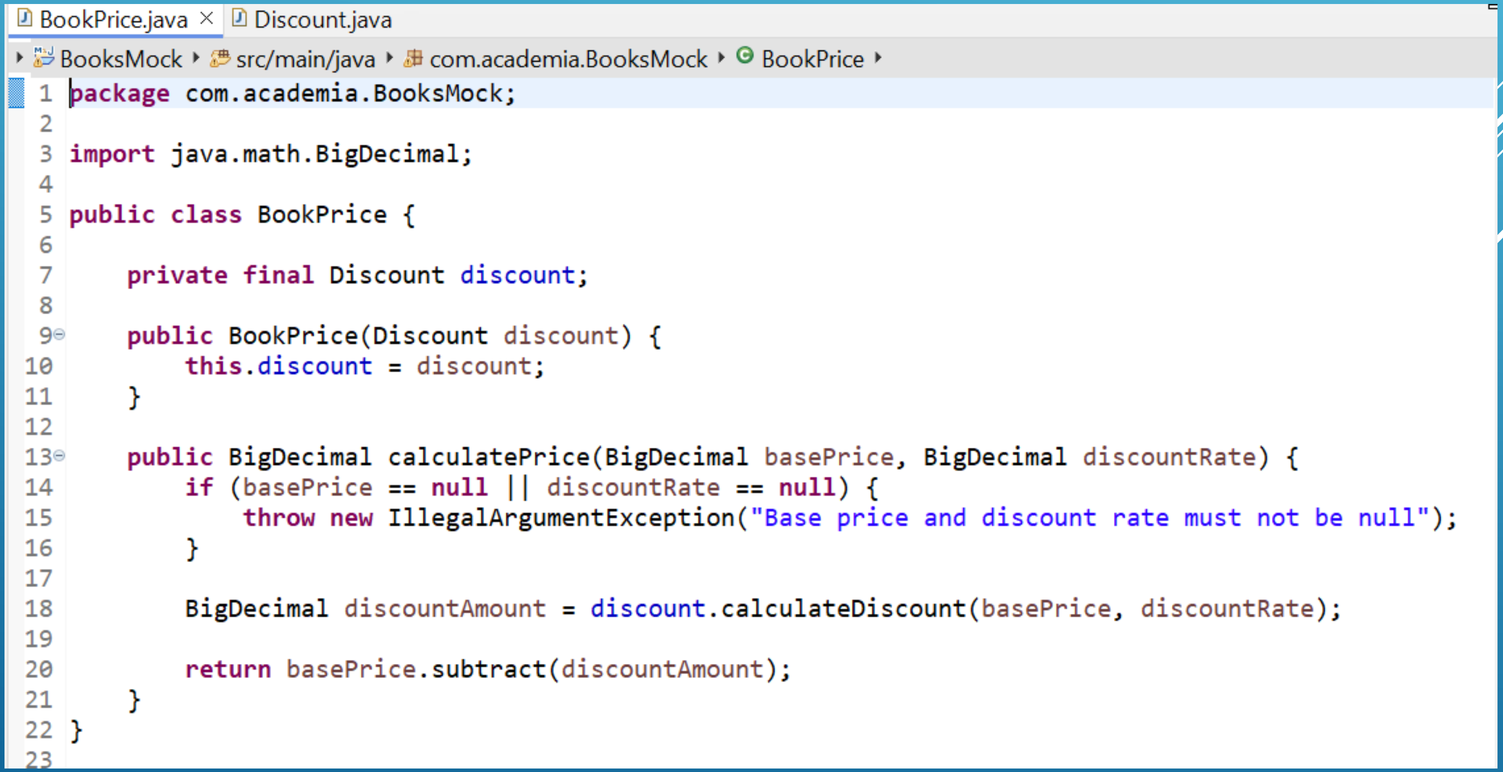
Ever Hinojosa Aguirre


BookPrice.java | Discount.java

BooksMock ▸ src/main/java ▸ com.academia.BooksMock ▸ Discount ▸

```java
package com.academia.BooksMock;

import java.math.BigDecimal;

public class Discount {

    public BigDecimal calculateDiscount(BigDecimal basePrice, BigDecimal discountRate) {
        return basePrice.multiply(discountRate);
    }
}
```

- Simple app that calculates discounts to showcase the use of mockito to test it. It takes it a Price and its discount to calculate the Price after.


BookPrice.java | Discount.java

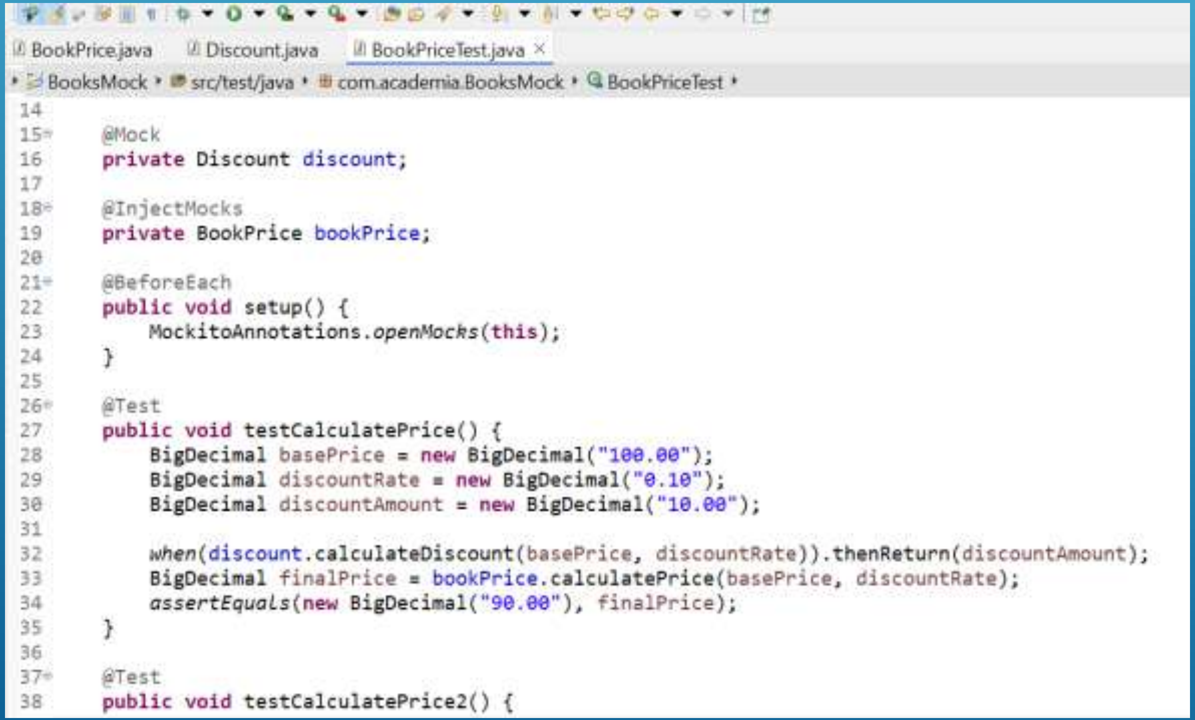
BooksMock ▸ src/main/java ▸ com.academia.BooksMock ▸ BookPrice ▸

```java
package com.academia.BooksMock;

import java.math.BigDecimal;

public class BookPrice {

    private final Discount discount;

    public BookPrice(Discount discount) {
        this.discount = discount;
    }

    public BigDecimal calculatePrice(BigDecimal basePrice, BigDecimal discountRate) {
        if (basePrice == null || discountRate == null) {
            throw new IllegalArgumentException("Base price and discount rate must not be null");
        }

        BigDecimal discountAmount = discount.calculateDiscount(basePrice, discountRate);

        return basePrice.subtract(discountAmount);
    }
}
```

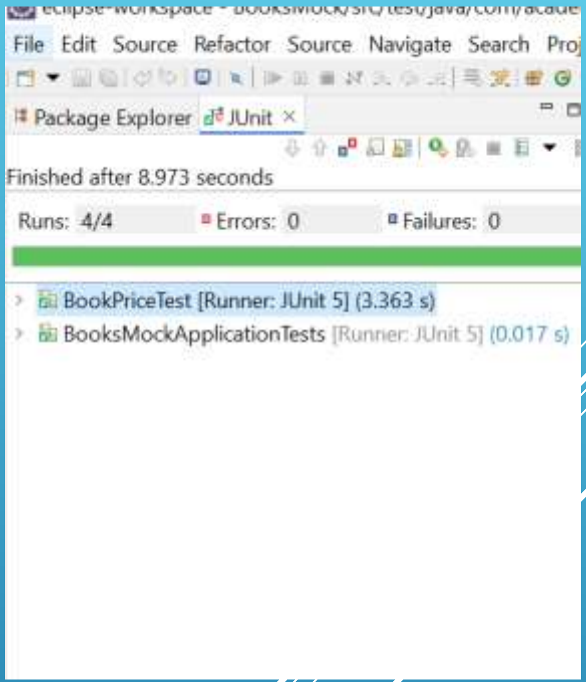
DISCOUNT

- BY “mocking” the discount I can simulate its behavior to ensure that the BookPrice class functions correctly without relying on the actual implementation of the Discount class. This allows me to control the output of the Discount methods and validate the logic in BookPrice independently of Discount.

TEST



```java
    @Mock
    private Discount discount;

    @InjectMocks
    private BookPrice bookPrice;

    @BeforeEach
    public void setup() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    public void testCalculatePrice() {
        BigDecimal basePrice = new BigDecimal("100.00");
        BigDecimal discountRate = new BigDecimal("0.10");
        BigDecimal discountAmount = new BigDecimal("10.00");

        when(discount.calculateDiscount(basePrice, discountRate)).thenReturn(discountAmount);
        BigDecimal finalPrice = bookPrice.calculatePrice(basePrice, discountRate);
        assertEquals(new BigDecimal("90.00"), finalPrice);
    }

    @Test
    public void testCalculatePrice2() {
```



```
Finished after 8.973 seconds

Runs: 4/4        Errors: 0        Failures: 0

> BookPriceTest [Runner: JUnit 5] (3.363 s)
> BooksMockApplicationTests [Runner: JUnit 5] (0.017 s)
```