```
package sqat.swc.neu.shop;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class BasketTest {

    private Product testProduct1 = new Product("test1", 100);
    private Product testProduct2 = new Product("test2", 200);

    @Test
    void testAddProduct() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 1);
        assertEquals(testProduct1, basket.findBasketItemWithProduct(testProduct1).getProduct());
    }

    @Test
    void testAddProduct_Duplicate() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 1);
        basket.addProduct(testProduct1, 2);
        assertEquals(3, basket.findBasketItemWithProduct(testProduct1).getQuantity());
    }

    @Test
    void testAddProduct_NegativeAmount() {
        Basket basket = new Basket();
        assertThrows(IllegalArgumentException.class, () -> basket.addProduct(testProduct1, -111));
    }

    @Test
    void testAddProduct_Null() {
        Basket basket = new Basket();
        assertThrows(NullPointerException.class, () -> {
            basket.addProduct(null, 233);
        });
    }

    @Test
    void testAddProduct_AfterPayment() {
        Basket basket = new Basket();
        basket.pay(0);
        assertFalse(basket.addProduct(testProduct1, 111));
    }

    @Test
    void testRemoveAllProductItems() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 233);
        assertTrue(basket.removeAllProductItems(testProduct1));
        assertEquals(basket.getTotal(), 0);
    }

    @Test
    void testRemoveAllProductItems_AfterPayment() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 123);
        basket.pay(123 * testProduct1.getPrice());
        assertFalse(basket.removeAllProductItems(testProduct1));
    }

    @Test
    void testRemoveAllProductItems_Null() {
        Basket basket = new Basket();
        assertFalse(basket.removeAllProductItems(null));
    }

    @Test
    void testRemoveAllProductItems_NoneExist() {
        Basket basket = new Basket();
        assertFalse(basket.removeAllProductItems(testProduct1));
    }

    @Test
    void testRemoveSomeProductItems() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 123);
        assertTrue(basket.removeSomeProductItems(testProduct1, 23));
        assertEquals(100, basket.findBasketItemWithProduct(testProduct1).getQuantity());
    }

    @Test
    void testRemoveSomeProductItems_Null() {
        Basket basket = new Basket();
        assertFalse(basket.removeSomeProductItems(null, 123));
    }

    @Test
    void testRemoveSomeProductItems_ZeroQuantity() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 10);
        basket.removeSomeProductItems(testProduct1, 10);
        assertNull(basket.findBasketItemWithProduct(testProduct1));
    }

    @Test
    void testRemoveSomeProductItems_OneQuantity() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 10);
        basket.removeSomeProductItems(testProduct1, 1);
        assertEquals(9, basket.findBasketItemWithProduct(testProduct1).getQuantity());
    }
```

```
@Test
void testRemoveSomeProductItems_AfterPayment() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 2333);
    basket.pay(2333 * testProduct1.getPrice());
    assertFalse(basket.removeSomeProductItems(testProduct1, 2333));
    assertEquals(2333, basket.findBasketItemWithProduct(testProduct1).getQuantity());
}

@Test
void testRemoveSomeProductItems_OverQuantity() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 1000);
    assertFalse(basket.removeSomeProductItems(testProduct1, 10000));
}

@Test
void testRemoveSomeProductItems_NegativeAmount() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 123);
    assertThrows(IllegalArgumentException.class, () -> basket.removeSomeProductItems(testProduct1, -123));
}

@Test
void testRemoveSomeProductItems_NoneExist() {
    Basket basket = new Basket();
    assertFalse(basket.removeSomeProductItems(testProduct1, 123));
}

@Test
void testFindBasketItemWithProduct() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 100);
    assertEquals(testProduct1, basket.findBasketItemWithProduct(testProduct1).getProduct());
}

@Test
void testFindBasketItemWithProduct_NoneExist() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 123);
    assertNull(basket.findBasketItemWithProduct(testProduct2));
}

@Test
void testFIndBasketItemWithProduct_Empty() {
    Basket basket = new Basket();
    assertNull(basket.findBasketItemWithProduct(testProduct1));
}

@Test
void testGetTotal() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 10);
    assertEquals(1000, basket.getTotal());
}

@Test
void testGetTotal_WithDiscount() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 10);
    Discount discount = new Discount("test1", testProduct1, 10, 2);
    basket.addDiscount(discount);
    assertEquals(800, basket.getTotal());
}

@Test
void testGetTotal_Empty() {
    Basket basket = new Basket();
    assertEquals(0, basket.getTotal());
}

@Test
void testGetTotal_MoreQuantity() {
    Basket basket = new Basket();
    Discount discount = new Discount("test", testProduct1, 10, 10);
    basket.addDiscount(discount);
    basket.addProduct(testProduct1, 1);
    assertEquals(100, basket.getTotal());
}

@Test
void testGetTotal_NoMathDiscount(){
    Basket basket = new Basket();
    Discount discount = new Discount("test", testProduct2, 10, 1);
    basket.addProduct(testProduct1, 10);
    basket.addDiscount(discount);
    assertEquals(1000, basket.getTotal());
}

@Test
void testGetTotal_LargerDiscountValue(){
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 10);
    Discount discount = new Discount("test", testProduct1, 10, 11);
    basket.addDiscount(discount);
    assertEquals(1000, basket.getTotal());
}
@Test
void testPay() {
    Basket basket = new Basket();
    basket.addProduct(testProduct1, 100);
    PaymentResult result = basket.pay(100 * testProduct1.getPrice() + 1);
    assertTrue(result.isSuccess());
```

```
        assertEquals(1, result.getChange());
    }

    @Test
    void testPay_Duplicate() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 100);
        basket.pay(100 * testProduct1.getPrice());
        PaymentResult result = basket.pay(100 * testProduct1.getPrice());
        assertFalse(result.isSuccess());
        assertEquals("Payment already complete", result.getMessage());
    }

    @Test
    void testPay_InsufficientMoney() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 100);
        PaymentResult result = basket.pay(100 * testProduct1.getPrice() - 1);
        assertFalse(result.isSuccess());
        assertEquals("Insufficient amount.", result.getMessage());
    }

    @Test
    void testAddDiscount() {
        Basket basket = new Basket();
        Discount discount = new Discount("test1", testProduct1, 10, 3);
        basket.addDiscount(discount);
        assertEquals(1, basket.getNumberOfDiscounts());
    }

    @Test
    void testAddDiscountNull() {
        Basket basket = new Basket();
        assertThrows(NullPointerException.class, () -> basket.addDiscount(null));
    }

    @Test
    void testGetNumberOfItems() {
        Basket basket = new Basket();
        basket.addProduct(testProduct1, 100);
        assertEquals(1, basket.getNumberOfItems());
    }

    @Test
    void testGetNumberOfDiscounts() {
        Basket basket = new Basket();
        Discount discount = new Discount("233", testProduct1, 10, 3);
        basket.addDiscount(discount);
        assertEquals(1, basket.getNumberOfDiscounts());
    }
}
```