Projektowanie obiektowe

Lab 5 Testy jednostkowe

Dawid Białka, Piotr Tekielak

1. Podatek

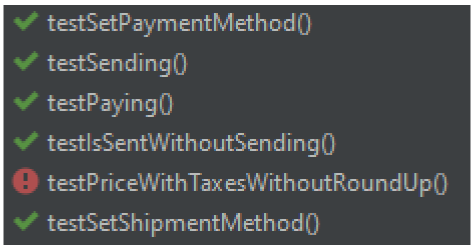
Zmieniamy wartość podatku w zamówieniach z 1.22 na 1.23.

```
public class Order {
    private static final BigDecimal TAX_VALUE = BigDecimal.valueOf(1.23);
   private final UUID id;
    private final Product product;
    private boolean paid;
    private Shipment shipment;
    private ShipmentMethod shipmentMethod;
    private PaymentMethod paymentMethod;
```

W produktach zamieniamy przestarzałe zaokrąglanie z BigDecimal na RoundingMode.

```
public static final int PRICE_PRECISION = 2;
public static final RoundingMode ROUND_STRATEGY = RoundingMode.HALF_UP;

   private final String name;
   private final BigDecimal price;

   public Product(String name, BigDecimal price) {
       this.name = name;
       this.price = price.setScale(PRICE_PRECISION, ROUND_STRATEGY);
   }
```

Po zmianie wartości podatku jeden test nie działa poprawnie.



```
✓ testSetPaymentMethod()
✓ testSending()
✓ testPaying()
✓ testIsSentWithoutSending()
! testPriceWithTaxesWithoutRoundUp()
✓ testSetShipmentMethod()
```

Zmieniamy w nim wartość 2.44 na 2.46.

```
@Test
public void testPriceWithTaxesWithoutRoundUp() {
   // given

   // when
   Order order = getOrderWithCertainProductPrice(2); // 2 PLN

   // then
   assertBigDecimalCompareValue(order.getPriceWithTaxes(),
BigDecimal.valueOf(2.46)); // 2.46 PLN
}
```

Poprawiamy literówkę w słowie Successful.

```
public void send() {
    boolean sentSuccessful = getShipmentMethod().send(shipment,
shipment.getSenderAddress(), shipment.getRecipientAddress());
    shipment.setShipped(sentSuccessful);
}
```

2. Więcej niż jeden produkt w zamówieniu

Zmieniamy pojedynczy produkt w zamówieniu na listę produktów.

```
private final LinkedList<Product> products;

public Order(LinkedList<Product> products) {
    this.products = products;
    id = UUID.randomUUID();
    paid = false;
}

public LinkedList<Product> getProducts() {
    return products;
}

public BigDecimal getPrice() {
    BigDecimal price = BigDecimal.valueOf(0);
    for(Product product : products)
        price = price.add(product.getPrice());
    return price;
}
```

Zmieniamy metodę w testach, aby tworzyła ona listę produktów.

```
private Order getOrderWithMockedProduct() {
   Product product = mock(Product.class);
   LinkedList<Product> products = new LinkedList<>();
   products.add(product);
   return new Order(products);
}
```

Tak samo zmieniamy inne testy i metody.

```java
@Test
public void testGetProductThroughOrder() {
   // given
   Product expectedProduct = mock(Product.class);
   LinkedList<Product> products = new LinkedList<>();
   products.add(expectedProduct);
   Order order = new Order(products);

   // when
   Product actualProduct = order.getProducts().get(0);

   // then
   assertSame(expectedProduct, actualProduct);
}
@Test
public void testGetPrice() throws Exception {
   // given
   BigDecimal expectedProductPrice = BigDecimal.valueOf(1000);
   Product product = mock(Product.class);
   given(product.getPrice()).willReturn(expectedProductPrice);
   LinkedList<Product> products = new LinkedList<>();
   products.add(product);
   Order order = new Order(products);

   // when
   BigDecimal actualProductPrice = order.getPrice();

   // then
   assertBigDecimalCompareValue(expectedProductPrice, actualProductPrice);
}
private Order getOrderWithCertainProductPrice(double productPriceValue) {
   BigDecimal productPrice = BigDecimal.valueOf(productPriceValue);
   Product product = mock(Product.class);
   given(product.getPrice()).willReturn(productPrice);
   LinkedList<Product> products = new LinkedList<>();
   products.add(product);
   return new Order(products);
}
```

Dodajemy nowy test, w którym tworzymy zamówienie z dwoma produktami.

```java
@Test
public void testTwoProductsOneOrder() {
   // given
   Product product1 = mock(Product.class);
   Product product2 = mock(Product.class);
   LinkedList<Product> products = new LinkedList<>();
   products.add(product1);
   products.add(product2);
   Order order = new Order(products);
   // when

   // then
   assertFalse(order.isPaid());
   assertEquals(2, order.getProducts().size());
}
```

Nowy test weryfikujący cenę zamówienia z dwoma produktami.

```
@Test
public void testTwoProductsOneOrderPrice() {
   // given
   BigDecimal productPrice = BigDecimal.valueOf(1000);
   Product product1 = mock(Product.class);
   given(product1.getPrice()).willReturn(productPrice);
   Product product2 = mock(Product.class);
   given(product2.getPrice()).willReturn(productPrice);
   LinkedList<Product> products = new LinkedList<>();
   products.add(product1);
   products.add(product2);
   Order order = new Order(products);

   // when

   // then
   assertEquals(BigDecimal.valueOf(2000), order.getPrice());
}
```

Nowy test weryfikujący cenę zamówienia z dwoma produktami razem z podatkiem.

```
@Test
public void testTwoProductsOneOrderPriceWithTaxes() {
   // given
   BigDecimal productPrice = BigDecimal.valueOf(1000);
   Product product1 = mock(Product.class);
   given(product1.getPrice()).willReturn(productPrice);
   Product product2 = mock(Product.class);
   given(product2.getPrice()).willReturn(productPrice);
   LinkedList<Product> products = new LinkedList<>();
   products.add(product1);
   products.add(product2);
   Order order = new Order(products);

   // when

   // then
   assertBigDecimalCompareValue(BigDecimal.valueOf(2000*1.23),
order.getPriceWithTaxes());
}
```

3. Zniżki na produktach i zamówieniach

Tworzymy nową klasę reprezentującą zniżki.

```
public class Discount {

    private final BigDecimal value;

    public Discount(BigDecimal value) {
        this.value = value;
    }
```

```java
    public BigDecimal getValue() {
        return value;
    }
}
```

Nowy atrybut w klasie Product i jego metody.

```java
private Discount productDiscount;

public BigDecimal getPriceWithDiscount() {
    return
price.multiply(BigDecimal.ONE.subtract(productDiscount.getValue())).setScale(PRICE
_PRECISION, ROUND_STRATEGY);
}

public Discount getProductDiscount() {
    return productDiscount;
}

public void setProductDiscount(Discount productDiscount) {
    this.productDiscount = productDiscount;
}
```

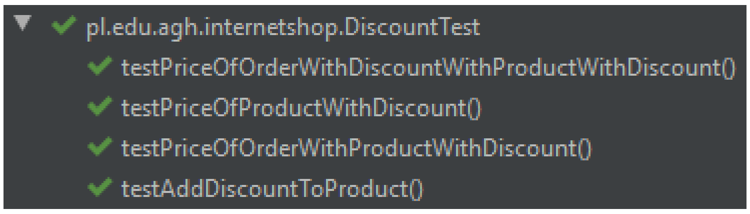
Nowy atrybut w klasie Order i nowe metody.

```java
private Discount orderDiscount;

public BigDecimal getPriceWithDiscounts() {
    BigDecimal price = BigDecimal.valueOf(0);
    for(Product product : products)
        price = price.add(product.getPriceWithDiscount());
    return
price.multiply(BigDecimal.ONE.subtract(orderDiscount.getValue())).setScale(2,
RoundingMode.HALF_UP);
}

public void setOrderDiscount(Discount orderDiscount) {
    this.orderDiscount = orderDiscount;
}
```

Tworzymy klasę z testami zniżek.

```java
public class DiscountTest {

    @Test
    public void testAddDiscountToProduct() {
        Discount discount = new Discount(BigDecimal.valueOf(0.15));
        Product product = new Product("Prod1", BigDecimal.valueOf(2));
        product.setProductDiscount(discount);
        assertEquals(product.getProductDiscount(), discount);
        assertEquals(product.getProductDiscount().getValue(),
discount.getValue());
    }
    @Test
    public void testPriceOfProductWithDiscount() {
        Discount discount = new Discount(BigDecimal.valueOf(0.15));
        Product product = new Product("Prod1", BigDecimal.valueOf(100));
        product.setProductDiscount(discount);
        assertEquals(product.getPriceWithDiscount(),
BigDecimal.valueOf(100).multiply(BigDecimal.valueOf(0.85)));
    }

    @Test
    public void testPriceOfOrderWithProductWithDiscount() {
        Discount discount = new Discount(BigDecimal.valueOf(0.15));
        Product product = new Product("Prod1", BigDecimal.valueOf(100));
        product.setProductDiscount(discount);
        LinkedList<Product> products = new LinkedList<>();
        products.add(product);
        Order order = new Order(products);
        assertEquals(order.getPriceWithDiscounts(),
BigDecimal.valueOf(100).multiply(BigDecimal.valueOf(0.85)));
    }

    @Test
    public void testPriceOfOrderWithDiscountWithProductWithDiscount() {
        Discount discount = new Discount(BigDecimal.valueOf(0.15));
        Product product = new Product("Prod1", BigDecimal.valueOf(100));
        product.setProductDiscount(discount);
        LinkedList<Product> products = new LinkedList<>();
        products.add(product);
        Order order = new Order(products);
        order.setOrderDiscount(discount);
        assertEquals(order.getPriceWithDiscounts(),
BigDecimal.valueOf(100).multiply(BigDecimal.valueOf(0.85)).multiply(BigDecimal.val
ueOf(0.85)).setScale(2));
    }
}
```

Wszystkie przebiegają pomyślnie.



```
▼ ✓ pl.edu.agh.internetshop.DiscountTest
    ✓ testPriceOfOrderWithDiscountWithProductWithDiscount()
    ✓ testPriceOfProductWithDiscount()
    ✓ testPriceOfOrderWithProductWithDiscount()
    ✓ testAddDiscountToProduct()
```

4. Historia zamówień i wyszukiwanie danego zamówienia

Do wyszukiwania po nazwie produktu, nazwisku zamawiającego, wartości zamówienia lub kombinacji tych warunków wykorzystamy strategię szukającą (SearchStrategy).

Tworzymy interfejs, który będzie implementowany przez poszczególne strategie wyszukiwań.

```
public interface SearchStrategy {
    boolean filter(Order order);
}
```

Klasa ProductNameSearchStrategy do wyszukiwania zamówienia, w którym znajduje się produkt o podanej nazwie.

```java
public class ProductNameSearchStrategy implements SearchStrategy {
        private String productName;
        public ProductNameSearchStrategy(String productName) {
            this.productName = productName;
        }

        public boolean filter(Order order) {
            for(Product product: order.getProducts()) {
                if(product.getName().equals(productName)) {
                    return true;
                }
            }
            return false;
        }
}
```

Klasa RecipientNameSearchStrategy do wyszukiwania zamówienia po nazwisku zamawiającego.

```java
public class RecipientNameSearchStrategy implements SearchStrategy{
    private String receiverName;
    public RecipientNameSearchStrategy(String receiverName) {
        this.receiverName = receiverName;
    }

    public boolean filter(Order order) {
        return
order.getShipment().getRecipientAddress().getName().equals(receiverName);
    }
}
```

Klasa OrderValueSearchStrategy do wyszukiwania zamówienia po wartości zamówienia.

```java
public class OrderValueSearchStrategy implements SearchStrategy {
    private BigDecimal orderValue;
    public static final int PRICE_PRECISION = 2;
    public static final RoundingMode ROUND_STRATEGY = RoundingMode.HALF_UP;

    public OrderValueSearchStrategy(BigDecimal orderValue) {
        this.orderValue = orderValue.setScale(PRICE_PRECISION, ROUND_STRATEGY);
    }

    public boolean filter(Order order) {
        return (orderValue.compareTo(order.getPrice()) == 0);
    }
}
```

Klasa CompositeSearchStrategy do wyszukiwania po kombinacji poprzednich strategii wyszukiwani.

```java
public class CompositeSearchStrategy implements SearchStrategy{
    private ArrayList<SearchStrategy> childSearchStrategy = new ArrayList<>();

    public CompositeSearchStrategy(Collection<SearchStrategy> searchStrategies) {
        this.childSearchStrategy.addAll(searchStrategies);
    }

    public boolean filter(Order order) {
        for(SearchStrategy searchStrategy: childSearchStrategy) {
            if(!searchStrategy.filter(order)) {
                return false;
            }
        }
        return true;
    }
}
```

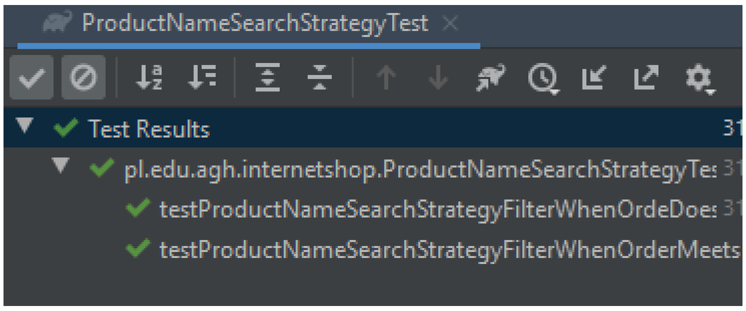
Klasa OrderHistoryDatabase, w której będą przechowywane zamówienia i z której będziemy mogli je pobierać wykorzystując daną strategię wyszukiwania.

```java
public class OrderHistoryDatabase {
    private ArrayList<Order> orders;

    public OrderHistoryDatabase(ArrayList<Order> orders) {
        this.orders = orders;
    }

    public OrderHistoryDatabase() {
        this.orders = new ArrayList<>();
    }

    public Iterator<Order> iterator() {
        return orders.iterator();
    }

    public Iterator<Order> iterator(SearchStrategy searchStrategy) {
        ArrayList<Order> ordersToBeReturned = new ArrayList<>();
        for(Order order: orders) {
            if(searchStrategy.filter(order)) {
                ordersToBeReturned.add(order);
            }
        }
        return ordersToBeReturned.iterator();
    }

    public void addOrder(Order order) {
        orders.add(order);
    }

}
```

Testy dla wyżej wymienionych klas:

Klasa z testami dla ProductNameSearchStrategy.

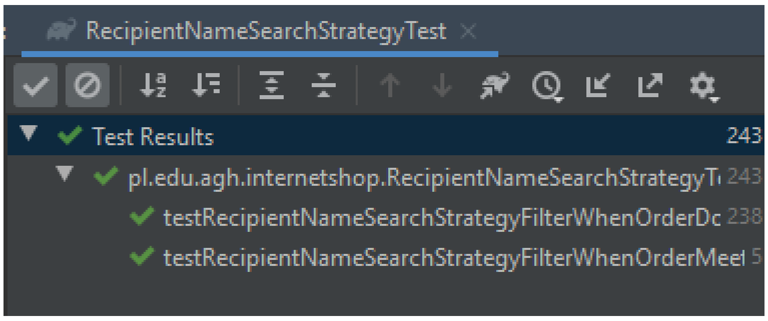
```
public class ProductNameSearchStrategyTest {
    private Product product1;
    private Product product2;
    private LinkedList<Product> products1;
    private LinkedList<Product> products2;
    private Order order1;
    private Order order2;

    private  void initTestComponents()
    {
        String productName1 = "Marchewka";
        String productName2 = "Baklazan";
        product1 = mock(Product.class);
        product2 = mock(Product.class);
        given(product1.getName()).willReturn(productName1);
        given(product2.getName()).willReturn(productName2);
        products1 = new LinkedList<>();
        products1.add(product1);
        products2 = new LinkedList<>();
        products2.add(product2);

        order1 = new Order(products1);
        order2 = new Order(products2);
    }

    @Test
    public void testProductNameSearchStrategyFilterWhenOrderMeetsSearchStrategy()
{
        // given
        initTestComponents();
        SearchStrategy productNameSearchStrategy = new
ProductNameSearchStrategy("Marchewka");
        // when
        boolean containsGivenOrder = productNameSearchStrategy.filter(order1);
        //then
        assertTrue(containsGivenOrder);
    }

    @Test
    public void
testProductNameSearchStrategyFilterWhenOrdeDoesntrMeetsSearchStrategy() {
        // given
        initTestComponents();
        SearchStrategy productNameSearchStrategy = new
ProductNameSearchStrategy("Marchewka");
        // when
        boolean containsGivenOrder = productNameSearchStrategy.filter(order2);
        //then
        assertFalse(containsGivenOrder);
    }
}
```



```
ProductNameSearchStrategyTest
Test Results                                          31
  pl.edu.agh.internetshop.ProductNameSearchStrategyTes 31
    testProductNameSearchStrategyFilterWhenOrdeDoes 31
    testProductNameSearchStrategyFilterWhenOrderMeets
```

Klasa z testami dla RecipientNameSearchStrategy.

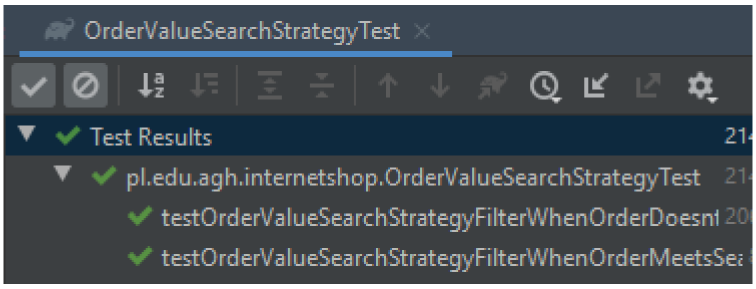
```java
public class RecipientNameSearchStrategyTest {
    private Product product1;
    private LinkedList<Product> products1;
    private Address address1;
    private Shipment shipment1;
    private Order order1;

    private  void initTestComponents()
    {
        String recipientName1 = "Franek";
        String recipientName2 = "Staszek";
        product1 = mock(Product.class);
        address1 = mock(Address.class);
        shipment1 = mock(Shipment.class);
        given(address1.getName()).willReturn(recipientName1);
        given(shipment1.getRecipientAddress()).willReturn(address1);
        products1 = new LinkedList<>();
        products1.add(product1);

        order1 = new Order(products1);
        order1.setShipment(shipment1);
    }

    @Test
    public void
testRecipientNameSearchStrategyFilterWhenOrderMeetsSearchStrategy() {
        // given
        initTestComponents();
        SearchStrategy recipientNameSearchStrategy = new
RecipientNameSearchStrategy("Franek");
        // when
        boolean containsGivenOrder = recipientNameSearchStrategy.filter(order1);
        //then
        assertTrue(containsGivenOrder);
    }

    @Test
    public void
testRecipientNameSearchStrategyFilterWhenOrderDoesntMeetsSearchStrategy() {
        // given
        initTestComponents();
        SearchStrategy recipientNameSearchStrategy = new
RecipientNameSearchStrategy("Staszek");
        // when
        boolean containsGivenOrder = recipientNameSearchStrategy.filter(order1);
        //then
        assertFalse(containsGivenOrder);
    }
}
```


RecipientNameSearchStrategyTest

```
Test Results                                           243
  pl.edu.agh.internetshop.RecipientNameSearchStrategyT 243
    testRecipientNameSearchStrategyFilterWhenOrderDc   238
    testRecipientNameSearchStrategyFilterWhenOrderMeet 5
```

Klasa z testami dla OrderValueSearchStrategy.

```
public class OrderValueSearchStrategyTest {
    private Product product1;
    private Product product2;
    private LinkedList<Product> products1;
    private LinkedList<Product> products2;
    private Order order1;
    private Order order2;

    private  void initTestComponents()
    {
        BigDecimal productPrice1 = BigDecimal.valueOf(5);
        BigDecimal productPrice2 = BigDecimal.valueOf(10);
        product1 = mock(Product.class);
        product2 = mock(Product.class);
        given(product1.getPrice()).willReturn(productPrice1);
        given(product2.getPrice()).willReturn(productPrice2);
        products1 = new LinkedList<>();
        products1.add(product1);
        products2 = new LinkedList<>();
        products2.add(product2);

        order1 = new Order(products1);
        order2 = new Order(products2);
    }

    @Test
    public void testOrderValueSearchStrategyFilterWhenOrderMeetsSearchStrategy() {
        // given
        initTestComponents();
        SearchStrategy orderValueSearchStrategy = new
OrderValueSearchStrategy(BigDecimal.valueOf(5));
        // when
        boolean containsGivenOrder = orderValueSearchStrategy.filter(order1);
        //then
        assertTrue(containsGivenOrder);
    }

    @Test
    public void
testOrderValueSearchStrategyFilterWhenOrderDoesntMeetSearchStrategy() {
        // given
        initTestComponents();
        SearchStrategy orderValueSearchStrategy = new
OrderValueSearchStrategy(BigDecimal.valueOf(5));
        // when
        boolean containsGivenOrder = orderValueSearchStrategy.filter(order2);
        //then
        assertFalse(containsGivenOrder);
    }
}
```

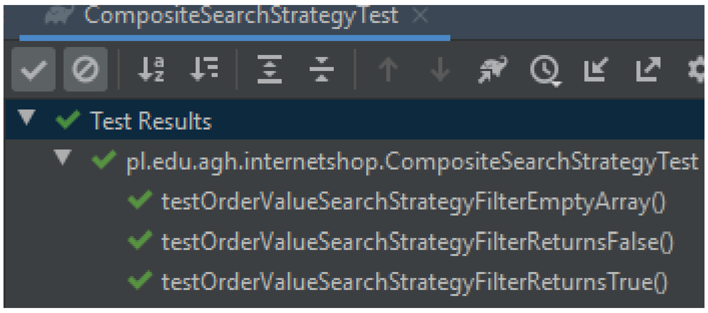


```
OrderValueSearchStrategyTest
Test Results                                                21
  pl.edu.agh.internetshop.OrderValueSearchStrategyTest      21
    testOrderValueSearchStrategyFilterWhenOrderDoesn        20
    testOrderValueSearchStrategyFilterWhenOrderMeetsSe
```

Klasa z testami dla CompositeSearchStrategy.

```
public class CompositeSearchStrategyTest {

    private Product product1;
    private Product product2;
    private LinkedList<Product> products1;
    private LinkedList<Product> products2;
    private Order order1;
    private Order order2;

    private  void initTestComponents()
    {
        String productName1 = "Marchewka";
        String productName2 = "Baklazan";
        BigDecimal productPrice1 = BigDecimal.valueOf(5);
        BigDecimal productPrice2 = BigDecimal.valueOf(10);
        product1 = mock(Product.class);
        product2 = mock(Product.class);
        given(product1.getName()).willReturn(productName1);
        given(product2.getName()).willReturn(productName2);
        given(product1.getPrice()).willReturn(productPrice1);
        given(product2.getPrice()).willReturn(productPrice2);
        products1 = new LinkedList<>();
        products1.add(product1);
        products2 = new LinkedList<>();
        products2.add(product2);

        order1 = new Order(products1);
        order2 = new Order(products2);
    }

    @Test
    public void testOrderValueSearchStrategyFilterReturnsTrue() {
        // given
        initTestComponents();

        ArrayList<SearchStrategy> searchStrategies = new ArrayList<>();
        SearchStrategy productNameSearchStrategy = new
ProductNameSearchStrategy("Marchewka");
        SearchStrategy orderValueSearchStrategy = new
```

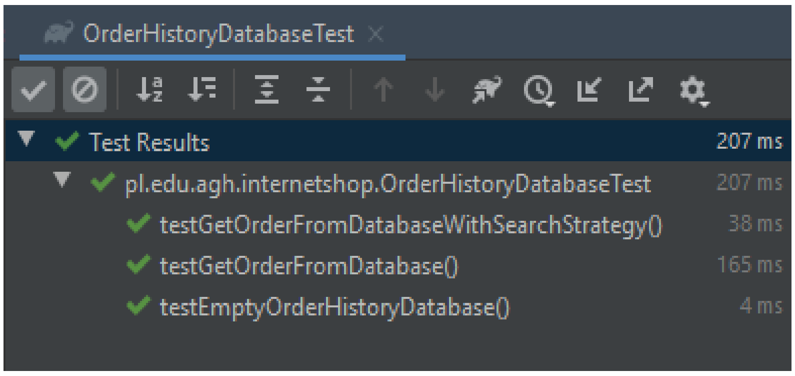
```java
OrderValueSearchStrategy(BigDecimal.valueOf(5));

        searchStrategies.add(productNameSearchStrategy);
        searchStrategies.add(orderValueSearchStrategy);

        SearchStrategy compositeSearchStrategy = new
CompositeSearchStrategy(searchStrategies);
        // when
        boolean containsGivenOrder = compositeSearchStrategy.filter(order1);
        //then
        assertTrue(containsGivenOrder);
    }

    @Test
    public void testOrderValueSearchStrategyFilterReturnsFalse() {
        // given
        initTestComponents();

        ArrayList<SearchStrategy> searchStrategies = new ArrayList<>();
        SearchStrategy productNameSearchStrategy = new
ProductNameSearchStrategy("Marchewka");
        SearchStrategy orderValueSearchStrategy = new
OrderValueSearchStrategy(BigDecimal.valueOf(5));

        searchStrategies.add(productNameSearchStrategy);
        searchStrategies.add(orderValueSearchStrategy);

        SearchStrategy compositeSearchStrategy = new
CompositeSearchStrategy(searchStrategies);
        // when
        boolean containsGivenOrder = compositeSearchStrategy.filter(order2);
        //then
        assertFalse(containsGivenOrder);
    }

    @Test
    public void testOrderValueSearchStrategyFilterEmptyArray() {
        // given
        initTestComponents();

        ArrayList<SearchStrategy> searchStrategies = new ArrayList<>();

        SearchStrategy compositeSearchStrategy = new
CompositeSearchStrategy(searchStrategies);
        // when
        boolean containsGivenOrder = compositeSearchStrategy.filter(order2);
        //then
        assertTrue(containsGivenOrder);
    }
}
```



```
CompositeSearchStrategyTest
Test Results
    pl.edu.agh.internetshop.CompositeSearchStrategyTest
        testOrderValueSearchStrategyFilterEmptyArray()
        testOrderValueSearchStrategyFilterReturnsFalse()
        testOrderValueSearchStrategyFilterReturnsTrue()
```

Klasa z testami dla OrderHistoryDatabase.

```java
public class OrderHistoryDatabaseTest {

    private Order getOrderWithMockedProduct() {
        Product product = mock(Product.class);
        LinkedList<Product> products = new LinkedList<>();
        products.add(product);
        return new Order(products);
    }

    @Test
    public void testEmptyOrderHistoryDatabase() {
        // given
        OrderHistoryDatabase database = new OrderHistoryDatabase();

        // when
        boolean hasNext = database.iterator().hasNext();

        // then
        assertEquals(hasNext, false);
    }

    @Test
    public void testGetOrderFromDatabase() {
        // given
        OrderHistoryDatabase database = new OrderHistoryDatabase();
        Order expectedOrder = getOrderWithMockedProduct();

        // when
        database.addOrder(expectedOrder);

        // then
        assertSame(expectedOrder, database.iterator().next());
    }

    @Test
    public void testGetOrderFromDatabaseWithSearchStrategy() {
        // given
        OrderHistoryDatabase database = new OrderHistoryDatabase();
        Product expectedProduct1 = new Product("Marchewka",
BigDecimal.valueOf(5));
        LinkedList<Product> products1 = new LinkedList<>();
        products1.add(expectedProduct1);

        Product expectedProduct2 = new Product("Baklazan", BigDecimal.valueOf(5));
        LinkedList<Product> products2 = new LinkedList<>();
        products2.add(expectedProduct2);
        Order expectedOrder1 = new Order(products1);
        Order expectedOrder2 = new Order(products2);
        SearchStrategy searchStrategy = new
         ProductNameSearchStrategy(expectedProduct1.getName());

        // when
        database.addOrder(expectedOrder1);
        database.addOrder(expectedOrder2);

        // then
        assertSame(expectedOrder1, database.iterator(searchStrategy).next());
    }
}
```



```
OrderHistoryDatabaseTest
Test Results                                              207 ms
  pl.edu.agh.internetshop.OrderHistoryDatabaseTest        207 ms
    testGetOrderFromDatabaseWithSearchStrategy()           38 ms
    testGetOrderFromDatabase()                            165 ms
    testEmptyOrderHistoryDatabase()                         4 ms
```