Nazar Kordiumov
Radosław Kopeć

Laboratorium 5

Testy jednostkowe

Zadania

1. **Zmienić wartość procentowa naliczanego podatku z 22% na 23%. Należy zweryfikować przypadki brzegowe przy zaokragleniach**
    a. Najpierw zmieniłem testy (jeden test, ponieważ zmiana podatku nie wpłyneła na inne 2 testy)

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

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

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

    b. Najpierw oczywiście test nie przechodził, ale po zmianie podatku, wszystko zadziałało

```
Run:  Tests in 'lab-tests.test'
Tests passed: 24 of 24 tests – 286 ms
Test Results                                   286 ms
  pl.edu.agh.internetshop.AddressTest           34 ms
  pl.edu.agh.internetshop.MoneyTransferTest      7 ms
  pl.edu.agh.internetshop.OrderTest            221 ms
  pl.edu.agh.internetshop.ProductTest            4 ms
  pl.edu.agh.internetshop.ShipmentTest          20 ms
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
WARNING: An illegal reflective access op
WARNING: Illegal reflective access by or
```

```
blic class Order {
  private static final BigDecimal TAX_VALUE = BigDecimal.valueOf(1.23);
  private final UUID id;
```

2. **Rozszerzyć funkcjonalność systemu, tak aby zamówienie mogło obejmować więcej niż jeden produkt na raz**
    a. Stworzyłem kilka testów testujących 1, kilka bądź null jako Product przy tworzeniu obiektu klasy Order

```java
@Test
public void testGetProductThroughOrder() {
    // given
    Product expectedProduct = mock(Product.class);
    Order order = new Order(Collections.singletonList(expectedProduct));

    // when
    List<Product> actualProduct = order.getProducts();

    // then
    assertSame(expectedProduct, actualProduct.get(0));
}

@Test
public void testGetMultipleProductsThroughOrder() {
    // given
    Product expectedProduct1 = mock(Product.class);
    Product expectedProduct2 = mock(Product.class);
    Product expectedProduct3 = mock(Product.class);
    Order order = new Order(Arrays.asList(expectedProduct1, expectedProduct2, expectedProduct3));

    //when
    List<Product> products = order.getProducts();

    //then
    assertEquals( expected: 3, products.size());
    assertEquals(expectedProduct1, products.get(0));
    assertEquals(expectedProduct2, products.get(1));
    assertEquals(expectedProduct3, products.get(2));
}

@Test
public void testSetProductsAsNullInOrder() {
    //given,when,then
    assertThrows(NullPointerException.class, () -> new Order( products: null));
}
```

b. Zmieniłem klasę Order, aby powyższe testy przechodziły

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

    public Order(List<Product> products) {
        this.products = Objects.requireNonNull(products);
        id = UUID.randomUUID();
        paid = false;
    }
```

c. Dodałem metodę, dzięki której możemy stworzyć zamówienie z określoną liczbą produktów o takiej samej cenie

```
private Order getOrderWithCertainNumberOfProductsAndPrice(int numberOfProducts, double productPriceValue) {
    BigDecimal productPrice = BigDecimal.valueOf(productPriceValue);
    List<Product> products = new ArrayList<>();
    IntStream.range(0, numberOfProducts)
            .forEach(i -> {
                Product product = mock(Product.class);
                given(product.getPrice()).willReturn(productPrice);
                products.add(product);
            });

    return new Order(products);
}
```

d. Następnie zmieniłem kilka testy tak, aby sprawdzamy przypadki brzegowe przy zaokrągleniach w cenie całego zamówienia oraz dodałem przypadek testowy, gdy podamy pustą listę produktów (cena -> 0)

```
public void testPriceWithNoProductsInOrder() {
    //given, when
    Order order = new Order(Collections.emptyList()); // 0 PLN

    //then
    assertBigDecimalCompareValue(BigDecimal.valueOf(0), order.getPrice()); // 0 PLN
}

@Test
public void testPriceWithTaxesWithoutRoundUp() {
    // given

    // when
    Order order = getOrderWithCertainNumberOfProductsAndPrice( numberOfProducts: 1, productPriceValue: 2); // 2 PLN

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

@Test
public void testPriceWithTaxesWithRoundDown() {
    // given

    // when
    Order order = getOrderWithCertainNumberOfProductsAndPrice( numberOfProducts: 2, productPriceValue: 0.01); // 0.02 PLN

    // then
    assertBigDecimalCompareValue(order.getPriceWithTaxes(), BigDecimal.valueOf(0.02)); // 0.02 PLN

}

@Test
public void testPriceWithTaxesWithRoundUp() {
    // given

    // when
    Order order = getOrderWithCertainNumberOfProductsAndPrice( numberOfProducts: 4,  productPriceValue: 0.03); // 0.12 PLN

    // then
    assertBigDecimalCompareValue(order.getPriceWithTaxes(), BigDecimal.valueOf(0.15)); // 0.15 PLN

}
```

e. Zaimplementowałem potrzebną do obliczania wartości całego zamówienia funkcjonalność

```
public BigDecimal getPrice() {
    return products.stream()
            .map(Product::getPrice)
            .reduce(BigDecimal.valueOf(0), BigDecimal::add);
}
```

f. Teraz wszystkie testy zaczęły przechodzić

```
Run:  Tests in 'pl.edu.agh.internetshop'
Tests passed: 27 of 27 tests – 284 ms
Test Results                                   284 ms
  pl.edu.agh.internetshop.AddressTest           32 ms
  pl.edu.agh.internetshop.MoneyTransferTest      8 ms
  pl.edu.agh.internetshop.OrderTest            227 ms
  pl.edu.agh.internetshop.ProductTest            4 ms
  pl.edu.agh.internetshop.ShipmentTest          13 ms

> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
WARNING: An illegal reflective access opera
WARNING: Illegal reflective access by org.m
```

3. Dodać możliwość naliczania rabatu do pojedynczego produktu i do ca lego zamówienia

a. Stworzyłem klasę Discount (bez implementacji)

```
public class Discount {

    public static final BigDecimal ONE_HUNDRED = new BigDecimal( val: 100);
    private final BigDecimal discountPercentage;

    public Discount(int discountPercentage) {

        this.discountPercentage = BigDecimal.valueOf(discountPercentage);
    }

    public BigDecimal applyTo(BigDecimal price) {
        return null;
    }
}
```

b. Stworzyłem 5 testów dla każdej klasy równoważności (rabat <0,0,0<rabat<100,100,>100)

```java
@Test
public void testProductPriceWithoutDiscount() throws Exception{
    //given

    // when
    Product product = new Product(NAME, PRICE);

    // then
    assertBigDecimalCompareValue(product.getPrice(), PRICE);
}

@Test
public void testPriceWithFullDiscount() {
    //given

    //when
    Product product = new Product(NAME, PRICE, new Discount( discountPercentage: 100));

    //then
    assertBigDecimalCompareValue(BigDecimal.valueOf(0), product.getDiscountedPrice());
}

@Test
public void testPriceWithTooBigDiscount() {
    assertThrows(IllegalArgumentException.class, () -> new Product(NAME, PRICE, new Discount( discountPercentage: 101)));
}

@Test
public void testPriceWithTooSmallDiscount() {
    assertThrows(IllegalArgumentException.class, () -> new Product(NAME, PRICE, new Discount( discountPercentage: -1)));
}

@Test
public void testPriceWithDiscount() {
    //given

    //when
    Product product = new Product(NAME, PRICE, new Discount( discountPercentage: 50)); // price -> 1 PLN, with discount -> 0.50

    //then
    assertBigDecimalCompareValue(BigDecimal.valueOf(0.5), product.getDiscountedPrice());
}
```

c. Zaimplementowałem potrzebne metody w klasie Product (dodałem pole Discount oraz discountedPrice)

```java
public class Product {

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

    private final String name;
    private final BigDecimal price;
    private final Discount discount;
    private final BigDecimal discountedPrice;

    public Product(String name, BigDecimal price, Discount discount) {
        this.name = name;
        this.discount = discount;
        this.price = price.setScale(PRICE_PRECISION, ROUND_STRATEGY);
        this.discountedPrice = applyDiscountTo(price);
    }

    public Product(String name, BigDecimal price) {
        this.name = name;
        this.price = price;
        this.discount = new Discount( discountPercentage: 0);
        this.discountedPrice = price;
    }

    private BigDecimal applyDiscountTo(BigDecimal price) {
        return discount.applyTo(price);
    }

    public String getName() { return name; }

    public BigDecimal getPrice() {
        return price;
    }

    public BigDecimal getDiscountedPrice() {
        return discountedPrice;
    }
}
```

d. Oraz zaimplementowałem potrzebną funkcjonalność w klasie Discount

```java
private Order getOrderWithCertainNumberOfProductsAndPrice(int numberOfProducts, double productPriceValue) {
    BigDecimal productPrice = BigDecimal.valueOf(productPriceValue);
    List<Product> products = new ArrayList<>();
    IntStream.range(0, numberOfProducts)
            .forEach(i -> {
                Product product = mock(Product.class);
                given(product.getPrice()).willReturn(productPrice);
                products.add(product);
            });

    return new Order(products);
}
```

e. Wszystkie testy przeszły

```
Run:  Tests in 'pl.edu.agh.internetshop'
Tests passed: 31 of 31 tests – 241 ms
Test Results                                   241 ms
  pl.edu.agh.internetshop.AddressTest           36 ms
  pl.edu.agh.internetshop.MoneyTransferTest      4 ms
  pl.edu.agh.internetshop.OrderTest            174 ms
  pl.edu.agh.internetshop.ProductTest           17 ms
  pl.edu.agh.internetshop.ShipmentTest          10 ms

> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
WARNING: An illegal reflective access opera
WARNING: Illegal reflective access by org.
```

f. Dodałem jeszcze jeden test, aby sprawdzić, czy w zamówieniu poprawnie obliczane są ceny wszystkich produktów ze zniżkami

```java
@Test
public void testProductDiscountedPrices() {
    //given
    Product product1 = mock(Product.class);
    given(product1.getDiscountedPrice()).willReturn(BigDecimal.valueOf(2.5));  // 2.5 PLN
    Product product2 = mock(Product.class);
    given(product2.getDiscountedPrice()).willReturn(BigDecimal.valueOf(5)); // 5 PLN

    //when
    Order order = new Order(Arrays.asList(product1, product2));

    //then
    assertBigDecimalCompareValue(BigDecimal.valueOf(7.5), order.getPriceWithProductsDiscount()); // 7.5 PLN
}
```

g. Dodałem testy sprawdzające rabat naliczany do zamówienia

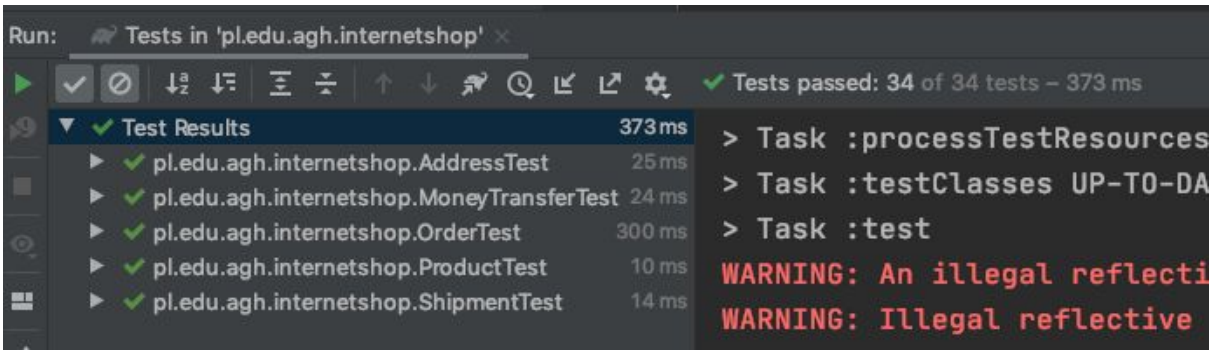
```java
@Test
public void testOrderDiscountWithRoundUp() {
    //given
    Product product1 = mock(Product.class);
    given(product1.getDiscountedPrice()).willReturn(BigDecimal.valueOf(5));  // 5 PLN
    Product product2 = mock(Product.class);
    given(product2.getDiscountedPrice()).willReturn(BigDecimal.valueOf(0.7)); // 0.7 PLN

    //when
    Order order = new Order(Arrays.asList(product1, product2), new Discount( discountPercentage: 5)); // 5.7 PLN * 0.95 = 5.415 = 5.42 PLN

    //then
    assertBigDecimalCompareValue(BigDecimal.valueOf(5.42), order.getPriceWithOrderDiscount());
}

@Test
public void testOrderDiscountWithoutRounding() {
    //given
    Product product1 = mock(Product.class);
    given(product1.getDiscountedPrice()).willReturn(BigDecimal.valueOf(5));  // 5 PLN
    Product product2 = mock(Product.class);
    given(product2.getDiscountedPrice()).willReturn(BigDecimal.valueOf(1)); // 1 PLN

    //when
    Order order = new Order(Arrays.asList(product1, product2), new Discount( discountPercentage: 8)); // 6 PLN * 0.92 = 5.52 PLN

    //then
    assertBigDecimalCompareValue(BigDecimal.valueOf(5.52), order.getPriceWithOrderDiscount());
}
```

h. Zaimplementowałem potrzebną funkcjonalność w klasie Order

```java
public class Order {
    private static final BigDecimal TAX_VALUE = BigDecimal.valueOf(1.23);
    private final UUID id;
    private final List<Product> products;
    private boolean paid;
    private Shipment shipment;
    private ShipmentMethod shipmentMethod;
    private PaymentMethod paymentMethod;
    private final Discount discount;

    public Order(List<Product> products) {
        this.products = Objects.requireNonNull(products);
        id = UUID.randomUUID();
        paid = false;
        this.discount = new Discount( discountPercentage: 0);
    }

    public Order(List<Product> products, Discount discount) {
        this.products = Objects.requireNonNull(products);
        id = UUID.randomUUID();
        paid = false;
        this.discount = discount;
    }

    private BigDecimal applyDiscountTo(BigDecimal price) {
        return discount.applyTo(price);
    }
```

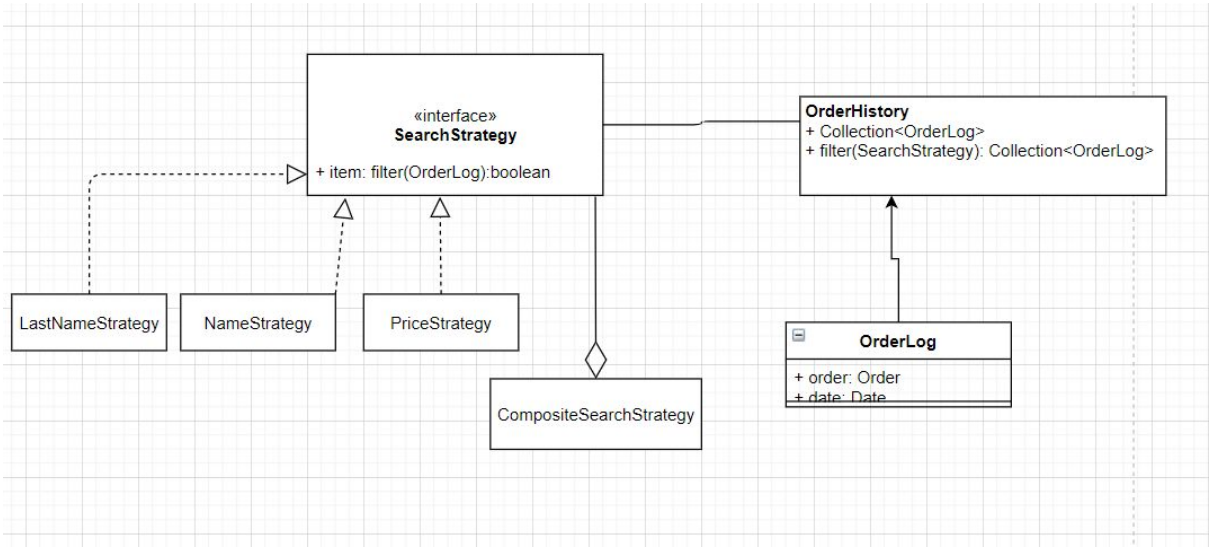
```java
public BigDecimal getPrice() { return getPriceWithFunction((Product::getPrice)); }

public BigDecimal getPriceWithOrderDiscount() {
    return applyDiscountTo(getPriceWithProductsDiscount());
}

public BigDecimal getPriceWithProductsDiscount() {
    return getPriceWithFunction((Product::getDiscountedPrice));
}

private BigDecimal getPriceWithFunction(Function<Product, BigDecimal> function) {
    return products.stream()
            .map(function)
            .reduce(BigDecimal.ZERO, BigDecimal::add);
}
```

i. Wyniki testów



4. **Umożliwić przechowywanie historii zamówień z wyszukiwaniem po: nazwie produktu, kwocie zamówienia, nazwisku zamawiającego. Wyszukiwać można przy użyciu jednego lub wielu kryteriów**

Do wprowadzenia strategii wyszukiwania zastosujemy Composite design pattern. Tak jak miało to miejsce w labolatorium 4.



Klasa reprezentująca pojedynczy rekord historii:

```
package pl.edu.agh.internetshop;

import java.util.Date;

public class OrderLog {

    Order order;
    Date date;
//   put here another atributes which you want to log

    public OrderLog(Order o){
        order = o;
    }
}
```

Klasa reprezentująca całą historię z możliwością wyszukiwania:

```java
public class OrderHistory {

    Collection<OrderLog> history;
    SearchStrategy strategy;

    public OrderHistory(SearchStrategy strategy) {
        this.history = new LinkedList<>();
        this.strategy = strategy;
    }

    public Collection<OrderLog> getHistory() { return history; }

    public void addLog(OrderLog o) { this.history.add(o); }

    public SearchStrategy getStrategy() {
        return strategy;
    }

    public void setStrategy(SearchStrategy strategy) { this.strategy = strategy; }

    public ArrayList<OrderLog> filter(){
        ArrayList<OrderLog> result = new ArrayList<>();
        for (OrderLog o: this.history) {
            if(strategy.filter(o)){
                result.add(o);
            }
        }
        return result;
    }
}
```

Intrefejs do implementacji nowych strategii wyszukiwania

```java
package pl.edu.agh.internetshop;

public interface SearchStrategy {
    boolean filter(OrderLog o);
}
```

Poszczególne strategie wyszukiwania:

```java
public class OrderPriceStrategy implements SearchStrategy{

    double price;

    public OrderPriceStrategy(double price) {
        this.price = price;
    }

    @Override
    public boolean filter(OrderLog o) {
        return o.order.getPrice().equals(price);
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

```java
public class ProductNameStrategy implements SearchStrategy {
    String productName;

    public ProductNameStrategy(String productName) {
        this.productName = productName;
    }

    @Override
    public boolean filter(OrderLog o) {
        return o.order.getProducts().stream().anyMatch(p -> p.getName().equals(this.productName));
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }
}
```

```java
public class NameStrategy implements SearchStrategy{

    String lastName;

    public NameStrategy(String lastName) {
        this.lastName = lastName;
    }

    @Override
    public boolean filter(OrderLog o) {

        return o.order.getShipment().getRecipientAddress().getName().equals(lastName);

    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

Composite Search Strategy potrzebny do łączenia warunków wyszukiwania

```java
public class CompositeSearchStrategy implements SearchStrategy{

    Collection<SearchStrategy> strategies;

    public CompositeSearchStrategy() {
        this.strategies = new ArrayList<>();
    }

    @Override
    public boolean filter(OrderLog o) {
        boolean flag = true;
        for(SearchStrategy s: strategies){
            if(!s.filter(o)){
                flag = false;
                break;
            }
        }
        return flag;
    }

    public  void addSearchStrategy(SearchStrategy s){
        strategies.add(s);
    }
}
```

Przykładowe testy do strategii wyszukiwania:

```java
public class NameStrategyTest {

    private static final String NAME = "Sparkle";

    @Test
    public void testCorrectFilter(){
        //given
        OrderLog orderLog = mock(OrderLog.class);
        orderLog.order = mock(Order.class);
        NameStrategy strategy = new NameStrategy(NAME);
        //when
        when(orderLog.order.getShipment()).thenReturn(mock(Shipment.class));
        when(orderLog.order.getShipment().getRecipientAddress()).thenReturn(mock(Address.class));
        when(orderLog.order.getShipment().getRecipientAddress().getName()).thenReturn(NAME);
        //then
        assertTrue(strategy.filter(orderLog));
    }
```

```java
    @Test
    public void testUncorrectFIlter(){
        //given
        OrderLog orderLog = mock(OrderLog.class);
        orderLog.order = mock(Order.class);
        NameStrategy strategy = new NameStrategy( lastName: "NotName");
        //when
        when(orderLog.order.getShipment()).thenReturn(mock(Shipment.class));
        when(orderLog.order.getShipment().getRecipientAddress()).thenReturn(mock(Address.class));
        when(orderLog.order.getShipment().getRecipientAddress().getName()).thenReturn(NAME);
/then
        assertFalse(strategy.filter(orderLog));
    }
```

```java
    @Test
    public void testSetStrategyName(){
        //given
        OrderLog orderLog = mock(OrderLog.class);
        orderLog.order = mock(Order.class);
        NameStrategy strategy = new NameStrategy( lastName: "NotName");
        //when
        when(orderLog.order.getShipment()).thenReturn(mock(Shipment.class));
        when(orderLog.order.getShipment().getRecipientAddress()).thenReturn(mock(Address.class));
        when(orderLog.order.getShipment().getRecipientAddress().getName()).thenReturn(NAME);
//then
        assertFalse(strategy.filter(orderLog));
        strategy.setLastName(NAME);
        assertTrue(strategy.filter(orderLog));
    }

}
```

Testowanie CompositeSearchStrategy

```java
public class CompositeSearchStrategyTest {

    @Test
    public void SearchOneFalse(){
        //given
        CompositeSearchStrategy strategy = new CompositeSearchStrategy();
        SearchStrategy s1 = mock(SearchStrategy.class);
        SearchStrategy s2 = mock(SearchStrategy.class);
        SearchStrategy s3 = mock(SearchStrategy.class);
        SearchStrategy s4 = mock(SearchStrategy.class);
        OrderLog o = mock(OrderLog.class);
        //when

        when(s1.filter(o)).thenReturn(true);
        when(s1.filter(o)).thenReturn(true);
        when(s1.filter(o)).thenReturn(true);
        when(s1.filter(o)).thenReturn(false);

        strategy.addSearchStrategy(s1);
        strategy.addSearchStrategy(s2);
        strategy.addSearchStrategy(s3);
        strategy.addSearchStrategy(s4);
        //then
        assertFalse(strategy.filter(o));
    }
```

```java
    public void SearchAllTrue(){
        //given
        CompositeSearchStrategy strategy = new CompositeSearchStrategy();
        SearchStrategy s1 = mock(SearchStrategy.class);
        SearchStrategy s2 = mock(SearchStrategy.class);
        SearchStrategy s3 = mock(SearchStrategy.class);
        SearchStrategy s4 = mock(SearchStrategy.class);
        OrderLog o = mock(OrderLog.class);
        //when

        when(s1.filter(o)).thenReturn(true);
        when(s1.filter(o)).thenReturn(true);
        when(s1.filter(o)).thenReturn(true);
        when(s1.filter(o)).thenReturn(true);

        strategy.addSearchStrategy(s1);
        strategy.addSearchStrategy(s2);
        strategy.addSearchStrategy(s3);
        strategy.addSearchStrategy(s4);
        //then
        assertTrue(strategy.filter(o));
    }

}
```

Testowanie Filtrowania OrderHistory:

```java
import static org.mockito.Mockito.when;
public class OrderHistoryTest {

    @Test
    public void filterTest(){
        //given
        SearchStrategy s1 = mock(SearchStrategy.class);
        OrderHistory history = new OrderHistory(s1);
        OrderLog o0 = mock(OrderLog.class);
        OrderLog o1 = mock(OrderLog.class);
        OrderLog o2 = mock(OrderLog.class);
        OrderLog o3= mock(OrderLog.class);
        ArrayList<OrderLog> ol = new ArrayList<>();
        ol.add(o0);
        ol.add(o1);
        ol.add(o2);
        ol.add(o3);

        ArrayList<OrderLog> expected = new ArrayList<>();
        expected.add(o0);
        expected.add(o2);
        //when
        when(s1.filter(o0)).thenReturn(true);
        when(s1.filter(o1)).thenReturn(false);
        when(s1.filter(o2)).thenReturn(true);
```

```java
        ArrayList<OrderLog> expected = new ArrayList<>();
        expected.add(o0);
        expected.add(o2);
        //when
        when(s1.filter(o0)).thenReturn(true);
        when(s1.filter(o1)).thenReturn(false);
        when(s1.filter(o2)).thenReturn(true);
        when(s1.filter(o3)).thenReturn(false);
        history.strategy = s1;
        history.history = ol;
        ArrayList<OrderLog> res = history.filter();

        //then
        assertEquals( expected: 2, res.size());
        assertEquals(o0, res.get(0));
        assertEquals(o2, res.get(1));
    }
}
```

Na koniec wszystkie testy przechodzą: