# Testowanie i weryfikacja oprogramowania

Projekt 1 - Baza danych "Sklep"

Mateusz Bocheński
Paweł Kłapsa
Jacek Kozieja
Mateusz Stefaniak

16 grudnia 2015

## Spis treści



## 1 Wstęp

Celem projektu byłu stworzenie aplikacji z wykorzystaniem Test-driven development. Technika ta zaliczana jest do metodyk zwinnych. Charakterystyczna cecha dla tej techniki jest tworzenie testów przed właściwą implementacją.

W projekcie rozwijaliśmy aplikację Java, porozumiewającą się z bazą danych, dodając do niej metody umożliwiające pobieranie danych statystycznych sprzedaży.

## 2 Nowe funkcjonalności

Nowe funkcjonalności reprezentowane są przez następujący interfejs:

```
List<Product> getBestSellingProducts(Date from, Date to, int count);

List<User>  getBestBuyersByValue(int count);

List<User> getBestBuyersByOperations(int count);
```

Opis poszczególnych metod:

- *getBestSellingProducts* - najlepiej sprzedające się produkty w danym okresie. Przyjmowane parametry:
  - **from** - data początkowa
  - **to** - data końcowa
  - **count** - ilość
- *getBestBuyersByValue* - najwięcej wydający klienci (według łącznie wydanej kwoty). Jedynym parametrem jest ilość klientów.
- *getBestBuyersByOperations* - najczęściej kupujący klienci (według ilości zakupionych produktów). Jedynym parametrem jest ilość klientów.

## 3 Testy

W kolejnych sekcjach, umieszczone zostały opisy testów poszczególnych funkcjonalności.

### 3.1 Test metody *getBestBuyersByValue*

Opis: metoda testowa sprawdza czy zwracana lista użytkowników jest posortowana według ilości wydanej przez nich kwoty. Kolejno metoda sprawdza prawidłowość działania dla różnej ilości pobieranych użytkowników (w tym dla wartości skrajnych jak wartości ujemne oraz wartości bardzo duże, wykraczające poza rzeczywistą ilość użytkowników)

Kod:

```
@Override
public void getBestBuyersByValueTest() throws CRUDOperationException {
    List<ShoppingHistory> list = new LinkedList<ShoppingHistory>();
    User u = new User("Bogdan");
    list.add(createObject(u, "b1", "productName", 1));
    list.add(createObject(u, "b2", "ProductName", 120.12));
    list.add(createObject(u, "b2", "ProductName", 70.75));
    list.add(createObject(new User("Adam"), "b2", "ProductName", 25.5));
    list.add(createObject(new User("Zyta"), "b2", "ProductName", 0));
    u = new User("Marta");
    list.add(createObject(u, "b2", "ProductName", 9.99));
    list.add(createObject(u, "b2", "ProductName", 12.59));

    StatisticsServiceImpl stat = new StatisticsServiceImpl(list);
    //Oczekiwane wyniki to: Bogdan->Adam->Marta->Zyta
    //Dla 0 metoda ma zwracać null, dla arg. wiekszego od historii zakupów - posortowana calosc
    List<User> list1 = new LinkedList(Arrays.asList(list.get(0).getUser()));
    assertEquals(list1, stat.getBestBuyersByValue(1));
    list1.add(list.get(3).getUser());
    assertEquals(list1, stat.getBestBuyersByValue(2));
    assertNull(stat.getBestBuyersByValue(0));
    list1.add(list.get(5).getUser());
    list1.add(list.get(4).getUser());
    assertEquals(list1, stat.getBestBuyersByValue(100));
}
```

### 3.2 Test metody *getBestBuyersByOperations*

Opis: metoda testowa sprawdza czy zwracana lista użytkowników jest posortowana według ilości zakupionych przez nich produktów. Kolejno metoda sprawdza prawidłowość działania dla różnej ilości pobieranych użytkowników (w tym dla wartości skrajnych jak wartości ujemne oraz wartości bardzo duże, wykraczające poza rzeczywistą ilość użytkowników)

Kod:

```
@Override
    public void getBestBuyersByOperationsTest() throws CRUDOperationException {
        List<ShoppingHistory> list = new LinkedList();
        User u = new User("Marta");
        list.add(createObject(u, "b1", "productName", 1));
        list.add(createObject(u, "b2", "ProductName", 12.59));
```

```
        list.add(createObject(u, "b2", "ProductName", 9.99));
        u = new User("Adam");
        list.add(createObject(u, "b2", "ProductName", 25.5));
        list.add(createObject(u, "b2", "ProductName", 70.75));
        list.add(createObject(new User("Bogdan"), "b2", "ProductName", 120.12));
        list.add(createObject(new User("Zyta"), "b2", "ProductName", 0));

        StatisticsServiceImpl stat = new StatisticsServiceImpl(list);
        //Oczekiwane wyniki to: Marta -> Adam -> Bogdan/Zyta
        List<User> list1 = new LinkedList();
        assertEquals(list1, stat.getBestBuyersByValue(0));
        assertEquals(list1, stat.getBestBuyersByValue(-10));
        list1.add(list.get(0).getUser());
        assertEquals(list1, stat.getBestBuyersByOperations(1));
        list1.add(list.get(3).getUser());
        assertEquals(list1, stat.getBestBuyersByOperations(2));
        list1.add(list.get(5).getUser());
        list1.add(list.get(6).getUser());
        assertEquals(list1, stat.getBestBuyersByOperations(100));
    }
```

# 4 Wnioski

Testy wykazały poprawność działania programu przy standardowych, zamierzonych przypadkach użycia.

Test driven development daje większą gwarancję poprawnego działania aplikacji, dzięki konieczności pisania testów już na samym początku implementacji nowych funkcjonalności.