Projektowanie obiektowe

Lab 2 Wprowadzanie zmian w istniejącej aplikacji

Dawid Białka, Piotr Tekielak

Zad 2.1.

W celu dodania nowych właściwości dla produktów w poszczególnych kategoriach stworzyliśmy nowe klasy dziedziczące z klasy Item:

1. Book – klasa reprezentująca książki posiadająca dodatkowe atrybuty pages i hardcover

```java
package pl.edu.agh.dronka.shop.model;

public class Book extends Item {

    private int pages;

    private boolean hardcover;

    public Book(String name, Category category, int price, int quantity, int pages, boolean hardcover) {
        this.name = name;
        this.category = category;
        this.price = price;
        this.quantity = quantity;
        this.pages = pages;
        this.hardcover = hardcover;
    }

    public Book() {
    }

    public void setPages(int pages) { this.pages = pages; }

    public int getPages() { return pages; }

    public void setHardcover(boolean hardcover) { this.hardcover = hardcover; }

    public boolean isHardcover() { return hardcover; }
}
```

2. Electronics – klasa reprezentująca elektronikę posiadająca dodatkowe atrybuty mobile i warranty

```java
package pl.edu.agh.dronka.shop.model;

public class Electronics extends Item {

    private boolean mobile;

    private boolean warranty;

    public Electronics(String name, Category category, int price, int quantity, boolean mobile, boolean warranty) {
        this.name = name;
        this.category = category;
        this.price = price;
        this.quantity = quantity;
        this.mobile = mobile;
        this.warranty = warranty;
    }
    public Electronics() {
    }

    public void setMobile(boolean mobile) { this.mobile = mobile; }

    public boolean isMobile() { return mobile; }

    public void setWarranty(boolean warranty) { this.warranty = warranty; }

    public boolean isWarranty() { return warranty; }
}
```

3. Food – klasa reprezentująca jedzenie posiadająca dodatkowy atrybut expirationDate

```java
package pl.edu.agh.dronka.shop.model;

import java.util.Date;

public class Food extends Item {

    private Date expirationDate;

    public Food(String name, Category category, int price, int quantity, Date expirationDate) {
        this.name = name;
        this.category = category;
        this.price = price;
        this.quantity = quantity;
        this.expirationDate = expirationDate;
    }
    public Food() {
    }

    public void setExpirationDate(Date expirationDate) { this.expirationDate = expirationDate; }

    public Date getExpirationDate() { return expirationDate; }
}
```

4. Music – klasa reprezentująca muzykę posiadająca dodatkowe atrybuty genre i videoAdded

```java
package pl.edu.agh.dronka.shop.model;

public class Music extends Item {

    private MusicGenre genre;

    private boolean videoAdded;

    public Music(String name, Category category, int price, int quantity, MusicGenre genre, boolean videoAdded) {
        this.name = name;
        this.category = category;
        this.price = price;
        this.quantity = quantity;
        this.genre = genre;
        this.videoAdded = videoAdded;
    }

    public Music() {
    }

    public void setGenre(MusicGenre genre) { this.genre = genre; }

    public MusicGenre getGenre() { return genre; }

    public void setVideoAdded(boolean videoAdded) { this.videoAdded = videoAdded; }

    public boolean isVideoAdded() { return videoAdded; }
}
```

Tworzymy także dodatkowy typ wyliczeniowy MusicGenre, wykorzystywany przez atrybut genre z klasy Music, reprezentujący różne gatunki muzyczne.

```java
package pl.edu.agh.dronka.shop.model;

public enum MusicGenre {

    DISCOPOLO( displayName: "Disco Polo"), RAP( displayName: "Rap"), METAL( displayName: "Metal"),
    SERBIANMILITARYMUSIC( displayName: "Serbian military music");

    private String displayName;

    public String getDisplayName() { return displayName; }

    private MusicGenre(String displayName) { this.displayName = displayName; }
}
```

W celu wyświetlenia dodatkowych atrybutów wprowadzamy zmiany w dwóch klasach – PropertiesHelper i ShopProvider.

1. W klasie PropertiesHelper jeśli przedmiot należy do kategorii z dodatkowymi atrybutami mapujemy do na obiekt z klasy dziedziczącej i dodajemy jego właściwości wyłączne dla niego do mapy z wszystkimi właściwościami

```
public static Map<String, Object> getPropertiesMap(Item item) {
    Map<String, Object> propertiesMap = new LinkedHashMap<>();

    propertiesMap.put( k: "Nazwa", item.getName());
    propertiesMap.put( k: "Cena", item.getPrice());
    propertiesMap.put( k: "Kategoria", item.getCategory().getDisplayName());
    propertiesMap.put( k: "Ilość", Integer.toString(item.getQuantity()));
    propertiesMap.put( k: "Tanie bo polskie", item.isPolish());
    propertiesMap.put( k: "Używany", item.isSecondhand());
    if(item.getCategory() == Category.BOOKS) {
        propertiesMap.put( k: "Liczba stron", ((Book)item).getPages());
        propertiesMap.put( k: "Twarda oprawa", ((Book)item).isHardcover());
    }
    if(item.getCategory() == Category.ELECTRONICS) {
        propertiesMap.put( k: "Mobilny", ((Electronics)item).isMobile());
        propertiesMap.put( k: "Gwarancja", ((Electronics)item).isWarranty());
    }
    if(item.getCategory() == Category.FOOD) {
        propertiesMap.put( k: "Data ważności", ((Food)item).getExpirationDate());
    }
    if(item.getCategory() == Category.MUSIC) {
        propertiesMap.put( k: "Gatunek muzyczny", ((Music)item).getGenre().getDisplayName());
        propertiesMap.put( k: "Z teledyskiem", ((Music)item).isVideoAdded());
    }

    return propertiesMap;
}
```

2. W klasie ShopProvider, która za pomocą informacji z plików CSV tworzy nowe obiekty uzależniamy typ tworzonego obiektu od kategorii do której należy przypisując mu dodatkowe atrybuty

```
switch (category) {
    case BOOKS:
        int pages = Integer.parseInt(reader.getValue(dataLine,  name: "Liczba stron"));
        boolean hardback = Boolean.parseBoolean(reader.getValue(dataLine,  name: "Twarda oprawa"));
        item = new Book(name, category, price, quantity, pages, hardback);
        item.setPolish(isPolish);
        item.setSecondhand(isSecondhand);

        items.add(item);
        break;
    case ELECTRONICS:
        boolean mobile = Boolean.parseBoolean(reader.getValue(dataLine,  name: "Mobilny"));
        boolean warranty = Boolean.parseBoolean(reader.getValue(dataLine,  name: "Gwarancja"));
        item = new Electronics(name, category, price, quantity, mobile, warranty);
        item.setPolish(isPolish);
        item.setSecondhand(isSecondhand);

        items.add(item);
        break;
    case FOOD:
        int year = Integer.parseInt(reader.getValue(dataLine,  name: "Data przydatności rok"));
        int month = Integer.parseInt(reader.getValue(dataLine,  name: "Data przydatności miesiąc"));
        int day = Integer.parseInt(reader.getValue(dataLine,  name: "Data przydatności dzień"));
        Date date = new GregorianCalendar(year, month, day).getTime();
        item = new Food(name, category, price, quantity, date);
        item.setPolish(isPolish);
        item.setSecondhand(isSecondhand);

        items.add(item);
        break;
```

Zad 2.2

Chcąc dodać możliwość filtrowania po właściwościach typu boolean, specyficznego dla danej kategorii, na początku tworzymy puste konstruktory (jak w klasie Item) dla klas Book, Electronics i Music.

```java
package pl.edu.agh.dronka.shop.model;

public class Book extends Item {

    private int pages;

    private boolean hardcover;

    public Book(String name, Category category, int price, int quantity, int pages, boolean hardcover) {
        this.name = name;
        this.category = category;
        this.price = price;
        this.quantity = quantity;
        this.pages = pages;
        this.hardcover = hardcover;
    }

    public Book() {
    }

    public void setPages(int pages) { this.pages = pages; }

    public int getPages() { return pages; }

    public void setHardcover(boolean hardcover) { this.hardcover = hardcover; }

    public boolean isHardcover() { return hardcover; }
}
```

W klasie ItemFilter zmieniamy konstruktor w ten sposób, aby tworzył obiekt na podstawie kategorii, która zostanie mu przekazana. Następnie w zależności od kategorii w ShopController filtr jest ustawiany, aby filtrował wartość charakterystyczną dla danej kategorii lub nie.

```java
package pl.edu.agh.dronka.shop.model.filter;

import pl.edu.agh.dronka.shop.model.*;

public class ItemFilter {

    private Item itemSpec;

    public ItemFilter(Category category) {
        if(category.equals(Category.BOOKS)) {
            this.itemSpec = new Book();
        }
        else if(category.equals(Category.ELECTRONICS)) {
            this.itemSpec = new Electronics();
        }
        else if(category.equals(Category.MUSIC)) {
            this.itemSpec = new Music();
        }
        else this.itemSpec = new Item();
    }
```

```java
    // applies filter only if the flag (secondHand) is true)
    if (itemSpec.isSecondhand() && !item.isSecondhand()) {
        return false;
    }

    // applies filter only if the flag (polish) is true)
    if (itemSpec.isPolish() && !item.isPolish()) {
        return false;
    }

    if (itemSpec.getCategory().equals(Category.BOOKS) &&
            ((Book)itemSpec).isHardcover() && !((Book)item).isHardcover()) {
        return false;
    }

    if (itemSpec.getCategory().equals(Category.ELECTRONICS) &&
            ((Electronics)itemSpec).isMobile() && !((Electronics)item).isMobile()) {
        return false;
    }

    if (itemSpec.getCategory().equals(Category.ELECTRONICS) &&
            ((Electronics)itemSpec).isWarranty() && !((Electronics)item).isWarranty()) {
        return false;
    }

    if (itemSpec.getCategory().equals(Category.MUSIC) &&
            ((Music)itemSpec).isVideoAdded() && !((Music)item).isVideoAdded()) {
        return false;
    }

    return true;
```

Aby wyświetlić dodatkowe możliwości filtrowania, gdy przeglądana jest dana kategoria, w klasie PropertiesPanel zostały dodane nowe check boxy. W zależności od kategorii, obiekt Item pobrany z filtra mapujemy i wykorzystując metodą dla danej kategorii ustawiamy, żeby filtr był włączony na daną wartość lub nie w zależności, czy check box jest zaznaczony, czy nie.

```
if(currentCategory.equals(Category.BOOKS)) {

    add(createPropertyCheckbox( propertyName: "Twarda okładka", new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent event) {
            ((Book) filter.getItemSpec()).setHardcover(
                    ((JCheckBox) event.getSource()).isSelected());
            shopController.filterItems(filter);
        }
    }));
}

if(currentCategory.equals(Category.ELECTRONICS)) {

    add(createPropertyCheckbox( propertyName: "Mobilny", new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent event) {
            ((Electronics)filter.getItemSpec()).setMobile(
                    ((JCheckBox) event.getSource()).isSelected());
            shopController.filterItems(filter);
        }
    }));

    add(createPropertyCheckbox( propertyName: "Gwarancja", new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent event) {
            ((Electronics)filter.getItemSpec()).setWarranty(
                    ((JCheckBox) event.getSource()).isSelected());
            shopController.filterItems(filter);
        }
    }));
}
```

```
if(currentCategory.equals(Category.MUSIC)) {

    add(createPropertyCheckbox( propertyName: "Z teledyskiem", new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent event) {
            ((Music)filter.getItemSpec()).setVideoAdded(
                    ((JCheckBox) event.getSource()).isSelected());
            shopController.filterItems(filter);
        }
    }));

}
```