Projektowanie obiektowe

Laboratorium 2

Wprowadzanie zmian w istniejącej aplikacji

1. Na początku przetworzyłem klasę Item na klasę abstraktną

```
public abstract class Item {
```

2. Stworzyłem klasy dla każdej kategorii, które są klasami podrzędnymi klasy Item

2.1 Books

```
public class Books extends Item {

    private int pageNumber;

    private boolean hardcover;

    public Books(String name, Category category, int price, int quantity) { super(name, category, price, quantity); }

    public Books() {
    }

    public void setPageNumber(int pageNumber) { this.pageNumber = pageNumber; }

    public int getPageNumber() { return this.pageNumber; }

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

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

2.2 Electronics

```
public class Electronics extends Item {

    private boolean mobile;

    private boolean warranty;

    public Electronics(String name, Category category, int price, int quantity) {
        super(name, category, price, quantity);
    }

    public Electronics() {
    }

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

    public boolean isMobile() { return this.mobile; }

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

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

2.3 Food

```java
public class Food extends Item {

    private Date expirationDate;

    public Food(String name, Category category, int price, int quantity) {
        super(name, category, price, quantity);
    }

    public Food() {
    }

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

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

2.4 Music

```java
public class Music extends Item {

    private MusicalGenre musicalGenre;

    private boolean videoAttached;

    public Music(String name, Category category, int price, int quantity) {
        super(name, category, price, quantity);
    }

    public Music() {
    }

    public void setMusicalGenre(MusicalGenre musicalGenre) { this.musicalGenre = musicalGenre; }

    public MusicalGenre getMusicalGenre() { return this.musicalGenre; }

    public void setVideoAttached(boolean videoAttached) { this.videoAttached = videoAttached; }

    public boolean isVideoAttached() { return this.videoAttached; }
}
```

2.5 Enum MusicGenre

```java
public enum MusicalGenre {
    RAP( displayName: "Rap"), POP( displayName: "Pop"), ROCK( displayName: "Rock"), DANCE( displayName: "Dance"), CLASSICAL( displayName: "Classical");

    private String displayName;

    public String getDisplayName() { return displayName; }

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

2.6 Sport

```java
public class Sport extends Item {

    public Sport(String name, Category category, int price, int quantity) {
        super(name, category, price, quantity);
    }

    public Sport() {
    }
}
```

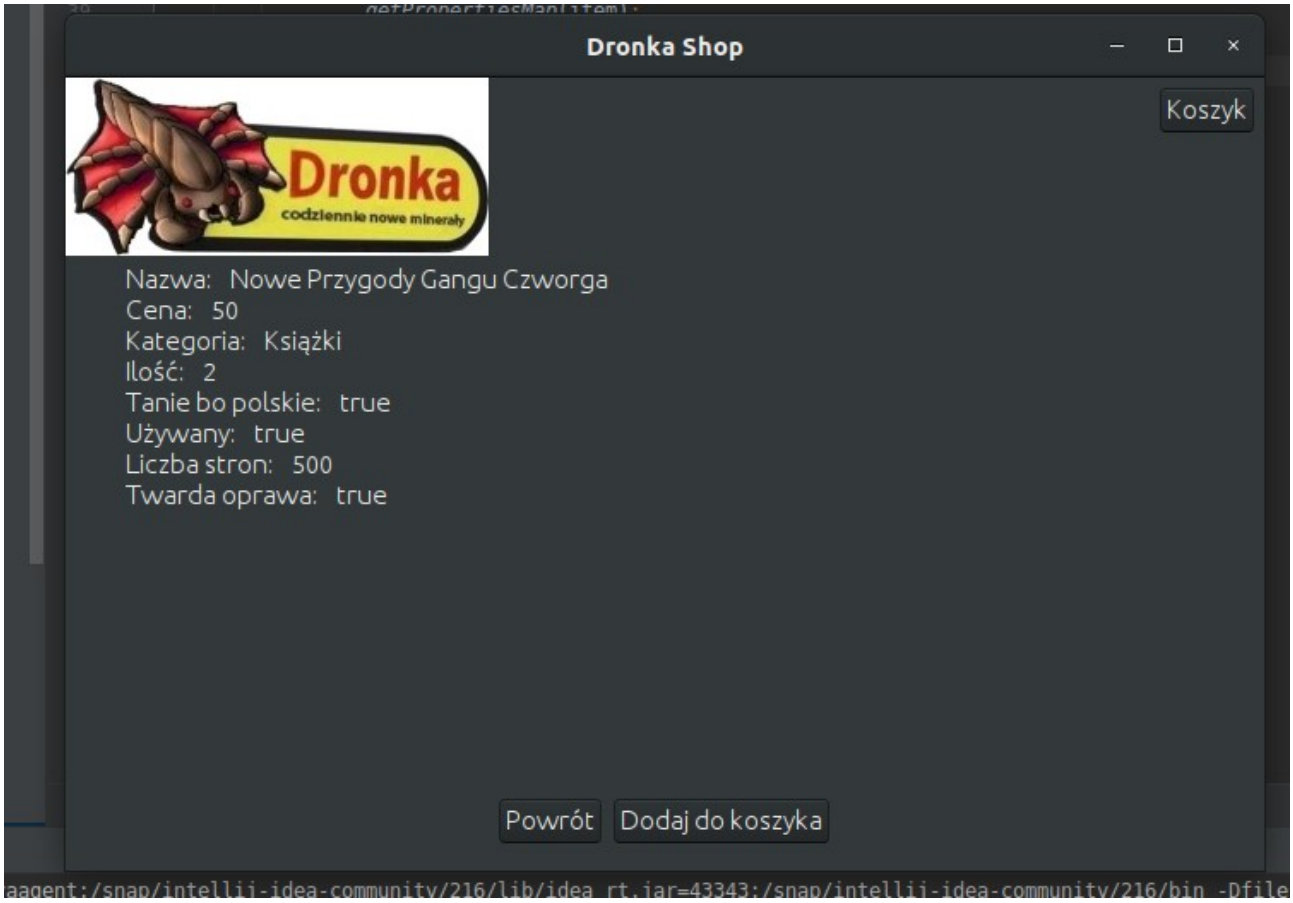
3. W klasie ShopProvider w funkcji readItems dodałem switch’a, sprawdzającego, która to kategoria I dodaje nowe atrybuty, jeżeli są one potrzebne. Dodatkowo dodałem niektóre niewystarczające dane w plikach konfiguracyjnych.

```java
Item item;

switch (category) {
    case BOOKS:
        int pageNumber = Integer.parseInt(reader.getValue(dataLine, name: "Liczba stron"));
        boolean isHardcover = Boolean.parseBoolean(reader.getValue(dataLine, name: "Twarda oprawa"));
        item = new Books(name, category, price, quantity);
        ((Books) item).setPageNumber(pageNumber);
        ((Books) item).setHardcover(isHardcover);
        break;

    case ELECTRONICS:
        boolean isMobile = Boolean.parseBoolean(reader.getValue(dataLine, name: "Mobilny"));
        boolean isWarranty = Boolean.parseBoolean(reader.getValue(dataLine, name: "Gwarancja"));
        item = new Electronics(name, category, price, quantity);
        ((Electronics) item).setMobile(isMobile);
        ((Electronics) item).setWarranty(isWarranty);
        break;

    case FOOD:
        String sExpirationDate = reader.getValue(dataLine, name: "Data przydatności do spożycia");
        SimpleDateFormat formatter = new SimpleDateFormat( pattern: "dd/MM/yyyy");
        Date expirationDate = formatter.parse(sExpirationDate);
        item = new Food(name, category, price, quantity);
        ((Food) item).setExpirationDate(expirationDate);
        break;

    case MUSIC:
        String displayName = reader.getValue(dataLine, name: "Gatunek muzyczny");
        MusicalGenre musicalGenre = MusicalGenre.valueOf(displayName.toUpperCase());
        boolean videoAttached = Boolean.parseBoolean(reader.getValue(dataLine, name: "Dołączone video"));
        item = new Music(name, category, price, quantity);
        ((Music) item).setMusicalGenre(musicalGenre);
        ((Music) item).setVideoAttached(videoAttached);
        break;

    case SPORT:
        item = new Sport(name, category, price, quantity);
        break;

    default:
        throw new IllegalStateException("Unexpected value: " + category);
}

item.setPolish(isPolish);
item.setSecondhand(isSecondhand);

items.add(item);
```

4. Żeby wyświetlić nowe parametry, dodałem do funkcji getPropertiesmap w klasie PropertiesHelper if’a, który po sprawdzeniu instancji item’a dodaje do mapy dodatkowe elementy.

```
public class PropertiesHelper {

    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 instanceof Books) {
            propertiesMap.put( k: "Liczba stron", ((Books) item).getPageNumber());
            propertiesMap.put( k: "Twarda oprawa", ((Books) item).isHardcover());
        } else if (item instanceof Electronics) {
            propertiesMap.put( k: "Mobilny", ((Electronics) item).isMobile());
            propertiesMap.put( k: "Gwarancja", ((Electronics) item).isWarranty());
        } else if (item instanceof Food) {
            propertiesMap.put( k: "Data przydatności do spożycia",
                    ((Food) item).getExpirationDate().toString());
        } else if (item instanceof Music) {
            propertiesMap.put( k: "Gatunek muzyczny",
                    ((Music) item).getMusicalGenre().getDisplayName());
            propertiesMap.put( k: "Dołączone video",
                    ((Music) item).isVideoAttached());
        }

        return propertiesMap;
    }
}
```

5. Poniżej przedstawiłem przykładowy screen z wyświetlającymi się na nim dodatkowym polami.



6. Następnym krokiem było umożliwienie sortowania dla nowych parametrów. Na początku zmieniłem klasę ItemFilter w taki sposób, żeby klasa ta wiedziała, dla której kategorii musi tworzyć instancje item’a

```
import java.awt.print.Book;

public class ItemFilter {

    private Item itemSpec;
    private Category category;

    public ItemFilter(Category category) {
        this.category = category;

        if (category == Category.BOOKS) {
            itemSpec = new Books();
        } else if (category == Category.ELECTRONICS) {
            itemSpec = new Electronics();
        } else if (category == Category.FOOD) {
            itemSpec = new Food();
        } else if (category == Category.MUSIC) {
            itemSpec = new Music();
        } else if (category == Category.SPORT) {
            itemSpec = new Sport();
        } else {
            itemSpec = null;
        }
    }
```

7. Dodatkowo w klasie ItemFilter zmieniłem funkcje appliesTo w taki sposób, żeby mogła ona filtrować nowe pola.

```
public boolean appliesTo(Item item) {
    if (itemSpec.getName() != null
            && !itemSpec.getName().equals(item.getName())) {
        return false;
    }
    if (itemSpec.getCategory() != null
            && !itemSpec.getCategory().equals(item.getCategory())) {
        return false;
    }

    // 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 (category == Category.BOOKS) {
        // applies filter only if the flag (hardcover) is true
        if (((Books) itemSpec).isHardcover() && !((Books) item).isHardcover()) {
            return false;
        }
    }

    if (category == Category.ELECTRONICS) {
        // applies filter only if the flag (mobile) is true
        if (((Electronics) itemSpec).isMobile() && !((Electronics) item).isMobile()) {
            return false;
        }

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

    if (category == Category.MUSIC) {
        // applies filter only if the flag (videoAttached) is true
        if (((Music) itemSpec).isVideoAttached() && !((Music) item).isVideoAttached()) {
            return false;
        }
    }

    return true;
}
```

8. Na koniec zmieniłem funkcje fillProperties w klasie PropertiesPanel w takie sposób, żeby ona umożliwiała sortowanie item’ów po dodanych wcześniej pól.

```
nFilter.java × PropertiesPanel.java ×
            }
        }));

        if (shopController.getCurrentCategory() == Category.BOOKS) {
            add(createPropertyCheckbox( propertyName: "Twarda oprawa", new ActionListener() {

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

        if (shopController.getCurrentCategory() == 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 (shopController.getCurrentCategory() == Category.MUSIC) {
            add(createPropertyCheckbox( propertyName: "Dołączone video", new ActionListener() {

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

9. Poniżej przedstawiłem przykładowy screen z filtrowaniem item’ów przez dodane pola.