Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5

Теорія Розробки Програмного Забезпечення

Шаблон «Prototype»

Предметна область: **Особиста бухгалтерія**

Виконав
студент групи ІА-14:
Яковенко Ю.О.

Перевірив:
Мягкий М.Ю.

Київ 2023

Мета: Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

Виконання

Завданням на лобораторну було реалізувати шаблон ***prototype***.

Шаблон "prototype" (прототип) використовується для створення об'єктів за "шаблоном" (чи "кресленню", "ескізу") шляхом копіювання шаблонного об'єкту. Для цього визначається метод "клонувати" в об'єктах цього класу.

Цей шаблон зручно використати, коли заздалегідь відомо як виглядатиме кінцевий об'єкт (мінімізується кількість змін до об'єкту шляхом створення шаблону), а також для видалення необхідності створення об'єкту - створення відбувається за рахунок клонування, і зухвалій програмі абсолютно немає необхідності знати, як створювати об'єкт.

Також, це дозволяє маніпулювати об'єктами під час виконання програми шляхом настроювання відповідних шаблонів; значно зменшується ієрархія спадкоємства (оскільки в іншому випадку це були б не шаблони, а вкладені класи, що наслідують).

Я реалізовував його в entity класі Transaction який інколи доводиться копіювати. Тож я імплементував інтерфейс Clonable та реалізував його функціонал.

Імплементація:

```
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Entity
@Table(name = "transaction")
public class Transaction implements Cloneable {}
```

Параметри класу:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id")
private int id;

@Min(value = 0, message = "Transaction sum can't be negative")
@Column(name = "sum")
private int sum;

@Length(max = 150, message = "Comment should be less then 150 characters")
@Column(name = "comment")
private String comment;
```

```
@NotNull(message = "Category can't be empty")
@Enumerated(value = EnumType.STRING)
@Column(name = "category")
private TransactionCategory category;

@Column(name = "refill")
private boolean refill;

@PastOrPresent
@Temporal(TemporalType.TIMESTAMP)
@Column(name = "date_time")
private LocalDateTime dateTime;

@ManyToOne
@JoinColumn(name = "user_id", referencedColumnName = "id")
private User user;
```

Реалізація clonable:

```
@Override
public Transaction clone() {

    try {
        Transaction clone = (Transaction) super.clone();
        clone.setSum(sum);
        clone.setDateTime(dateTime);
        clone.setCategory(category);
        clone.setRefill(refill);
        clone.setComment(comment);
        return clone;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError();
    }
}
```

Ввесь клас Transaction:

```
package com.example.PersonalAccounting.entity;

import com.example.PersonalAccounting.entity.enums.TransactionCategory;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.hibernate.validator.constraints.Length;

import java.time.LocalDateTime;

@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Entity
@Table(name = "transaction")
public class Transaction implements Cloneable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    @Column(name = "id")
    private int id;

    @Min(value = 0, message = "Transaction sum can't be negative")
    @Column(name = "sum")
    private int sum;

    @Length(max = 150, message = "Comment should be less then 150
characters")
    @Column(name = "comment")
    private String comment;

    @NotNull(message = "Category can't be empty")
    @Enumerated(value = EnumType.STRING)
    @Column(name = "category")
    private TransactionCategory category;

    @Column(name = "refill")
    private boolean refill;

    @PastOrPresent
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "date_time")
    private LocalDateTime dateTime;

    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

    public boolean isEmpty() {
        return sum == 0 || category == null;
    }

    @Override
    public Transaction clone() {

        try {
            Transaction clone = (Transaction) super.clone();
            clone.setSum(sum);
            clone.setDateTime(dateTime);
            clone.setCategory(category);
            clone.setRefill(refill);
            clone.setComment(comment);
            return clone;
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }


}
```

Висновок: В даній лабораторній роботі я реалізував частину проекту використавши шаблон проектування “Clonable”