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 №8

Теорія Розробки Програмного Забезпечення

Шаблон «Flyweight»

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 ***flyweight***.

Шаблон проектування "Flyweight" входить в категорію структурних шаблонів і використовується для оптимізації роботи з великою кількістю схожих об'єктів. Основна ідея полягає в тому, щоб спільно використовувати однакові частини об'єктів, замість того щоб кожен об'єкт зберігав свої власні копії цих частин. Це дозволяє економити пам'ять і прискорює виконання програми.

У шаблоні "Flyweight" виділяють два типи об'єктів: внутрішній та зовнішній. Внутрішні об'єкти є незмінними та можуть бути використані багатьма зовнішніми об'єктами. Зовнішні об'єкти, у свою чергу, зберігають лише ті дані, які вони спільно використовують, вказуючи на відповідний внутрішній об'єкт. Це сприяє зменшенню обсягу пам'яті, що використовується, і поліпшує продуктивність програми.

Шаблон "Flyweight" особливо ефективний у випадках, коли велика кількість об'єктів має спільні атрибути, і може бути використаний для оптимізації різних аспектів програмного коду, таких як графічні інтерфейси, обробка текстової інформації чи мережеві операції.

Реалізація в моєму проекту виправдана великою кількістю транзакцій у користувача які він може отримувати одночасно. Отже параметри які залежать один від одного варто винести в окремий клас.

Ось цей клас:

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

@Data
@AllArgsConstructor
public class TransactionType {

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

    private boolean refill;
}
```

Поле refill найчастіше є константою для певної category від категорії.

КласTransaction.

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


@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class Transaction {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

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

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

    private TransactionType transactionType;

    @PastOrPresent
    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime dateTime;

    private User user;

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

Тепер потрібно створити фабрику TransactionType з кешом.

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

public class TransactionTypeFactory {

    static Map<String,TransactionType> typeMap = new HashMap<>();

    public static TransactionType getTransactionType(TransactionCategory
category, boolean isRefill) {

        TransactionType result = typeMap.get(category.toString());
        if(result == null) {
            result = new TransactionType(category, isRefill);
            typeMap.put(category.toString(), result);
        }
        return result;
    }
}
```

Оскільки тепер в entity transaction є параметр об’єкт якого не відповідає таблиці в базі даних. Нам доведеться реалізувати зв’язок з бд інакше. Я обрав технологію JDBC Template.

Наведу приклад Transaction Dao.

```
package com.example.PersonalAccounting.dao;


@Component
public class TransactionDao {

    private final JdbcTemplate jdbcTemplate;

    private final TransactionRowMapper rowMapper;

    public TransactionDao(JdbcTemplate jdbcTemplate, TransactionRowMapper
rowMapper) {
        this.jdbcTemplate = jdbcTemplate;
        this.rowMapper = rowMapper;
    }


    public List<Transaction> index() {
        return jdbcTemplate.query("SELECT * FROM transaction", rowMapper);
    }

    public Transaction show(int id){
        return jdbcTemplate.query("SELECT * FROM transaction WHERE id=?",
rowMapper, id)
                .stream().findAny().orElse(null);
    }

    public void save(Transaction transaction) {
        jdbcTemplate.update(
                "INSERT INTO transaction(sum,
personal_accounting.transaction.comment, refill, date_time, user_id,
category)" +
                        " VALUES (?,?,?,?,?,?)",
                transaction.getSum(), transaction.getComment(),
transaction.getDateTime(),
                transaction.getTransactionType().getCategory(),
transaction.getTransactionType().isRefill(),
                transaction.getUser().getId());
    }

    public void update(int id, Transaction transaction) {
        jdbcTemplate.update("UPDATE transaction SET sum=?,
personal_accounting.transaction.comment=?, refill=?," +
                        " date_time=?, user_id=?, category=? WHERE id=?",
                transaction.getSum(), transaction.getComment(),
transaction.getDateTime(),
                transaction.getTransactionType().getCategory(),
transaction.getTransactionType().isRefill(),
                transaction.getUser().getId());
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE FROM personal_accounting.transaction
WHERE id=?", id);
    }
}
```

Клас що парсить результати запиту в entity:

```
package com.example.PersonalAccounting.dao.row_mapper;



@Component
public class TransactionRowMapper implements RowMapper<Transaction> {

    @Override
    public Transaction mapRow(ResultSet rs, int rowNum) throws SQLException {
        Transaction transaction = new Transaction();

        TransactionCategory category =
TransactionCategory.getCategoryByName(rs.getString("category"));
        TransactionType type = TransactionTypeFactory
                .getTransactionType(category, rs.getBoolean("refill"));

        transaction.setId(rs.getInt("id"));
        transaction.setSum(rs.getInt("sum"));
        transaction.setComment(rs.getString("comment"));
        transaction.setTransactionType(type);

transaction.setDateTime(LocalDateTime.parse(rs.getString("date_time")));
        transaction.setUser(new User(rs.getInt("user_id")));
        return null;
    }
}
```

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