Объектно-ориентированное программирование

Задание 1.

Необходимо реализовать проект удовлетворяющий изученному материалу. Проект содержит интерфейсы, реализован с использованием MVP паттерна, удовлетворяет всем принципам SOLID. Создать проект с записной книжкой. Идея в том, что пользователь может делать записи на различные дни (например в 19:00 стоматолог), читать записи, сохранять и загружать в файл, сортировать, искать записи на конкретный день или неделю.
Приложение должно быть консольным

Подсказка № 1

Соблюдайте принцип единственной ответственности. Каждый класс в вашем проекте должен иметь единую ответственность. Например, класс `Note` должен только хранить данные о записи, а класс `Notebook` должен управлять списком записей и обеспечивать логику работы с ними, не занимаясь выводом данных или взаимодействием с пользователем.

Подсказка № 2

Используйте интерфейсы для обеспечения гибкости. Интерфейс `NotebookView` позволяет отделить представление (вывод данных) от логики приложения. Это означает, что вы можете легко изменить реализацию представления (например, перейти от консольного ввода/вывода к графическому интерфейсу), не изменяя логику в классе `NotebookPresenter`.

Подсказка № 3

Реализуйте методы сортировки и фильтрации в модели. Методы для получения записей по дате и неделе (`getNotesForDay()` и `getNotesForWeek()`) должны быть реализованы в классе `Notebook`, так как это относится к бизнес-логике. Не забудьте, что сортировка должна учитывать временные метки, чтобы отображать записи в правильном порядке.

Подсказка № 4

Следуйте принципу инверсии зависимостей. В классе `NotebookPresenter` не должно быть прямых зависимостей от конкретных реализаций представления. Вместо этого используйте интерфейс `NotebookView`, что позволит вам легко заменить реализацию представления, если это потребуется.

Подсказка № 5

Правильно форматируйте даты и время при вводе и выводе данных. Используйте `DateTimeFormatter` для корректного преобразования `LocalDateTime` в строку и обратно. Убедитесь, что формат даты и времени соответствует ожидаемому формату при вводе и выводе данных.

Подсказка № 6

Проверьте правильность сериализации и десериализации данных. При загрузке записей из файла убедитесь, что строки правильно разбираются и преобразуются в объекты `Note`. Обратите внимание на возможные ошибки при разборе строки и форматировании даты и времени.

Эталонное решение:

1. Модель

Note.java:

```
import java.time.LocalDateTime;


public class Note {
    private LocalDateTime dateTime;
    private String description;


    public Note(LocalDateTime dateTime, String description) {
        this.dateTime = dateTime;
        this.description = description;
    }


    public LocalDateTime getDateTime() {
        return dateTime;
    }


    public String getDescription() {
```

```
        return description;
    }

    @Override
    public String toString() {
        return dateTime.toString() + ": " + description;
    }
}
```

Notebook.java:

```
import java.io.*;
import java.time.LocalDateTime;
import java.util.*;
import java.util.stream.Collectors;

public class Notebook {
    private List<Note> notes = new ArrayList<>();

    public void add(Note note) {
        notes.add(note);
    }

    public List<Note> getNotes() {
        return new ArrayList<>(notes);
    }

    public List<Note> getNotesForDay(LocalDateTime dateTime) {
```

```
        return notes.stream()
                .filter(note ->
note.getDateTime().toLocalDate().isEqual(dateTime.toLocalDate()))
                .sorted(Comparator.comparing(Note::getDateTime))
                .collect(Collectors.toList());
    }

    public List<Note> getNotesForWeek(LocalDateTime startOfWeek) {
        LocalDateTime endOfWeek = startOfWeek.plusWeeks(1);
        return notes.stream()
                .filter(note ->
!note.getDateTime().isBefore(startOfWeek) &&
!note.getDateTime().isAfter(endOfWeek))
                .sorted(Comparator.comparing(Note::getDateTime))
                .collect(Collectors.toList());
    }

    public void saveToFile(String fileName) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(fileName))) {
            for (Note note : notes) {
                writer.write(note.toString());
                writer.newLine();
            }
        }
    }

    public void loadFromFile(String fileName) throws IOException {
```

```
        notes.clear();

        try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {

            String line;

            while ((line = reader.readLine()) != null) {

                String[] parts = line.split(": ", 2);

                LocalDateTime dateTime =
LocalDateTime.parse(parts[0]);

                String description = parts[1];

                notes.add(new Note(dateTime, description));

            }

        }

    }

}
```

2. Представление

NotebookView.java:

```
import java.time.LocalDateTime;

import java.util.List;



public interface NotebookView {

    void showNotes(List<Note> notes);

    void showMessage(String message);

    LocalDateTime getDateTimeInput();

    String getDescriptionInput();

    String getFileNameInput();

}
```

3. Презентер

NotebookPresenter.java:

```
import java.io.IOException;

import java.time.LocalDateTime;

import java.util.List;


public class NotebookPresenter {

    private Notebook model;

    private NotebookView view;


    public NotebookPresenter(Notebook model, NotebookView view) {

        this.model = model;

        this.view = view;

    }


    public void addNote() {

        LocalDateTime dateTime = view.getDateTimeInput();

        String description = view.getDescriptionInput();

        model.add(new Note(dateTime, description));

        view.showMessage("Note added.");

    }


    public void showNotesForDay() {

        LocalDateTime dateTime = view.getDateTimeInput();

        List<Note> notes = model.getNotesForDay(dateTime);
```

```
        view.showNotes(notes);
    }

    public void showNotesForWeek() {
        LocalDateTime startOfWeek = view.getDateTimeInput();
        List<Note> notes = model.getNotesForWeek(startOfWeek);
        view.showNotes(notes);
    }

    public void saveNotes() {
        String fileName = view.getFileNameInput();
        try {
            model.saveToFile(fileName);
            view.showMessage("Notes saved to " + fileName);
        } catch (IOException e) {
            view.showMessage("Failed to save notes: " +
e.getMessage());
        }
    }

    public void loadNotes() {
        String fileName = view.getFileNameInput();
        try {
            model.loadFromFile(fileName);
            view.showMessage("Notes loaded from " + fileName);
        } catch (IOException e) {
```

```
            view.showMessage("Failed to load notes: " +
e.getMessage());

        }

    }

}
```

4. Основной класс приложения

Main.java:

```
import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;



public class Main {

    public static void main(String[] args) {

        Notebook model = new Notebook();

        NotebookView view = new ConsoleNotebookView();

        NotebookPresenter presenter = new NotebookPresenter(model,
view);



        Scanner scanner = new Scanner(System.in);



        while (true) {

            System.out.println("1. Add Note");

            System.out.println("2. Show Notes for Day");

            System.out.println("3. Show Notes for Week");

            System.out.println("4. Save Notes");

            System.out.println("5. Load Notes");
```

```
            System.out.println("6. Exit");


            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline


            switch (choice) {
                case 1:
                    presenter.addNote();
                    break;
                case 2:
                    presenter.showNotesForDay();
                    break;
                case 3:
                    presenter.showNotesForWeek();
                    break;
                case 4:
                    presenter.saveNotes();
                    break;
                case 5:
                    presenter.loadNotes();
                    break;
                case 6:
                    return;
                default:
                    System.out.println("Invalid choice");
            }
        }
```

```
    }
}
```

ConsoleNotebookView.java:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Scanner;


public class ConsoleNotebookView implements NotebookView {
    private Scanner scanner = new Scanner(System.in);


    @Override
    public void showNotes(List<Note> notes) {
        if (notes.isEmpty()) {
            System.out.println("No notes found.");
        } else {
            for (Note note : notes) {
                System.out.println(note);
            }
        }
    }


    @Override
    public void showMessage(String message) {
        System.out.println(message);
    }
```

```
    @Override
    public LocalDateTime getDateTimeInput() {
        System.out.println("Enter date and time
(yyyy-MM-dd'T'HH:mm):");
        String input = scanner.nextLine();
        DateTimeFormatter formatter =
DateTimeFormatter.ISO_LOCAL_DATE_TIME;
        return LocalDateTime.parse(input, formatter);
    }

    @Override
    public String getDescriptionInput() {
        System.out.println("Enter note description:");
        return scanner.nextLine();
    }

    @Override
    public String getFileNameInput() {
        System.out.println("Enter file name:");
        return scanner.nextLine();
    }
}
```