Объектно-ориентированное программирование

Задание 1.

Реализовать паттерн MVP в вашем проекте с семейным деревом

Подсказка № 1

Для реализации паттерна MVP (Model-View-Presenter) в проекте с генеалогическим древом, нужно разделить логику приложения на три основные компоненты:
1. Model. Модель, которая представляет данные и бизнес-логику. В данном случае это класс FamilyTree и связанные с ним классы, такие как Person.
2. View. Представление, которое отвечает за отображение данных и взаимодействие с пользователем. В консольном приложении это может быть интерфейс или класс, который обрабатывает ввод и вывод данных.
3. Presenter. Презентер, который взаимодействует с моделью и представлением. Презентер получает ввод от представления, обрабатывает его с помощью модели, и затем обновляет представление.

Подсказка № 2

Создание интерфейса для View. Этот интерфейс определяет методы, которые будут использоваться для взаимодействия с пользователем.

Подсказка № 3

Создание Presenter. Презентер будет содержать логику взаимодействия между моделью и представлением.

Подсказка № 4

Модификация Main класса для использования Presenter. В `Main` нужно инициализировать модель, представление и презентер.

Эталонное решение:

Пакет `view`:

1. **FamilyTree.java:**

```
package view;



import model.Person;


```

```
import java.util.List;


public interface TreeView {
    void displayMessage(String message);
    void displayPersons(List<Person> persons);
    String getUserInput();
    void setPresenter(TreePresenter presenter);
}
```

2. **ConsoleTreeView.java:**

```
package view;


import model.Person;
import presenter.TreePresenter;


import java.util.List;
import java.util.Scanner;


public class ConsoleTreeView implements TreeView {
    private TreePresenter presenter;
    private Scanner scanner;


    public ConsoleTreeView() {
        this.scanner = new Scanner(System.in);
    }


    @Override
```

```
    public void displayMessage(String message) {
        System.out.println(message);
    }

    @Override
    public void displayPersons(List<Person> persons) {
        for (Person person : persons) {
            System.out.println(person.getName() + ", born in " +
person.getBirthYear());
        }
    }

    @Override
    public String getUserInput() {
        return scanner.nextLine();
    }

    @Override
    public void setPresenter(TreePresenter presenter) {
        this.presenter = presenter;
    }
}
```

Пакет presenter

1. **TreePresenter.java:**

```
package presenter;
```

```
import model.FamilyTree;

import model.Person;

import service.FileOperations;

import view.TreeView;



import java.io.IOException;



public class TreePresenter {

    private FamilyTree<Person> familyTree;

    private TreeView view;

    private FileOperations<Person> fileOperations;



    public TreePresenter(FamilyTree<Person> familyTree, TreeView
view, FileOperations<Person> fileOperations) {

        this.familyTree = familyTree;

        this.view = view;

        this.fileOperations = fileOperations;

        this.view.setPresenter(this);

    }



    public void addPerson(String name, int birthYear) {

        Person person = new Person(name, birthYear);

        familyTree.addMember(person);

        view.displayMessage("Person added: " + name);

    }



    public void showAllPersons() {
```

```
        view.displayPersons(familyTree.getMembers());
    }

    public void sortPersonsByName() {
        familyTree.sortByName();
        view.displayMessage("Persons sorted by name:");
        showAllPersons();
    }

    public void sortPersonsByBirthYear() {
        familyTree.sortByBirthYear();
        view.displayMessage("Persons sorted by birth year:");
        showAllPersons();
    }

    public void saveTree(String fileName) {
        try {
            fileOperations.saveToFile(familyTree, fileName);
            view.displayMessage("Family tree saved to " + fileName);
        } catch (IOException e) {
            view.displayMessage("Error saving family tree: " +
e.getMessage());
        }
    }

    public void loadTree(String fileName) {
        try {
```

```
            familyTree = fileOperations.loadFromFile(fileName);

            view.displayMessage("Family tree loaded from " +
fileName);

        } catch (IOException | ClassNotFoundException e) {

            view.displayMessage("Error loading family tree: " +
e.getMessage());

        }

    }



    public void handleUserInput() {

        while (true) {

            view.displayMessage("Enter command (add, list,
sortByName, sortByBirthYear, save, load, exit):");

            String command = view.getUserInput();



            switch (command) {

                case "add":

                    view.displayMessage("Enter name:");

                    String name = view.getUserInput();

                    view.displayMessage("Enter birth year:");

                    int birthYear =
Integer.parseInt(view.getUserInput());

                    addPerson(name, birthYear);

                    break;

                case "list":

                    showAllPersons();

                    break;

                case "sortByName":
```

```
                    sortPersonsByName();
                    break;
                case "sortByBirthYear":
                    sortPersonsByBirthYear();
                    break;
                case "save":
                    view.displayMessage("Enter file name:");
                    saveTree(view.getUserInput());
                    break;
                case "load":
                    view.displayMessage("Enter file name:");
                    loadTree(view.getUserInput());
                    break;
                case "exit":
                    return;
                default:
                    view.displayMessage("Unknown command");
            }
        }
    }
}
```

Пакет `main`:

1. **Main.java:**

```
package main;


import model.FamilyTree;
import model.Person;
```

```
import presenter.TreePresenter;

import service.FileOperationsImpl;

import view.ConsoleTreeView;



public class Main {

    public static void main(String[] args) {

        FamilyTree<Person> familyTree = new FamilyTree<>();

        ConsoleTreeView view = new ConsoleTreeView();

        FileOperationsImpl<Person> fileOperations = new
FileOperationsImpl<>();



        TreePresenter presenter = new TreePresenter(familyTree,
view, fileOperations);



        presenter.handleUserInput();

    }

}
```