Объектно-ориентированное программирование

Задание 1.

Дополнить проект методами записи в файл и чтения из файла. Для этого создать отдельный класс и реализовать в нем нужные методы. Для данного класса сделайте интерфейс, который и используйте в своей программе. Пример работы с интерфейсом Serialazable можно найти в материалах к уроку.

Подсказка № 1

Начните с создания интерфейса, который будет содержать методы для сохранения и загрузки данных. Это поможет вам отделить логику работы с файлами от основной логики приложения и сделает код более гибким для дальнейшего расширения или изменения.

Подсказка № 2

Используйте сериализацию для сохранения объектов. Класс `Person` и `FamilyTree` должны реализовывать интерфейс `Serializable`. Это позволит сохранять объекты в файл и загружать их обратно, сохраняя их структуру. Не забудьте добавить поле `serialVersionUID`, чтобы избежать проблем при десериализации.

Подсказка № 3

Реализуйте методы интерфейса в отдельном классе. Создайте класс, который будет реализовывать методы интерфейса для записи и чтения из файла. В методе `saveToFile` используйте `ObjectOutputStream`, чтобы сериализовать и сохранить объект в файл, а в методе `loadFromFile` используйте `ObjectInputStream` для десериализации.

Подсказка № 4

Проверьте обработку исключений. Операции записи и чтения из файла могут вызвать исключения, такие как `IOException` и `ClassNotFoundException`. Убедитесь, что эти исключения правильно обрабатываются в вашем коде.

Эталонное решение:

```
import java.io.*;

import java.util.ArrayList;

import java.util.List;
```

```
// Интерфейс для операций с файлами

interface FileOperations {

    void saveToFile(FamilyTree familyTree, String fileName) throws
IOException;

    FamilyTree loadFromFile(String fileName) throws IOException,
ClassNotFoundException;

}



// Реализация интерфейса для операций с файлами

class FileOperationsImpl implements FileOperations {



    @Override

    public void saveToFile(FamilyTree familyTree, String fileName)
throws IOException {

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {

            oos.writeObject(familyTree);

        }

    }



    @Override

    public FamilyTree loadFromFile(String fileName) throws
IOException, ClassNotFoundException {

        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {

            return (FamilyTree) ois.readObject();

        }

    }

}
```

```
// Класс, представляющий человека
class Person implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private int birthYear;
    private Person mother;
    private Person father;
    private List<Person> children;

    public Person(String name, int birthYear) {
        this.name = name;
        this.birthYear = birthYear;
        this.children = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public int getBirthYear() {
        return birthYear;
    }

    public void setMother(Person mother) {
        this.mother = mother;
    }
```

```
    public void setFather(Person father) {
        this.father = father;
    }

    public void addChild(Person child) {
        this.children.add(child);
    }

    public List<Person> getChildren() {
        return children;
    }

    public Person getMother() {
        return mother;
    }

    public Person getFather() {
        return father;
    }
}

// Класс, представляющий генеалогическое древо
class FamilyTree implements Serializable {
    private static final long serialVersionUID = 1L;
    private List<Person> people;
```

```
    public FamilyTree() {
        this.people = new ArrayList<>();
    }

    public void addPerson(Person person) {
        this.people.add(person);
    }

    public List<Person> getChildren(Person parent) {
        return parent.getChildren();
    }

    public Person findPersonByName(String name) {
        for (Person person : people) {
            if (person.getName().equals(name)) {
                return person;
            }
        }
        return null;
    }

    public List<Person> getPeople() {
        return people;
    }
}

// Главный класс с точкой входа
```

```
public class Main {

    public static void main(String[] args) {

        FamilyTree familyTree = new FamilyTree();


        // Создаем людей

        Person john = new Person("John", 1950);

        Person mary = new Person("Mary", 1955);

        Person susan = new Person("Susan", 1980);


        // Устанавливаем родительские связи

        susan.setMother(mary);

        susan.setFather(john);

        john.addChild(susan);

        mary.addChild(susan);


        // Добавляем людей в древо

        familyTree.addPerson(john);

        familyTree.addPerson(mary);

        familyTree.addPerson(susan);


        // Создаем объект для работы с файлами

        FileOperations fileOps = new FileOperationsImpl();


        // Сохраняем генеалогическое древо в файл

        try {

            fileOps.saveToFile(familyTree, "familyTree.dat");

            System.out.println("Family tree saved to file.");
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Загружаем генеалогическое древо из файла
        FamilyTree loadedFamilyTree = null;
        try {
            loadedFamilyTree =
fileOps.loadFromFile("familyTree.dat");
            System.out.println("Family tree loaded from file.");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        // Проверяем, что древо загрузилось правильно
        if (loadedFamilyTree != null) {
            for (Person person : loadedFamilyTree.getPeople()) {
                System.out.println("Loaded person: " +
person.getName() + ", born in " + person.getBirthYear());
            }
        }
    }
}
```