Объектно-ориентированное программирование

Задание 1.

Продолжаем работать с проектом с семейным деревом. Реализовать интерфейс Iterable для дерева. Создать методы сортировки списка людей перед выводом, например по имени или по дате рождения (не менее 2). Создать пакетную структуру для проекта

Подсказка № 1

Для того чтобы класс `FamilyTree` поддерживал итерацию, реализуйте интерфейс `Iterable<Person>` и переопределите метод `iterator()`. В этом методе вы можете вернуть итератор списка `people`, что позволит использовать объекты `FamilyTree` в циклах `for-each`.

Подсказка № 2

Реализуйте методы `sortByName()` и `sortByBirthYear()` в классе `FamilyTree` для сортировки списка `people`. Используйте класс `Collections` и методы `sort`, передавая компараторы для сортировки по имени или году рождения. Это позволит вам легко сортировать данные перед их выводом.

Подсказка № 3

Разделите классы по функциональным пакетам, например, `model` для моделей данных, `service` для классов, реализующих функциональность, и `main` для точки входа. Это улучшит читаемость и поддерживаемость кода.

Подсказка № 4

После вызова методов сортировки (например, `sortByName()` или `sortByBirthYear()`), проверьте, что список людей действительно отсортирован. Используйте `for-each` цикл для вывода данных, чтобы убедиться, что сортировка прошла успешно.

Эталонное решение:

Пакет `model:`

1. **Person.java:**

```
package model;
```

```
import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;


public 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;
    }
}
```

2. **FamilyTree.java:**

```
package model;
```

```
import java.io.Serializable;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.List;


public class FamilyTree implements Serializable, Iterable<Person> {

    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;

    }



    @Override

    public Iterator<Person> iterator() {

        return people.iterator();

    }



    public void sortByName() {

        Collections.sort(people, (p1, p2) ->
p1.getName().compareTo(p2.getName()));

    }



    public void sortByBirthYear() {

        Collections.sort(people, (p1, p2) ->
Integer.compare(p1.getBirthYear(), p2.getBirthYear()));

    }

}
```

Пакет `service`:

1. **FileOperations.java:**

```
package service;
```

```
import model.FamilyTree;


import java.io.IOException;


public interface FileOperations {

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

    FamilyTree loadFromFile(String fileName) throws IOException,
ClassNotFoundException;

}
```

2. **FileOperationsImpl.java:**

```
package service;


import model.FamilyTree;


import java.io.*;


public 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();
        }
    }
}
```

Пакет `main`:

1. **Main.java:**

```
package main;

import model.FamilyTree;
import model.Person;
import service.FileOperations;
import service.FileOperationsImpl;

import java.io.IOException;

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);

        // Сортируем по имени
        System.out.println("Сортировка по имени:");
        familyTree.sortByName();
        for (Person person : familyTree) {
            System.out.println(person.getName() + " - " +
person.getBirthYear());
        }

        // Сортируем по дате рождения
        System.out.println("\nСортировка по дате рождения:");
        familyTree.sortByBirthYear();
        for (Person person : familyTree) {
```

```
            System.out.println(person.getName() + " - " +
person.getBirthYear());
        }

        // Сохраняем генеалогическое древо в файл
        FileOperations fileOps = new FileOperationsImpl();
        try {
            fileOps.saveToFile(familyTree, "familyTree.dat");
            System.out.println("\nFamily 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) {
            System.out.println("\nLoaded persons:");
            for (Person person : loadedFamilyTree) {
```

```
                System.out.println(person.getName() + ", born in " +
person.getBirthYear());

            }

        }

    }

}
```