Занятие 6. 26.02.2024

Задача 1

Пусть задан класс `Student`, который можно описать записью:

```
record Student (String name, int age) { }
```

Пусть вам потребовалось отсортировать список студентов `List<Student>` по имени. Для этого можно использовать метод `.sort`. Однако, метод `.sort` использует для сортировки либо естественный порядок, если класс реализует интерфейс `Comparable<T>`, либо требует в качестве параметра реализацию интерфейса `Comparator<T>`. Так как класс `Student` не реализует естественный порядок, то следует реализовать интерфейс `Comparator`. Напишите статический метод для сортировки студентов по имени, реализовав нужный компаратор с помощью анонимного класса:

```
public static List<Student> sortByName(List<Student> students);
```

Задача 2

Реализуйте сортировку списка элементов `Student` по возрасту, передав в метод `.sort` реализацию `Comparator` с использованием лямбда-выражения.

```
public static List<Student> sortByAge(List<Student> students);
```

Задача 3

Анонимные классы одновременно являются вложенными нестатическими классами (`Inner Classes`). Такие классы связываются с родительскими классами, становясь объектами-компаньонами для тех объектов включающего класса (`Outer Class`), в которых произошло создание объекта анонимного класса. Таким образом, они имеют доступ к внутренним данным включающего класса. Проиллюстрируем это.
Пусть задан класс `StudentGroup`, моделирующий группу студентов:

```
final class StudentGroup {
    public StudentGroup(List<Student> students) {
        this.students = students;
    }

    StudentGroup addStudent(Student s) {
        this.students.add(s);
        return this;
    }

    Iterable<String> overNames()  {
        // To Do
    }
    private final List<Student> students;
}
```

Нееобходимо реализовать с помощью анонимных классов (и/или лямбда-выражений) метод `overNames()`, который возвращает объект реализующий интерфейс `Iterable<String>` таким образом, чтобы при обходе этого Iterable возвращались имена студентов, а не сами объекты `Student`. Кроме того, такой итератор от Iterable можно было бы брать много раз. Пример:

```
var g = new Group(new ArrayList<>());
g.addStudent(new Student("Ann", 18));
g.addStudent(new Student("Sam", 19));
g.addStudent(new Student("Mary", 22));
g.addStudent(new Student("Boris", 17));
var names = g.overNames();
var it1 = names.iterator();
var it2 = names.iterator();
while (it1.hasNext()) {
    System.out.println(it1.next().equals(it2.next()));
}
System.out.println(it1.hasNext() == it2.hasNext());
for (var name : names) {
    System.out.println(name);
}
```

Результат:

```
true
true
true
true
```

```
true
Ann
Sam
Mary
Boris
```