Практическая работа № 10. Стандартные интерфейсы Джава. Интерфейс Comparator

Цель: цель данной практической работы - закрепить знания в области использования стандартных интерфейсов языка Джава, научиться применять интерфейсы для разработки практических программ на Джаве

Теоретические сведения

Comparator и Comparable в Java

Интерфейс Comparable содержит один единственный метод int compareTo(E item), который сравнивает текущий объект с объектом, переданным в качестве параметра. Если этот метод возвращает отрицательное число, то текущий объект будет располагаться перед тем, который передается через параметр. Если метод вернет положительное число, то, наоборот, после второго объекта. Если метод возвратит ноль, значит, оба объекта равны.

Интерфейс Comparator

Однако перед нами может возникнуть проблема, что, если разработчик не реализовал в своем классе, который мы хотим использовать, интерфейс Comparable, либо реализовал, но нас не устраивает его функциональность, и мы хотим ее переопределить? На этот случай есть еще более гибкий способ, предполагающий применение интерфейса Comparator<E>.

Интерфейс Comparator содержит ряд методов, ключевым из которых является метод compare():

Листинг 10.1 – Пример интерфейса Коимпаратор

```
public interface Comparator<E> {
    int compare(T a, T b);
    // остальные методы
}
```

Метод compare также возвращает числовое значение - если оно отрицательное, то объект a предшествует объекту b, иначе - наоборот. А если метод возвращает ноль, то объекты равны. Для применения интерфейса нам вначале надо создать класс компаратора, который реализует этот интерфейс:

Листинг 10.2 – Пример класса, реализующего интерфейс Компаратор

```
class PersonComparator implements Comparator<Person>{

    public int compare(Person a, Person b){

        return a.getName().compareTo(b.getName());
```

```
        }
    }
```

Замечание. Предполагаем, что тип Person у нас описан был ранее как класс

Листинг 10.3 – Пример класса Person

```
class Person{
    private String name;
    Person(String name){
        this.name=name;
    }
    String getName(){return name;}
}
```

Сортировка по нескольким критериям

Начиная с JDK 8 в механизм работы компараторов были внесены некоторые дополнения. В частности, теперь мы можем применять сразу несколько компараторов по принципу приоритета. Например, изменим класс Person следующим образом:

Листинг 10.4 – Пример класс Student

```
class Person{
 private String name;
 private int age;
 public Person(String  name, int age){
        this.name=name;
        this.age=age;
    }
 String getName(){return name;}
 int getAge(){return age;}
}
```

Здесь добавлено поле для хранения возраста пользователя. И, допустим, нам надо отсортировать пользователей по имени и по возрасту. Для этого определим два компаратора:

Листинг 10.5 – Пример реализующего компаратор

```
class PersonNameComparator implements
Comparator<Person>{
        public int compare(Person a, Person b){
            return a.getName().compareTo(b.getName());
        }
    }
```

Листинг 10.6 – Пример реализующего компаратор

```
    class PersonAgeComparator implements
Comparator<Person>{
        public int compare(Person a, Person b){

            if(a.getAge()> b.getAge())
                return 1;
            else if(a.getAge()< b.getAge())
                return -1;
            else
                return 0;
        }
    }
```

Интерфейс компаратора определяет специальный метод по умолчанию thenComparing, который позволяет использовать цепочки компараторов для сортировки набора:

Листинг 10.7 – Пример реализующего компаратор

```
    Comparator<Person> pcomp = new
PersonNameComparator().thenComparing(new
PersonAgeComparator());

    TreeSet<Person> people = new TreeSet(pcomp);
    people.add(new Person("Tom", 23));
    people.add(new Person("Nick",34));
    people.add(new Person("Tom",10));
    people.add(new Person("Bill",14));

    for(Person  p : people){

        System.out.println(p.getName() + " " +
p.getAge());
    }
```

Задания на практическую работу № 10

Задание 1. (5%)

Создать свой класс Student со всеми переменными экземпляра, конструктором, включающим все переменные, предпочтительно использовать

геттеры и сеттеры для каждой переменной. Класс студент имеет свойства: Имя, Фамилия, Специальность, Курс, Группа

Задание 2. (45%)

Напишите класс SortingStudentsByGPA (может у вас называться Tester или Main, так как содержит функцию main()) создайте поле как массив объектов Student с названием iDNumber, вы можете использовать как массив, так и и ArrayList или TreeSet для хранения данных о студентах

Добавьте методы класса: 1) заполнения массива setArray() 2) метод для сортировки по среднему баллу студентов quicksort() который реализует интерфейс Comparator таким образом, чтобы он сортировал студентов с их итоговым баллом в порядке убывания. В качестве алгоритма сортировки использовать методы сортировок: слиянием и быструю сортировку (добавьте в класс еще один метод). 3)метод для вывода массива студентов outArray()

4)Добавьте в класс возможность сортировать список студентов по другому полю

Задание 3. (50%)

Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках.