

Аннотация: этот документ является практическим руководством к Coding Bootcamp.

Week 1 Day 0

**Академия Ковалевского**

**Содержание**

[**1. Интерфейсы**](#_jd6ty15sdqeu) **2**

[Интерфейс Comparator](#_op97402fr90e) 3

[Рекомендации](#_dnmln7mujnsg) 3

[**2. Обобщения**](#_i3thh5s5xab3) **4**

[Рекомендации](#_vz5j0p9fc76a) 4

[**3. Практическая работа**](#_2kdd2ao9b0kp) **5**

[Пакет](#_6fx42t66hfqz) 5

[Класс Sorting](#_to3t9zhwvu4w) 5

[sort](#_4uejcv68eojj) 5

[sortReversedOrder](#_cas2qypsuom3) 5

[Класс PointWithValue](#_uhgq6gboxsug) 6

[Конструктор](#_7ktwfx4orgp5) 6

[getValue](#_ybrz9ph5u0en) 6

[mapPoint](#_7arjiaweuol0) 6

[toString](#_1c7qumv9y05m) 7

[Класс PointWithLabel](#_nyjcherdw6b9) 8

[Конструктор](#_kdxh0xy2u8u5) 8

[getLabel](#_w6l95qr44ed2) 8

[compareTo](#_ucm73g2gv0sk) 8

# 1. Интерфейсы

**Интерфейс** — это структура данных, которая может содержать поля, представленные в виде именованных констант и объявления методов. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может реализовать множество интерфейсов.

## Интерфейс Comparator

Рано или поздно появляется задача сравнения объектов по какому-либо принципу. Для этих целей существует интерфейс ***Comparator***. Он содержит ряд методов, ключевым из которых является метод ***compare(T o1, T o2)***, который должен быть реализован классом, имплементирующим компаратор. Этот метод возвращает числовое значение — если оно отрицательное, то объект o1 предшествует объекту o2, иначе — наоборот. А если метод возвращает ноль, то объекты равны.

## Рекомендации

* Найти информацию об интерфейсах в Java
* Посмотреть видео урок “[Интерфейсы в Java](https://youtu.be/CDa0INOc_1A)”
* Ответьте на вопрос: чем отличается компаратор от ***equals***

# 

# 2. Обобщения

**Обобщения** или **generics** (обобщенные типы и методы) позволяют нам уйти от жесткого определения используемых типов. С их помощью можно объявлять классы, интерфейсы и методы, где тип данных указан в виде параметра.

**Обобщения** — мощное дополнение к языку Java, поскольку оно облегчает работу программиста и снижает вероятность ошибок. Обобщения обеспечивают приведение типов во время компиляции и, что самое важное, позволяют реализовывать универсальные алгоритмы, не вызывая дополнительных затрат для приложений.

## Рекомендации

* Найти информацию об обобщениях
* Посмотреть видеоурок “[Что такое generic и какую проблему они решают](https://youtu.be/4Rpr7aeSzxA)”
* Посмотреть видео урок “[Знакомство с generic в Java](https://youtu.be/WPxcQtVWVKY)”
* Изучить общие понятия о функциональных интерфейсах и о способах их применения

# 3. Практическая работа

## Пакет

Нельзя использовать никакие импорты, кроме собственных классов из предыдущих уроков, *java.util.Comparator, java.util.function.Function*!

Все классы этого задания должны быть в пакете:

|  |
| --- |
| package com.kovalevskyi.academy.codingbootcamp.week1.day0 |

## Класс Sorting

### **sort**

Написать метод ***sort(T[] target, Comparator<T> comparator)***, который сортирует входящий массив неких элементов в порядке возрастания (ascending order) с помощью компаратора, который передается вторым аргументом.

|  |
| --- |
| *// сигнатура метода*  public static <T> void sort(T[] target, Comparator<T> comparator) {  *// TO DO*  } |

### **sortReversedOrder**

Написать метод ***sortReversedOrder(T[] target, Comparator<T> comparator),*** который сортирует входящий массив в порядке понижения (descending order) с помощью компаратора, который передается вторым аргументом.

|  |
| --- |
| *// сигнатура метода*  public static <T> void sortReversedOrder(T[] target, Comparator<T> comparator) {  *// TO DO*  } |

## 

## Класс PointWithValue

Данный класс наследуется от предыдущего класса Point и расширяет его, добавляя в точку какое-либо значение типа *Т*. Теперь наша точка может хранить в себе текст, который, например, указывает на то, что находится в этой точке. И теперь точку можно будет создавать вот так:

|  |
| --- |
| var newPoint = new PointWithValue<>(1, 2, "treasures"); |

В классе нужно реализовать следующее новые методы:

* новый конструктор
* getValue
* mapPoint
* toString

### **Конструктор**

Конструктор должен принимать, как и раньше, две координаты: X и Y. Только теперь он еще должен принимать третьим аргументом значение переменной, которое будет хранится:

|  |
| --- |
| public PointWithValue(int coordinateX, int coordinateY, T value) {   *// TO DO* } |

### **getValue**

Метод возвращает значение, которое хранится в этой точке:

|  |
| --- |
| *// сигнатура метода* public T getValue() {   *// TO DO* } |

### **mapPoint**

Этот метод принимает на вход функцию, а возвращает модифицированный PointWithValue. Точнее, точку с теми же координатами, но модифицированным *value*. Как именно будет производиться модификация *value* — знает только входящая функция.

Эта функция может быть полезна, если есть набор точек, у которых каждое значение — это число, которое хранится в виде строки (“12”). И если нужно преобразовать точку в новую, у которой значение будет хранится в виде **int***,* это можно сделать с помощью этого метода вот так:

|  |
| --- |
| var pointInput = new PointWithValue<>(1, 2, "12"); var mapFunction = (Function<String, Integer>) Integer::valueOf; var newPoint = pointInput.mapPoint(mapFunction); |

|  |
| --- |
| *// сигнатура метода*  public <R> PointWithValue<R> mapPoint(Function<T, R> mapFunction) {  *// TO DO*  } |

### **toString**

Возвращает текстовое представление точки, которое выглядит вот так:

***PointWithValue{ X: %d, Y: %d, value: %s}***

О переопределении этого метода вам уже известно из предыдущего Point.

## 

## Класс PointWithLabel

В данном классе мы наследуемся от предыдущего класса PointWithValue и на его базе сделаем конкретный класс для типа **String**. Этот класс будет иметь все те же методы, что и общий, но содержать еще:

* конструктор
* getLabel
* compareTo (Point other)

### **Конструктор**

Этот метод оставим без описания ;)

### **getLabel**

Возвращает строку, которая была сохранена при создании объекта.

### **compareTo**

Сравнивает строки двух точек. Но если входящий объект НЕ является типом PointWithLabel, то вызывает метод ***compareTo*** у предка.