

Аннотация: этот документ является практическим руководством к Coding Bootcamp.

Week 0 Day 3

**Академия Ковалевского**

**Содержание**

[**1. Экземпляры и классы**](#_eqqbfdqj2cli) **3**

[Рекомендации](#_xu6nsmdanrvd) 3

[**2. Поля класса**](#_7ur86wdj846s) **3**

[Статичиские поля](#_i264w7xfpl13) 3

[Рекомендации](#_qqb7iifqmcl0) 3

[**3. Конструкторы**](#_ep0vrn8bio1) **4**

[Рекомендации](#_fa0amfo6k4pd) 4

[**4. Методы доступа (getters)**](#_y5vn4rfy1l6j) **4**

[Рекомендации](#_z8xk0uyxtq7p) 4

[**5. Строковое представление объекта**](#_qsvb7wglbv3h) **5**

[Рекомендации](#_wkgtfwts1qs) 5

[**6. Неизменяемые объекты**](#_3ahmaop3ws79) **6**

[Рекомендации](#_n2f3lxtsjftk) 6

[**7. Сравнение объектов**](#_lfstfobulfka) **7**

[Метод equals()](#_l2zggtg4cq2e) 7

[Метод hashCode()](#_aj9x4jgyv9iw) 7

[Интерфейс Comparable](#_2uk351skpwkv) 7

[Рекомендации](#_xe8e7doaf1q7) 7

[**8. Практическая работа**](#_gsjcf8p02w4k) **8**

[Пакет](#_mpjjochptj21) 8

[Класс Point](#_185ogksjolwe) 8

[Point](#_3phj5kca5vx0) 8

[getX](#_11ro8k7xiyof) 8

[getY](#_d7otrdy01caj) 9

[sum](#_rj7ohme7nol6) 9

[updateX](#_ffxcrgct0zy8) 9

[updateY](#_dwnwcvtyh7si) 9

[distanceTo](#_fdt0brhsz81n) 10

[equals](#_tego2eu6bc1h) 10

[hashCode](#_y5urc69g89e9) 10

[toString](#_dxmjut9zxs49) 10

[compareTo](#_klx2979ytvdc) 11

# 1. Экземпляры и классы

**Класс** — это "шаблон для объекта", который Вы создаете в Java. Класс описывает свойства и методы, которые будут доступны у объекта, построенного по описанию, заложенному в классе.

**Экземпляр класса** (англ. instance) — это описание конкретного объекта в памяти. Экземпляры используются для представления (моделирования) конкретных сущностей реального мира. Понятие класса, как и понятие объекта, являются основой ООП.

## Рекомендации

* Найти информацию о экземплярах и классах
* Посмотреть видеоурок “Работа с экземплярами в Java”

# 2. Поля класса

**Поле класса** — это один из членов класса, предназначенный для хранения неких данных. Поля класса объявляются перед всеми методами. Данные в полях класса могут быть представлены различными типами из мира java.

## Статичиские поля

По аналогии с методами — это те поля класса, которые объявлены с ключевым словом **static**. Соответственно такие поля становятся общими для всех экземпляров (объектов) текущего класса. Статическими делают поля-константы.

## Рекомендации

* Найти информацию о полях классов в java

# 

# 3. Конструкторы

**Конструкторы** — это специальные методы, которые вызывается при создании объекта. Они "конструируют" новый объект определенного класса. В отличие от метода, конструктор никогда ничего не возвращает. Конструктор определяет действия, выполняемые при создании объекта класса, и является важной частью класса.

Как правило, программисты стараются явно указать конструктор. Если явного конструктора нет, то Java автоматически создаст его для использования по умолчанию.

## Рекомендации

* Найти информацию о конструкторах в Java
* Посмотреть видео урок “Конструкторы в Java”

# 4. Методы доступа (getters)

**Инкапсуляция** является базовой концепцией в ООП. Речь идет об обертывании данных и кода в виде единого блока. В этом случае рекомендуется объявлять переменные как private. Если переменная имеет уровень доступа private, к ней невозможно обратиться вне класса, в котором она объявлена. Но все равно необходим способ обращения к private переменным из другого класса, иначе такие изолированные переменные не будут иметь смысла. Это достигается с помощью объявления специальных public методов.

Методы, которые возвращают значение переменных **называются getters**.

## Рекомендации

* Найти информацию о getters и setters в Java

# 

# 5. Строковое представление объекта

Часто необходимо узнать содержимое того или иного объекта. Для этого в классе Object языка Java определен специальный метод **toString**(), который возвращает символьную строку, описывающую объект. При создании нового класса принято переопределение **toString()** таким образом, чтобы возвращающая строка содержала в себе имя класса, имена и значения всех переменных.

## Рекомендации

* Найти информацию о методе toString()

# 

# 6. Неизменяемые объекты

**Неизменяемый объект** (англ. immutable objects) — это объект, который не позволяет изменять свои параметры. А если вы все-же пытаетесь что-то изменить, то получаете новый объект. Но старый останется прежним. Неизменяемые объекты полезны тем, что они изначально ориентированы на многопоточность.

Типичным представителем неизменяемых объектов являются объекты класса **String** — любое изменение содержимого объекта приводит к созданию нового.

Чтобы поле или переменная были неизменяемыми — его обозначают ключевым словом **final**. Ссылка помеченная ключевым словом **final**:

* примитив не может изменить изначально заданное ему значение
* НЕпримитив не может сменить объект, на который он ссылается с момента инициализации

Для того чтобы сделать весь объект неизменяемым — все его поля должны быть **final**. Инициализация **final-полей** должна происходить только в месте их объявления или в конструкторе.

## Рекомендации

* Найти информацию о неизменяемых объектах в Java
* Посмотреть видео урок “[Неизменяемые объекты](https://youtu.be/z8bstpYRV9I)”
* Найти [информацию](https://dzone.com/articles/modifying-immutable-objects-with-chained-methods) о том как обновлять неизменяемые объекты

# 

# 7. Сравнение объектов

## Метод equals()

Мы используем метод **equals**() для сравнения объектов в Java. Чтобы определить, совпадают ли два объекта, equals() сравнивает значения атрибутов объектов. Может показаться, что оператор == и метод equals() делают то же самое. Но на самом деле они работают по-разному. Оператор == лишь сравнивает указывают ли две ссылки на один и тот же объект, а equals() должен сравнивать объекты по содержимому.

## Метод hashCode()

Мы используем метод **hashСode**() для оптимизации производительности при сравнении объектов. Выполнение hashСode() возвращает уникальный идентификатор для каждого объекта в программе. Что значительно облегчает реализацию. Если хэш-код объекта не совпадает с хэш-кодом другого объекта, нет причин для выполнения метода equals(). Вы просто будете знать, что два объекта не совпадают. Но если хэш-код одинаков, то нужно выполнить equals(), чтобы определить, совпадают ли значения и поля объектов.

## Интерфейс Comparable

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 возвращает ноль, значит, оба объекта равны.

## Рекомендации

* Найти [информацию](https://www.training.ru/#!/News/339?lang=ru) о сравнениях объектов в Java
* Найти [информацию](https://www.digizol.com/2008/07/java-sorting-comparator-vs-comparable.html) об интерфейсе Comparable
* Выясните и объясните друг другу разницу в применении == и equals, для сравнения объектов и примитивов

# 8. Практическая работа

**Нельзя** использовать никакие импорты, кроме собственных реализаций из прошлых уроков!

## Пакет

Все классы этого задания должны быть в пакете:

|  |
| --- |
| package com.kovalevskyi.academy.codingbootcamp.week0.day3 |

## Класс Point

Создать класс Point, который описывает точку на плоскости.

|  |
| --- |
| *// сигнатура класса*  public class Point implements Comparable<Point> {  *// ...*  } |

### Point

Создать конструктор, который принимает две координаты и сохраняет их в своих полях.

|  |
| --- |
| *// сигнатура конструктора*  public Point(final int coordinateX, final int coordinateY) {  *// TO DO*  } |

### getX

Написать метод **getX**(), который возвращает координату X, которая была передана конструктору.

|  |
| --- |
| *// сигнатура метода*  public int getX() {  *// TO DO*  } |

### getY

Написать метод **getY**(), который возвращает координату Y, которая была передана конструктору.

|  |
| --- |
| *// сигнатура метода*  public int getY() {  *// TO DO*  } |

### sum

Написать метод **sum**(final Point that), который возвращает новую точку. Координаты новой точки равны: координата X равна сумме координаты X текущей точки и координаты X входящей точки, а координата Y — сумме координаты Y текущей точки и координаты Y входящей точки.

|  |
| --- |
| *// сигнатура метода*  public Point sum(final Point that) {  *// TO DO*  } |

### updateX

Написать метод **updateX**(int newX), который создает новую координату: Y остается старой, X — обновляем на new X.

|  |
| --- |
| *// сигнатура метода*  public Point updateX(int newX) {  *// TO DO*  } |

### updateY

Написать метод **updateY**(int newY), который создает новую координату: X остается старой, Y — обновляем на new Y.

|  |
| --- |
| *// сигнатура метода*  public Point updateY(int newY) {  *// TO DO*  } |

### distanceTo

Написать метод **distanceTo**(Point that), который считает и возвращает расстояние от текущей точки A(X1, Y1) до входящей точки B(X2, Y2) по формуле: (A.X1 - B.X2)^2 + (A.Y1 - B.Y2)^2.

|  |
| --- |
| *// сигнатура метода*  public int distanceTo(Point that) {  *// TO DO*  } |

### equals

Написать метод **equals**(Object o), который сравнивает координаты текущей точки this с координатами входящего объекта o. Метод должен соответствовать лучшим практикам equals(), которые описаны здесь. Тема на Stack Overflow.

|  |
| --- |
| *// сигнатура метода*  @Override public boolean equals(Object o) {  *// TO DO*  } |

### hashCode

Написать метод **hashCode**(), который возвращает сумму координат X и Y текущей точки.

|  |
| --- |
| *// сигнатура метода*  @Override public int hashCode() {  *// TO DO*  } |

### toString

Написать метод **toString**(), который должен создавать строку: “Point{X: %d, Y: %d}”.

|  |
| --- |
| *// сигнатура метода*  @Override public String toString() {  *// TO DO*  } |

### compareTo

Написать метод **compareTo**(Point that), который вернет отрицательное число, если сумма координат X и Y входящей точки that больше суммы координат X и Y текущей точки this. В противном случае метод должен вернуть положительное целое число. Метод должен вернуть 0, если суммы равны.

|  |
| --- |
| *// сигнатура метода*  @Override  public int compareTo(Point that) {  *// TO DO*  } |