# CodingBootcamp

## Week 1 Day 1

Аннотация: этот документ является практическим руководством к Week 1 Day 1 в CodingBootcamp.

Академия Ковалевского

# Содержание

1. Экземпляры и классы ..... 2
2. Конструкторы ..... 3
3. Методы доступа (getters) ..... 4
4. Неизменяемые объекты ..... 5
5. Строковое представление объекта ..... 6
6. Сравнение объектов ..... 7
7. Практическая работа ..... 8

# 1. Экземпляры и классы

## Справка

### Определение

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. Конструкторы

## Справка

### Определение

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 видеоурок “Конструкторы в Java”

# 3. Методы доступа (getters)

## ℹ️ Справка

### Определение

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 извне 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

# 4. Неизменяемые объекты

## Справка

### Определение

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.

## Рекомендации

- Найти информацию о неизменяемых объектах в Java
- Посмотреть видеоурок “Неизменяемые объекты”
- Найти информацию о том как обновлять неизменяемые объекты

# 5. Строковое представление объекта

## Справка

### Определение

Часто необходимо узнать содержимое того или иного объекта. Для этого в классе `Object` языка Java определен специальный метод `toString()`, возвращающий 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. Сравнение объектов

## Справка

### 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.

### hashCode ()

Мы используем метод `hashCode()` для оптимизации производительности при сравнении объектов. Выполнение `hashCode()` 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 возвратит ноль, значит, оба объекта равны.

## Рекомендации

- Найти информацию о сравнениях объектов в Java
- Найти информацию об интерфейсе `Comparable`

# 7. Практическая работа

## Класс Point

### Класс Point

Создать класс `Point`, который описывает точку на плоскости.

Прототип класса:

```
1. public class Point implements Comparable<Point> {
2.   // ...
3. }
```

### Point

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

Прототип метода:

```
1. public Point(final int coordinateX, final int coordinateY);
```

### getX

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

Прототип метода:

```
1. public int getX();
```

### getY

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

Прототип метода:

```
1. public int getY();
```

## 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.

Прототип метода:

```
1. public Point sum(final Point that);
```

## updateX

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

Прототип метода:

```
1. public Point updateX(int newX);
```

## updateY

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

Прототип метода:

```
1. public Point updateY(int newY);
```

## 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: $\sqrt{(X1-X2)^2 + (Y1-Y2)^2}$.

Прототип метода:

```
1. public int distanceTo(Point that);
```

## 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.

Прототип метода:

```
1. @Override
2. public boolean equals(Object o);
```

### hashCode

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

Прототип метода:

```
1. @Override
2. public int hashCode();
```

### toString

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

Прототип метода:

```
1. @Override
2. public String toString();
```

### 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.

Прототип метода:

```
1. @Override
2. public int compareTo(Point that);
```

## Требования

### Запрещено использовать

Импорт пакетов.