

Аннотация: этот документ является практическим руководством к в Coding Bootcamp.

Week 1 Day 2

**Академия Ковалевского**

[**1. List**](#_lzwehs4eqg0u) **3**

[Рекомендации](#_qkvf5vawrkjl) 3

[**2. Практическая работа**](#_gqxts5hxfb6r) **4**

[Пакет](#_omdtqksmzc5t) 4

[Класс List](#_yil0yhx2g0y3) 4

[List](#_jibahdfuneif) 4

[createOne](#_ucgx8yf3xj4f) 5

[addToEnd](#_s2s14r96v8wt) 5

[addToStart](#_dobo3yiu19y) 5

[Contains](#_bnjpnudxgaam) 5

[Map](#_p2i9y0myo84t) 6

[Length](#_r9ewf8p32wrj) 6

[insertAfter](#_62z82m75izqp) 6

[insertAfter(T[] newValues)](#_568pswxnojns) 6

[Delete](#_3xteazmybnt2) 7

[Connect](#_r4c4vfhwmfqi) 7

[getNext](#_2ttxmnvcszgo) 7

[getPrev](#_9grenfed0nbn) 7

[setPrev](#_7jt20h9wun3d) 8

[setNext](#_bmwpkxtpc8d7) 8

[getValue](#_sxvsjbg0o7xt) 8

# 

# 1. List

**List** — это структура данных, которая позволяет хранить информацию в списке, а не, например, в массиве.

В мире Java **List** — это интерфейс, а его имплементаций существует несколько.

## Рекомендации

* Поищите различные реализации **List**
* Не старайтесь копировать готовые реализации

# 2. Практическая работа

Написать свою реализацию **List**, которая пройдет тесты. Изучите сигнатуры методов и подумайте это может стать двунаправленным LinkedList.

Совет: не старайтесь скопировать или написать по аналогии с уже реализованными двусвязными списками. Тут нужно изобрести свой двусвязный список, основываясь на выданном вам API.

**Нельзя** использовать внешние импорты, но можно использовать свои классы из прошлых дней и **Function**.

**Предупреждение:** если какой либо тест вывел просто *null* — значит было получено неожиданное исключение (постараемся выдать это исключение на экран, в обновлениях).

## Пакет

Класс должен быть в пакете:

|  |
| --- |
| package com.kovalevskyi.academy.codingbootcamp.week1.day2 |

## Класс List

Написать свой класс List и реализовать в нем все ниже описанные методы. Объект класса можно создавать только при помощи *new*. Каждый объект этого класса хранит в себе некое value.

Обратите внимание, что тестирование некоторых методов комплексное — методы взаимозависимы. Например *createOne* и *getValue*.

## List

Этот конструктор на вход принимает предыдущий и следующий элементы листа и текущее значение.

|  |
| --- |
| *// сигнатура конструктора*  private List(List<T> prev, List<T> next, T value) {   *// TODO* } |

## createOne

— создает новый объект со значением value.

|  |
| --- |
| *// сигнатура метода*  **public static <T> List<T> createOne(T value) {   *// TODO* }** |

## addToEnd

Этот метод добавляет новое значение в конец листа и возвращает ссылку на этот новый элемент листа.

|  |
| --- |
| *// сигнатура метода*  public static <T> List<T> addToEnd(List<T> list, T newValue) {  ***// TODO*** } |

## addToStart

Этот метод добавляет новое значение в начало листа и возвращает ссылку на этот новый элемент листа.

|  |
| --- |
| *// сигнатура метода*  public static <T> List<T> addToStart(List<T> list, T newValue) {  ***// TODO*** } |

## Contains

— возвращает true, если заданное значение присутствует в листе и false, если такого значения нет. `List<T> list` — любой случайный элемент листа.

|  |
| --- |
| *// сигнатура метода*  public static <T> boolean contains(List<T> list, T value) {  ***// TODO*** } |

## 

## Map

— создает новый лист равный по размеру текущему листу, в котором каждый элемент получен путем применения mapFunction к value элемента текущего листа.

Рекомендую почитать про обобщение методов в java.

|  |
| --- |
| *// сигнатура метода*  public static <T, R> List<R> map(List<T> list, Function<T, R> mapFunction) {  ***// TODO*** } |

## Length

— возвращает количество элементов в текущем листе. Независимо от того у какого элемента листа был вызван этот метод.

|  |
| --- |
| *// сигнатура метода*  public int length() {  ***// TODO*** } |

## insertAfter

— добавляет новый элемент листа после текущего и возвращает этот новый элемент.

|  |
| --- |
| *// сигнатура метода*  public List<T> insertAfter(T newValue) {  ***// TODO*** } |

## insertAfter(T[] newValues)

— добавляет новые элементы из входящего массива в текущий список, после текущего элемента списка.

|  |
| --- |
| *// сигнатура метода*  public void insertAfter(T[] newValues) {  ***// TODO*** } |

## Delete

— удаляет текущий элемент листа.

|  |
| --- |
| *// сигнатура метода*  public void delete() {  ***// TODO*** } |

## Connect

— связывает текущий элемент листа с входящим to.

|  |
| --- |
| *// сигнатура метода*  public void connect(List<T> to) {  *// TODO*  } |

## getNext

— возвращает следующий элемент листа.

|  |
| --- |
| *// сигнатура метода*  public List<T> getNext() {  *// TODO* } |

### getPrev

— возвращает предыдущий элемент листа.

|  |
| --- |
| *// сигнатура метода*  public List<T> getPrev() {  *// TODO* } |

### 

### setPrev

— устанавливает предыдущий элемент текущему элементу листа.

|  |
| --- |
| *// сигнатура метода*  public void setPrev(List<T> prev) {  *// TODO* } |

### setNext

— устанавливает следующий элемент текущему элементу листа.

|  |
| --- |
| *// сигнатура метода*  public void setNext(List<T> next) {  *// TODO* } |

### getValue

— возвращает значение текущего элемента листа.

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