Реализации интерфейса List

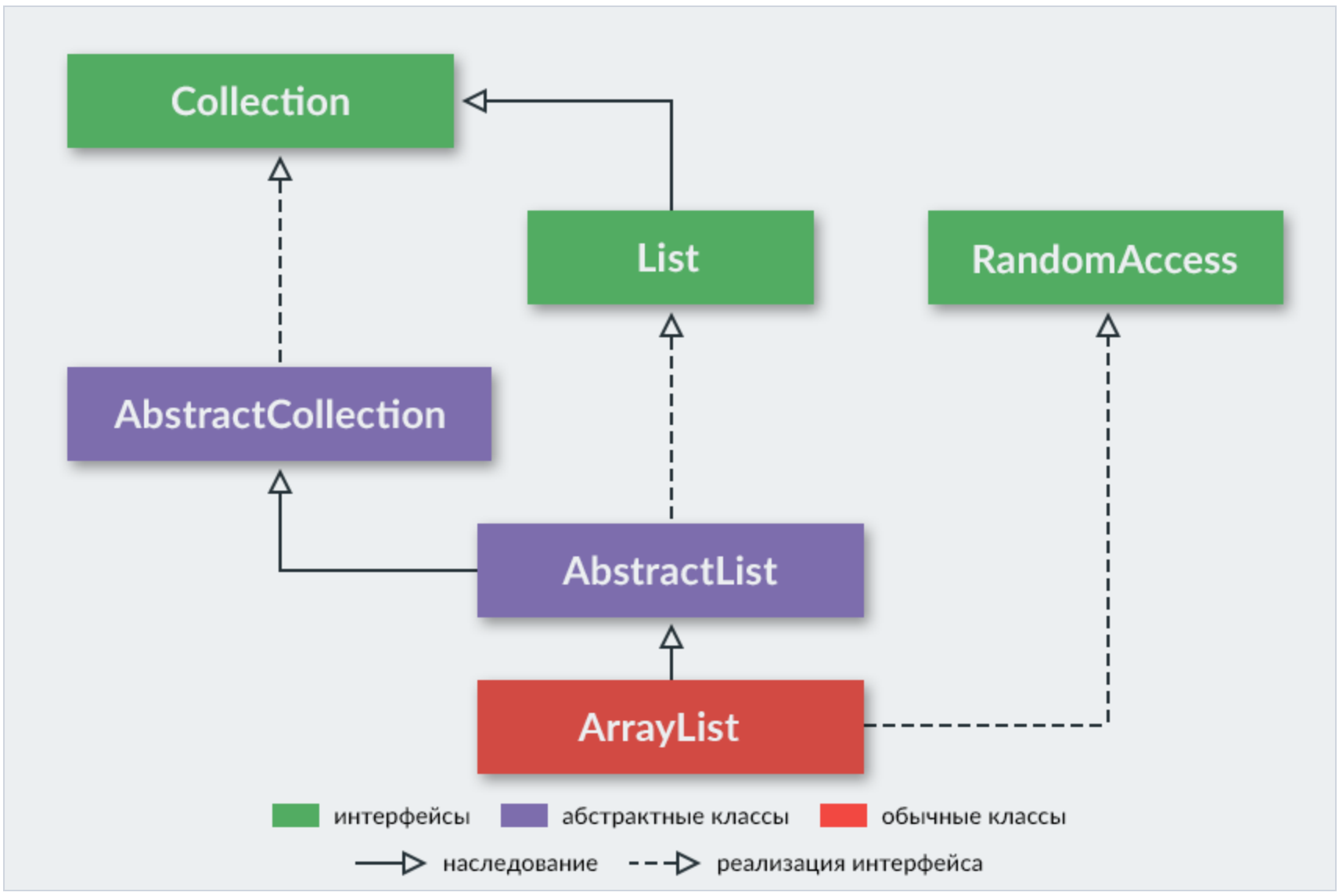
Java Collections
6 уровень, 5 лекция

ОТКРЫТА

— Если ты думаешь, что на интерфейсе List все закончилось, то ты ошибаешься, все только начинается. Давай я тебе расскажу про коллекции **LinkedList** и **ArrayList**.

Начну с коллекции ArrayList.

Вот как эта коллекция выглядит на схеме наследования:



Зеленым отмечены интерфейсы.

Фиолетовым – абстрактные классы.

Красным – обычные классы.

Сплошная линия – наследование, пунктирная – реализация интерфейса.

Это самая простая коллекция. Внутри **ArrayList** хранит элементы в простом массиве.

Основное преимущество такой коллекции над массивом – это расширяемость – увеличение длины при надобности.

Если в этом массиве заканчивается место, то создаётся второй массив побольше, куда копируются все элементы из первого. Затем второй массив занимает место первого, а первый – выбрасывается (будет уничтожен сборщиком мусора).

— А насколько увеличивается массив?

— Длина нового массива рассчитывается так (3*n)/2+1, где n – это длина старого массива. Т.е. если старый массив был длиной 100 элементов, то новый будет 300/2+1 = 151.

НАЧАТЬ ОБУЧЕНИЕ

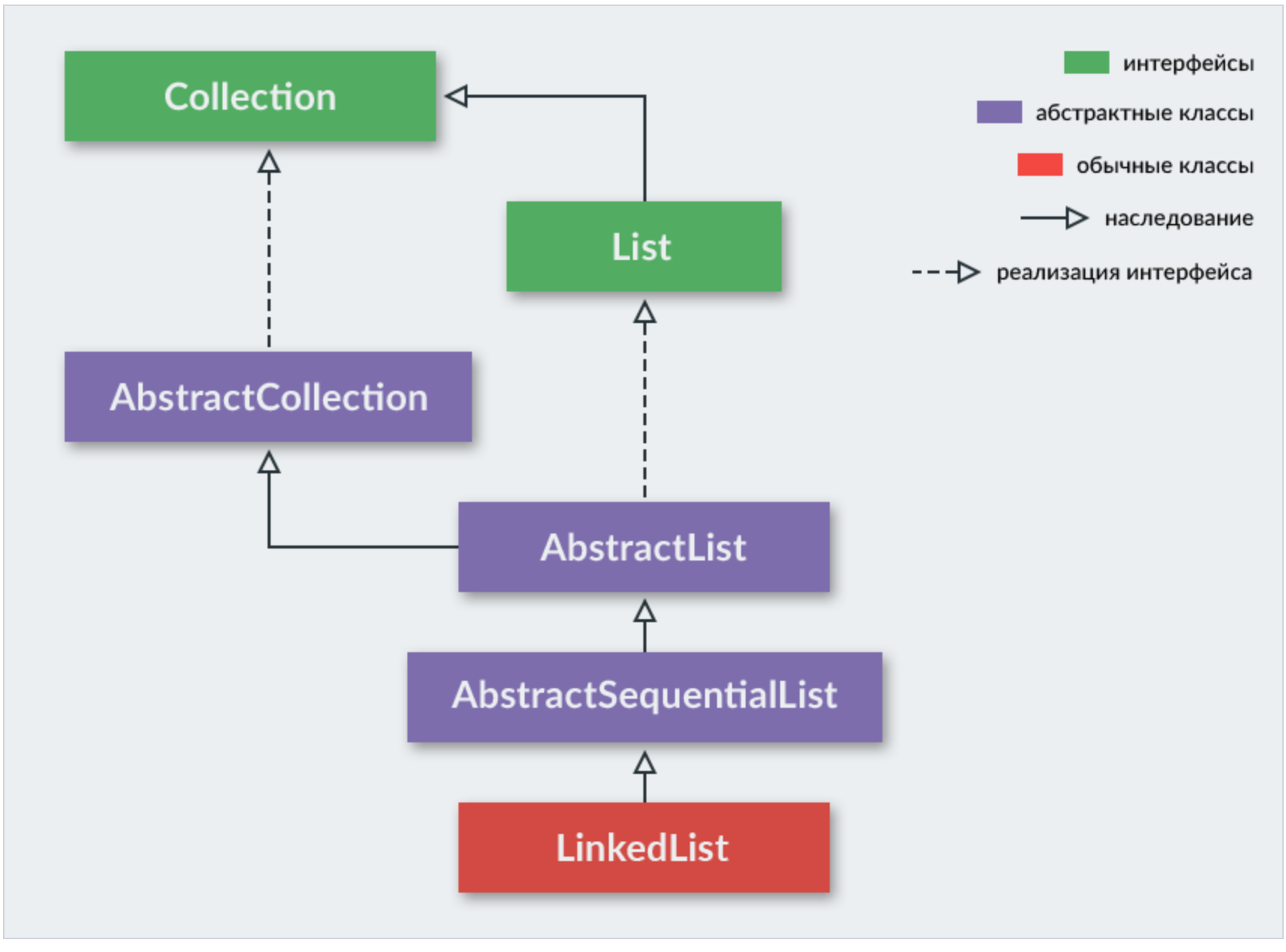
При удалении элемента из середины, все элементы справа от него копируются на 1 позицию влево.

— Ты говоришь, что ArrayList сам удлиняется, при добавлении элементов в него. А при удалении он сам укорачивается?

— Нет, сам внутренний массив в ArrayList никогда не укорачивается, но можно заставить ArrayList уменьшить свой внутренний массив до минимального уровня, если вызвать метод **trimToSize**().

Ну, и конечно, расскажу о LinkedList.

Вот как выглядит его схема наследования:



Зеленым отмечены интерфейсы.

Фиолетовым – абстрактные классы.

Красным – обычные классы.

Сплошная линия – наследование, пунктирная – реализация интерфейса.

Как ты уже знаешь, **LinkedList** хранит элементы в виде связного списка.

— Я постоянно об этом слышу, но не могла бы ты рассказать, что это такое?

— Конечно. Все просто.

Связный список состоит из одинаковых элементов, которые а) хранят данные, б) хранят ссылки на следующий и предыдущий элементы.

Вот так бы выглядел класс такого элемента для хранения строк:

Пример	Описание

`class LinkedListElement`
`{`
`String data;`
	data хранит строковое значение элемента.
	next хранит ссылку на следующий элемент в списке.
	previous хранит ссылку на предыдущий элемент в списке.

```
6   LinkedListElement previous;
    }
```

А если использовать generics для типа данных, то будет что-то типа такого:

Класс элемента связного списка с generic'ом

```
1   class LinkedListElement<T>
2   {
3    T data;
4    LinkedListElement<T> next;
5    LinkedListElement<T> previous;
6   }
```

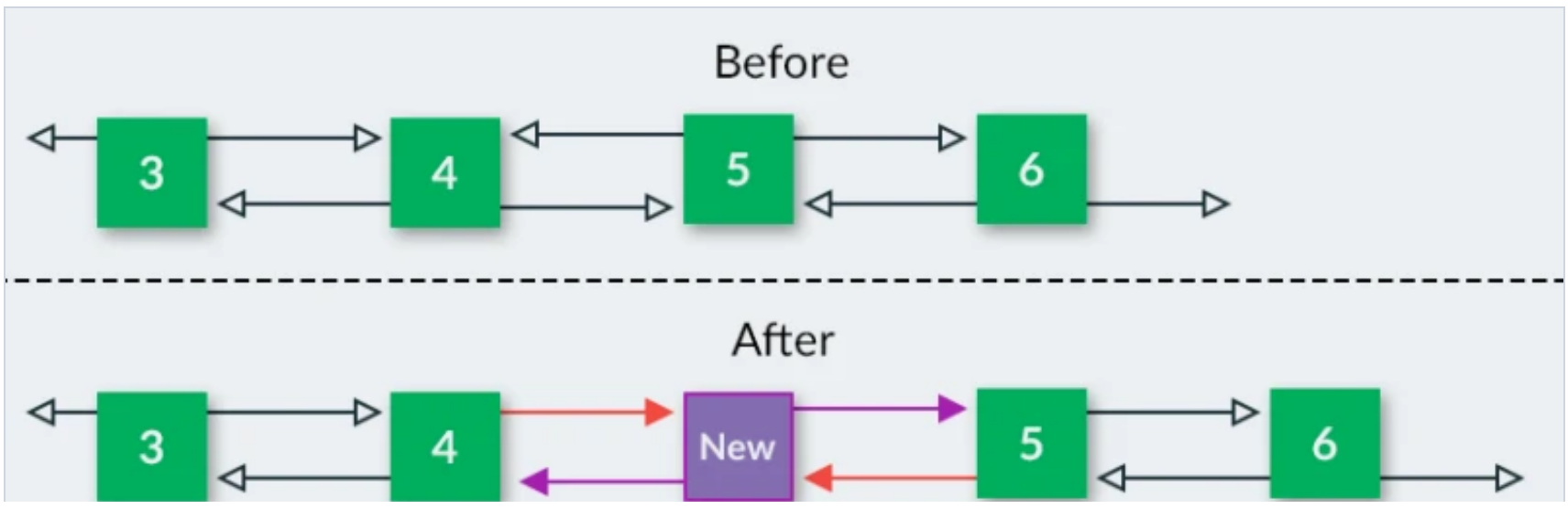
— Ясненько.

— Давай, для лучшего понимания, напишем код, где добавим в двусвязный список 10 элементов:

Пример

```
1   public static void main(String[] args)
2   {
3    LinkedListElement<Integer> tail; //хвост (самый последний элемент) списка
4
5    for(int i=0;i<10;i++)
6    {
7     LinkedListElement<Integer> element = new LinkedListElement<Integer>();
8     element.data = i;
9
10    if (tail == null) //если в хвосте нет элементов, сделать наш элемент последним
11    {
12     tail = element;
13    }
14    else //если хвост есть, добавить элемент
15    {
16     tail.next = element; //добавляем хвосту ссылку на следующий элемент
17     element.previous = tail; //добавляем новому элементу ссылку на хвост
18     tail = element; //объявляем новый элемент хвостом.
19    }
20   }
21  }
```

Допустим у нас в списке 10 элементов, вот как вставить элемент в середину:



Ярко-красным выделены ссылки, которые поменялись – они теперь указывают на новый элемент.

Ярко-фиолетовым выделены новые ссылки – ссылки нового элемента на его соседей.

А теперь – кодом:

Вставка элемента в середину связного списка

```
1  //тут содержится элемент – голова списка
2  LinkedListElement<Integer> head = …
3
4  //получаем 4-й элемент (нумерация с нуля)
5  LinkedListElement<Integer> element4 = head.next.next.next.next;
6  //получаем 5-й элемент
7  LinkedListElement<Integer> element5 = element4.next;
8
9  //Создаем новый элемент, который будем вставлять
10 LinkedListElement<Integer> newElement = new LinkedListElement<Integer>();
11 newElement.data = -18;
12
13 //обмениваемся ссылками с элементом слева
14 newElement.previous = element4;
15 element4.next = newElement;
16
17 //обмениваемся ссылками с элементом справа
18 newElement.next = element5;
19 element5.previous = newElement;
```

— Спасибо, Элли, действительно узнал о списках много нового.

‹ Предыдущая лекция

×26

+57

Комментарии (39) популярные новые старые

JavaCoder

Введите текст комментария

Dolivo Serg Уровень 37, Харьков 17 августа, 21:32

"Связный список состоит из одинаковых элементов..." Эти элементы (LinkedListElement) это простая структура данных, которую называют Node.

Ответить 0

Andrey Karelin Уровень 41, Sumy, Украина 3 мая, 16:55

И все это нам рассказывают уже после задач на деревья

Ответить 0

Panda Dzho Уровень 51 28 апреля, 21:49

Оставлю здесь ссыль на человека, который расписал очень доступно про основные структуры данных.
Тык

НАЧАТЬ ОБУЧЕНИЕ

Mykhailo_Trofimov Уровень 37, Poznan, Poland — 2 февраля, 22:39

И ни намека в картинке о implements Deque<E> LinkedList-ом. Мелочь, скажете вы, а на собесах цепляются.

Ответить — 0

Max Zap Уровень 41 — 11 ноября 2021, 09:53

На этом уровне читать про различные реализации List одно удовольствие)

Ответить — +1

Иван Уровень 41, Рязань, Россия — 7 сентября 2021, 17:37

Господа знатоки, подскажите почему List именно наследует (extends) интерфейс Collection, а не реализовывает (implements). Обычно наследуем классы. Что то я видимо упустил дойдя до 36 уровня

Ответить — 0

Кисмер Уровень 41, Санкт-Петербург — 8 сентября 2021, 16:29

интерфейсы расширяют (extends) интерфейсы
классы расширяют (extends) классы
классы реализуют (implements) интерфейсы

Ответить — +6

LuneFox инженер по сопровождению в **BIFIT** EXPERT — 9 февраля, 03:01

Интерфейс не может имплементировать другой интерфейс, потому что интерфейс это ещё не имплементация :)

Ответить — 0

Stepan Уровень 27, Москва, Россия — 20 марта, 23:42

Интерфейс по своей сути не может реализовывать другой интерфейс, а только расширять. Так же класс не может реализовывать другой класс, а только расширять.

Ответить — 0

VitalyK #1116124 Уровень 39, Рига, Латвия — 1 сентября 2021, 08:20

принцип понятен, не понятно как это может работать (или я что-то пропустил :)),
допустим "ссылка . ссылка"
tail.next = element;
или
LinkedListElement<Integer> element4 = head.next.next.next.next;

Ответить — 0

Сергей Java Developer в **Адвантум** — 30 июля 2021, 17:56

Зачем в предыдущей лекции про Collection ввели людей в заблуждение?
По схеме, которая там была ArrayList не наследуется от AbstractList. Да и LinkedList не соответствует.

Ответить — +1

Александр Черенков Уровень 37, Бердск, Россия — 18 мая 2021, 16:38

Судя по реализации классов , да и проверка:

показывает что формула длинны нового массива в ArrayList, будет max(n >> 1; 1) + n, а это на единицу меньше чем приведенная в лекции формула.
Судя по всему ошибка появилась из-за неправильного понимания вызова метода grow без параметров,

Но здесь задается, что новый массив будет больше старого на единицу, если не указано другое число (вызов grow(int minCapacity), что точно нужно когда длинна массива 0.
Реализация деления на 2 путем смещения вправо на один бит, определяет как будет происходить округление, поэтому использование в формуле коэффициента 1,5 неверно, так как это не покажет как округляется полученное значение, да и вообще это операции с целыми числами, а не дробными.
Само вычисление длинны нового массива осуществляется в методе newLength класса ArraysSupport

Ответить — +1

Андрей Лихтарович Уровень 40, Минск, Беларусь — 13 февраля 2021, 19:21

Длина нового массива рассчитывается так (3*n)/2+1, где n – это длина старого массива. Т.е. если старый массив был длиной 100 элементов, то новый будет 300/2+1 = 151
для тех кто подзабыл математику эта формула преобразовывается в (1,5* N + 1) так её легче запомнить а на собеседованиях такие вопросы часто встречаются!
Ставь лайк если сократил в уме при прочтении лекции)

Показать еще комментарии

RUSH

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.