

ТИНЬКОФФ

Модуль 2. Коллекции, Generics и Stream API

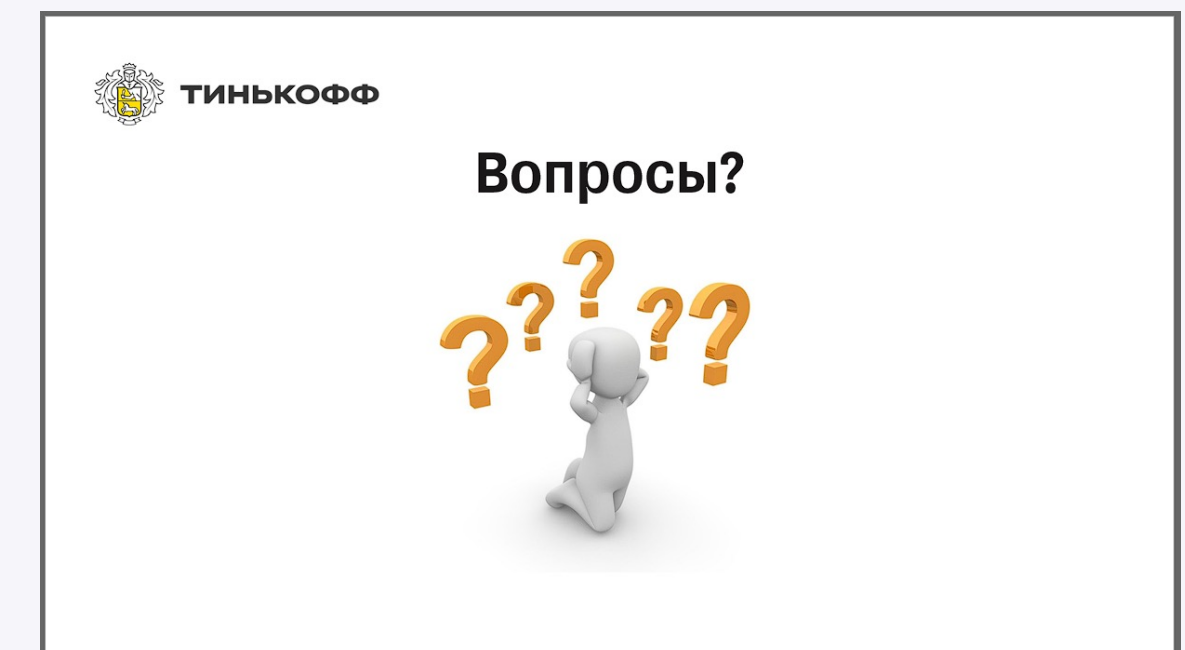
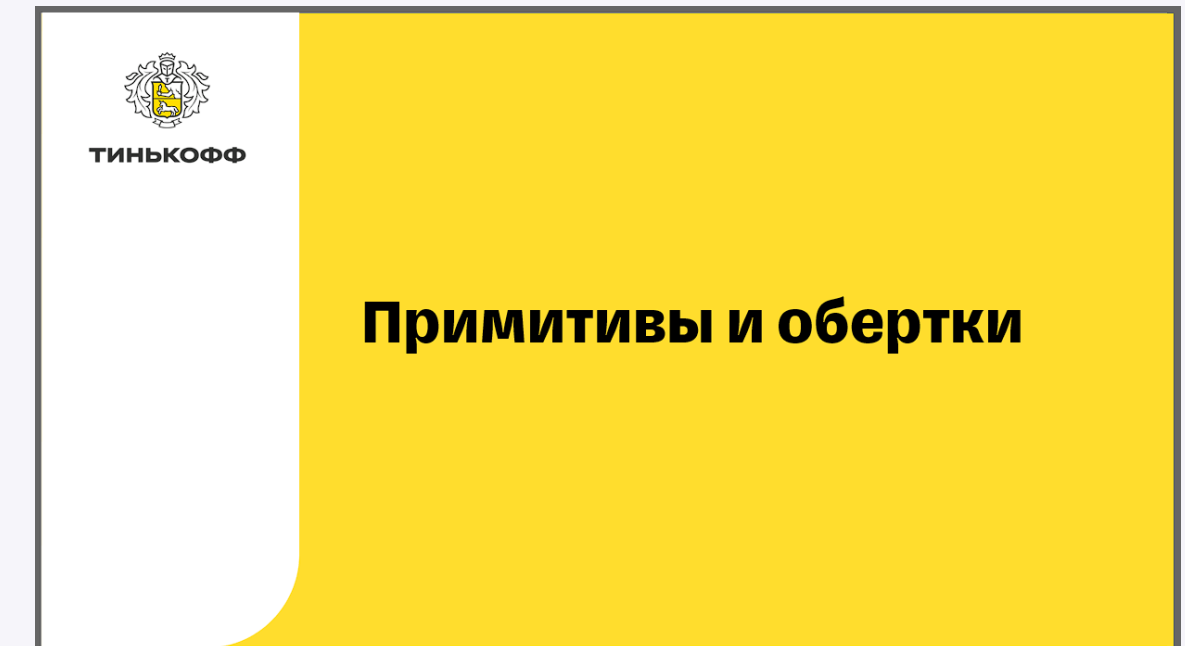
Лектор: Хватов Сергей



Коммуникация

- Во время чтения отдельных секций не вижу реакции и комментарии.
- Будут мини перерывы на каждом слайде-отбивке и слайде с вопросами.
- Можете задавать вопросы в чате толка.

Примеры слайдов с остановками



Кратко о себе



Образование

Закончил бакалавриат в СПбПУ Петра Великого в 2021, а магистратуру в ИТМО в 2023.



Карьера

Работаю Java/Kotlin разработчиком в Отделе Расчетных Технологий (ОРТ) с мая 2022. Имею более 3-х лет опыта коммерческой разработки на Java.



Преподавание

Проверяю ваши домашние задания и являюсь лектором 2 модуля “Коллекции, Generics и Stream API”.

Что было на прошлой лекции?

Познакомились с основными структурами данных

Рассмотрели иерархию интерфейсов и классов, а также основные структуры данных из JCF – List, Set, Map, Queue и Stack.

Узнали, что такое Generic'и

Изучили, как они работают и для чего нужны. Познакомились со стиранием типов и boxing'ом примитивов.

Что ждет нас сегодня?

01 Упорядоченные структуры данных

Узнаем, как можно сравнивать объекты в Java. Посмотрим на методы equals() и hashCode(). Изучим интерфейсы Comparable и Comparator, а также рассмотрим структуры данных, которые их используют.

03 Итераторы

Узнаем, как можно итерироваться по коллекциям, и как написать собственный итератор.

02 Утилитарные классы для работы с коллекциями

Посмотрим на функциональные возможности утилитарных классов Arrays и Collections, а также сторонние библиотеки, предназначенные для упрощения работы с коллекциями.

04 Лямбда-выражения

Рассмотрим реализацию функциональной парадигмы программирования в Java.

05 Stream API

Прикоснемся к Stream API и попробуем понять, что такое Stream'ы, с чем их едят и как правильно их готовить. Посмотрим на их аналоги из сторонних библиотек и языков программирования.

Сравнение объектов в Java

Метод equals()

Документация и полезные ссылки

- [Документация](#)
- [Статья про контракт метода equals\(\)](#)
- [Статья про генерацию методов equals и hashCode в Lombok](#)

Описание

Метод `equals()` - определяет отношение эквивалентности объектов.

При сравнении объектов с помощью `==` сравнение происходит лишь между ссылками. При сравнении с помощью переопределённого разработчиком метода `equals()` - по внутреннему состоянию объектов.

Условия, предъявляемые к методу equals()

Пусть даны два объекта `a` и `b` класса `T`, такие что `a != null` и `b != null`.

Реализация метода `equals()` в классе `T` должна удовлетворять следующим условиям:

- **Рефлексивность**

$$a.equals(a) == true$$

- **Симметричность**

$$a.equals(b) == b.equals(a)$$

- **Транзитивность**

$$(a.equals(b) \&\& b.equals(c)) == true \rightarrow a.equals(c) == true$$

- **Непротиворечивость**

$$a.equals(b) == true \rightarrow a.equals(c) == b.equals(c)$$

- **Идемпотентность** (при условии неизменности объектов)

Метод equals()



Пример реализации метода

```
1  @Override
2  public boolean equals(Object o) {
3      // Проверка на то, что обе ссылки
4      // ссылаются на один и тот же объект в памяти
5      if (o == this) {
6          return true;
7      }
8
9      // Проверка на то, что объекты одного и того же класса
10     if (!(o instanceof SomeClass)) {
11         return false;
12     }
13
14     // Проверка полей, используемых для определения идентичности
15     var other = (SomeClass) o;
16     return this.field1.equals(o.field1) && ...;
17 }
```


Метод hashCode()

Описание

Метод `hashCode()` позволяет посчитать хэш-объекта – целочисленное значение, которое может быть ассоциировано с каждым объектом в Java. Так как тип возвращаемого значения – `int`, значение `hashCode()` не уникально, и ограничено `Integer.MAX_VALUE`.

Условия, предъявляемые к методу hashCode()

Пусть даны два объекта `a` и `b` класса `T`, такие что `a != null` и `b != null`.

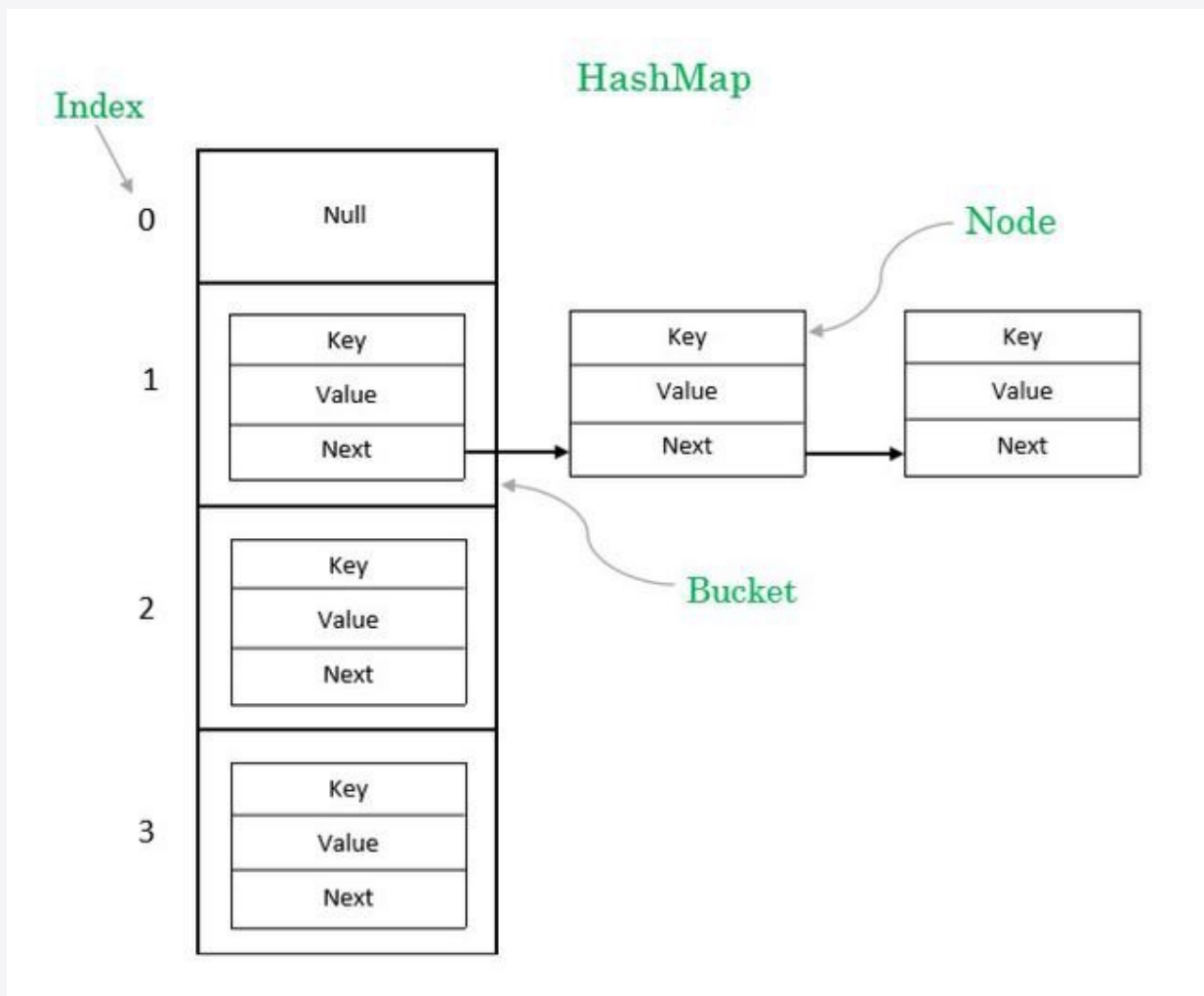
Реализация метода `hashCode()` в классе `T` должна удовлетворять следующему условию:

$$a.equals(b) \rightarrow a.hashCode() == b.hashCode()$$

Использование методов equals() и hashCode() в JCF

Документация и полезные ссылки

- [Internal Working of HashMap in Java](#)
- [HashMap in Java](#)
- [The Java HashMap Under the Hood](#)



© HashMap.java

Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced.

Params: key – key with which the specified value is to be associated
value – value to be associated with the specified key

Returns: the previous value associated with key, or null if there was no mapping for key. (A null return can also indicate that the map previously associated null with key.)

```
609 public V put(K key, V value) {  
610     return putVal(hash(key), key, value, false, true);  
611 }  
612
```

Implements Map.put and related methods.

Params: hash – hash for key
key – the key
value – the value to put
onlyIfAbsent – if true, don't change existing value
evict – if false, the table is in creation mode.

Returns: previous value, or null if none

```
623 final V putVal(int hash, K key, V value, boolean onlyIfAbsent,  
624               boolean evict) {  
625     Node<K,V>[] tab; Node<K,V> p; int n, i;  
626     if ((tab = table) == null || (n = tab.length) == 0)  
627         n = (tab = resize()).length;  
628     if ((p = tab[i = (n - 1) & hash]) == null)  
629         tab[i] = newNode(hash, key, value, null);  
630     else {  
631         Node<K,V> e; K k;  
632         if (p.hash == hash &&  
633             ((k = p.key) == key || (key != null && key.equals(k))))  
634             e = p;  
635         else if (p instanceof TreeNode)  
636             e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);  
637         else {  
638             for (int binCount = 0; ; ++binCount) {  
639                 if ((e = p.next) == null) {  
640                     p.next = newNode(hash, key, value, null);  
641                     if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st  
642                         treeifyBin(tab, hash);  
643                     break;  
644                 }  
645                 if (e.hash == hash &&  
646                     ((k = e.key) == key || (key != null && key.equals(k))))
```

Comparable

Документация и полезные ссылки

- [Документация](#)
- [Comparator & Comparable in Java](#)

Интерфейс



Контракт метода compareTo()

Пусть даны два объекта *a* и *b* класса *T*, такие что *a* **!=** null и *b* **!=** null.

Тогда, если:

- *a.compareTo(b)* == 0, то объекты равны
- *a.compareTo(b)* < 0, то объект *a* меньше объекта *b*
- *a.compareTo(b)* > 0, то объект *a* больше объекта *b*

При этом, реализация метода должна удовлетворять следующим условиям:

- **Рефлексивность**

$$a.compareTo(a) == 0$$

- **Симметричность**

$$\text{signum}(a.compareTo(b)) == -\text{signum}(b.compareTo(a))$$

- **Транзитивность**

$$(a.compareTo(b) > 0 \ \&\& \ b.compareTo(c) > 0) \rightarrow a.compareTo(c) > 0$$

- **Непротиворечивость**

$$a.compareTo(b) == 0 \rightarrow \text{signum}(a.compareTo(c)) == \text{signum}(b.compareTo(c))$$

- **Идемпотентность** (при условии неизменности объектов)

Comparable

Примеры реализации

Float

```
public static int compare(float f1, float f2) {  
    if (f1 < f2)  
        return -1;           // Neither val is NaN, thisVal is smaller  
    if (f1 > f2)  
        return 1;           // Neither val is NaN, thisVal is larger  
  
    // Cannot use floatToRawIntBits because of possibility of NaNs.  
    int thisBits    = Float.floatToIntBits(f1);  
    int anotherBits = Float.floatToIntBits(f2);  
  
    return (thisBits == anotherBits ? 0 : // Values are equal  
           (thisBits < anotherBits ? -1 : // (-0.0, 0.0) or (!NaN, NaN)  
           1));                          // (0.0, -0.0) or (NaN, !NaN)  
}
```

Enum

```
public final int compareTo(E o) {  
    Enum<?> other = (Enum<?>)o;  
    Enum<E> self = this;  
    if (self.getClass() != other.getClass() && // optimization  
        self.getDeclaringClass() != other.getDeclaringClass())  
        throw new ClassCastException();  
    return self.ordinal - other.ordinal;  
}
```


Comparator

Документация и полезные ссылки

- [Документация](#)

Интерфейс

@ FunctionalInterface

 Comparator<T>

| | | |
|---|---|---------------|
|   | <code>thenComparingDouble(ToDoubleFunction<T>)</code> | Comparator<T> |
|   | <code>reverseOrder()</code> | Comparator<T> |
|   | <code>comparing(Function<T, U>, Comparator<U>)</code> | Comparator<T> |
|   | <code>comparingLong(ToLongFunction<T>)</code> | Comparator<T> |
|   | <code>comparingDouble(ToDoubleFunction<T>)</code> | Comparator<T> |
|   | <code>thenComparing(Function<T, U>)</code> | Comparator<T> |
|   | <code>nullsLast(Comparator<T>)</code> | Comparator<T> |
|   | <code>equals(Object)</code> | boolean |
|   | <code>reversed()</code> | Comparator<T> |
|   | <code>thenComparingInt(ToIntFunction<T>)</code> | Comparator<T> |
|   | <code>thenComparingLong(ToLongFunction<T>)</code> | Comparator<T> |
|   | <code>naturalOrder()</code> | Comparator<T> |
|   | <code>thenComparing(Function<T, U>, Comparator<U>)</code> | Comparator<T> |
|   | <code>compare(T, T)</code> | int |
|   | <code>thenComparing(Comparator<T>)</code> | Comparator<T> |
|   | <code>nullsFirst(Comparator<T>)</code> | Comparator<T> |
|   | <code>comparing(Function<T, U>)</code> | Comparator<T> |
|   | <code>comparingInt(ToIntFunction<T>)</code> | Comparator<T> |

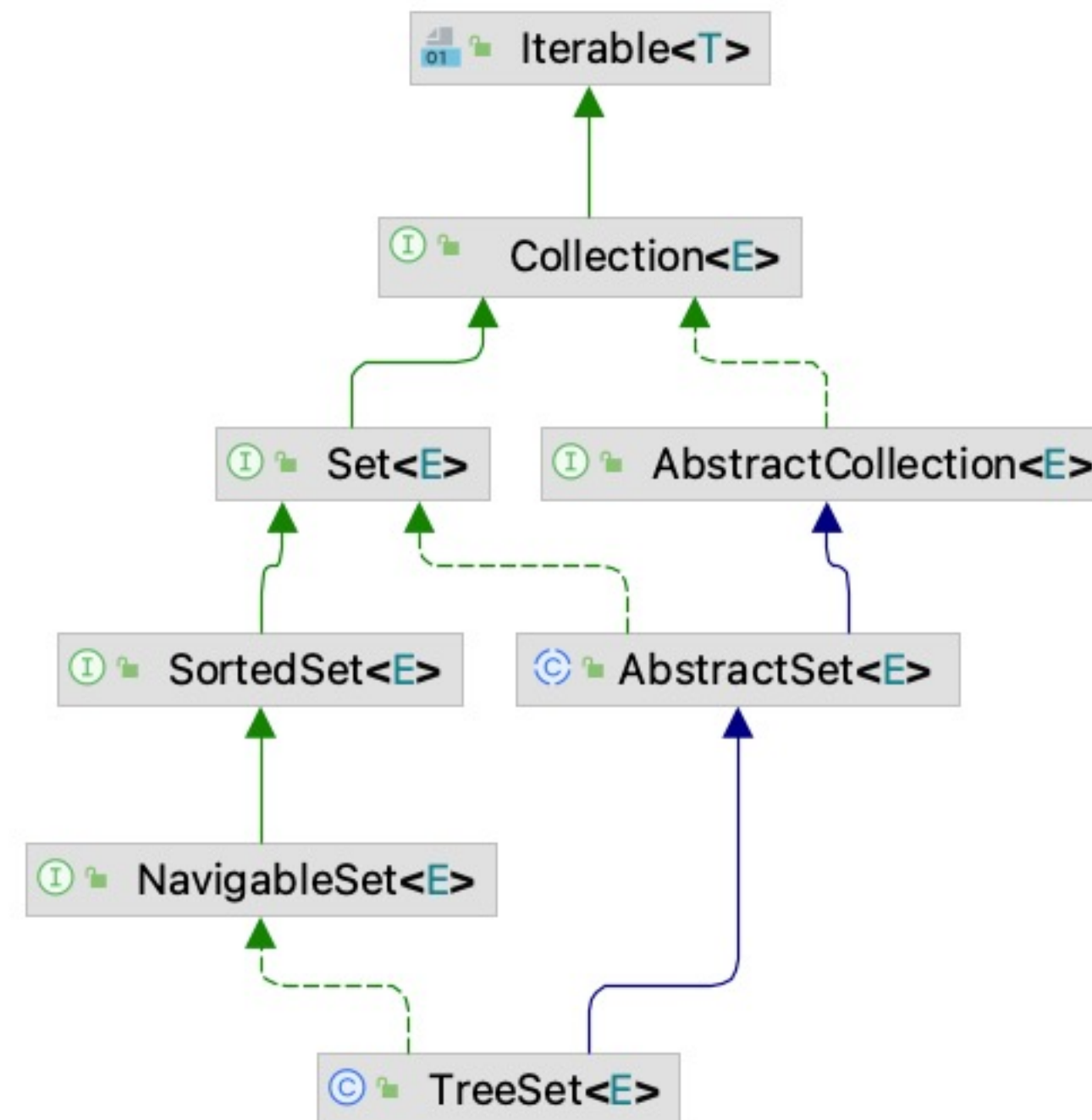
Упорядоченные структуры данных

SortedSet, NavigableSet, TreeSet

Документация и полезные ссылки

- [Документация](#)
- [A Guide to TreeSet in Java](#)

Иерархия классов

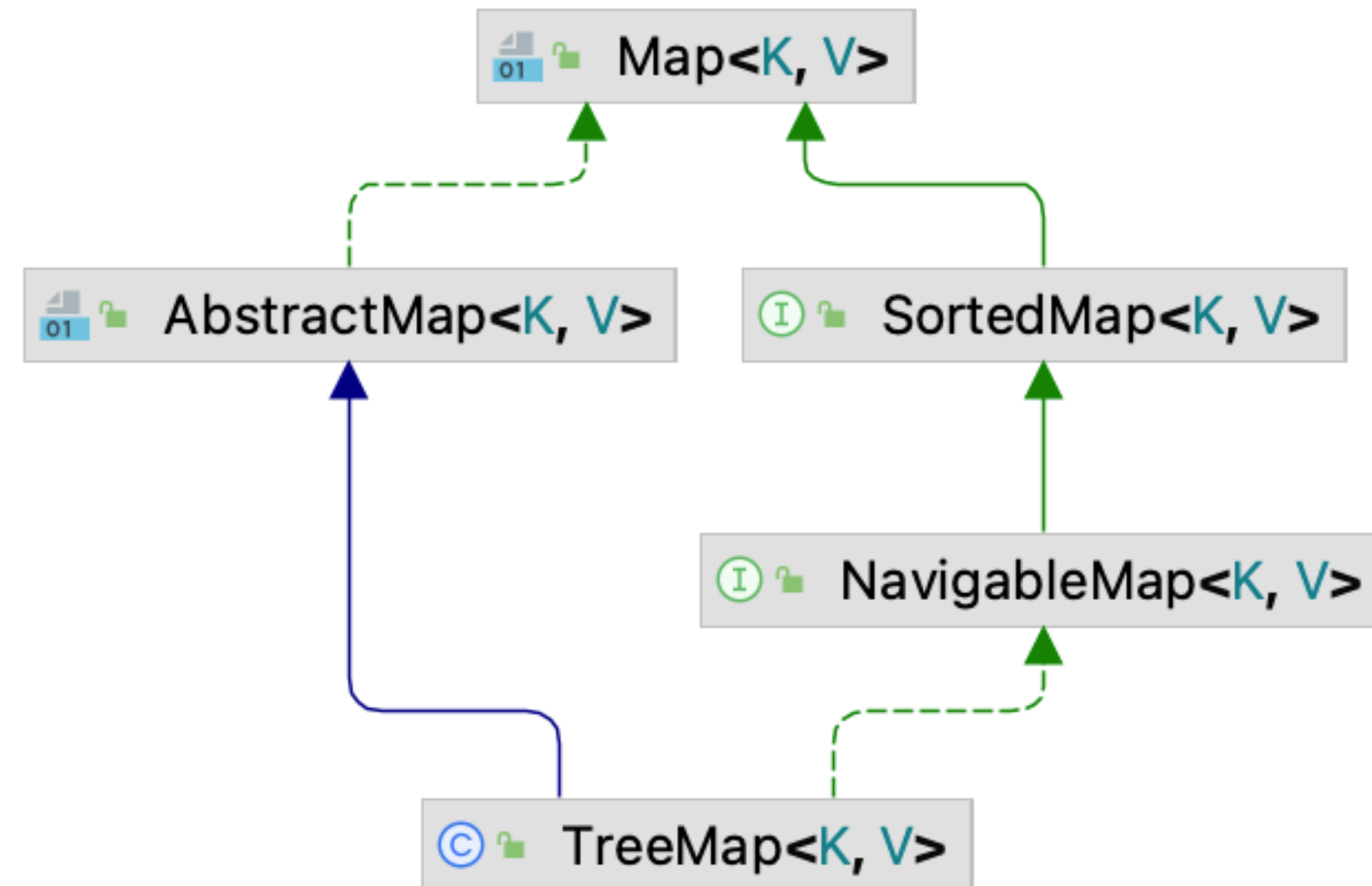


SortedMap, NavigableMap, TreeMap

Документация и полезные ссылки

- [Документация](#)
- [Introduction to Red-Black Trees](#)
- [ИТМО Алго Вики](#)

i Иерархия классов

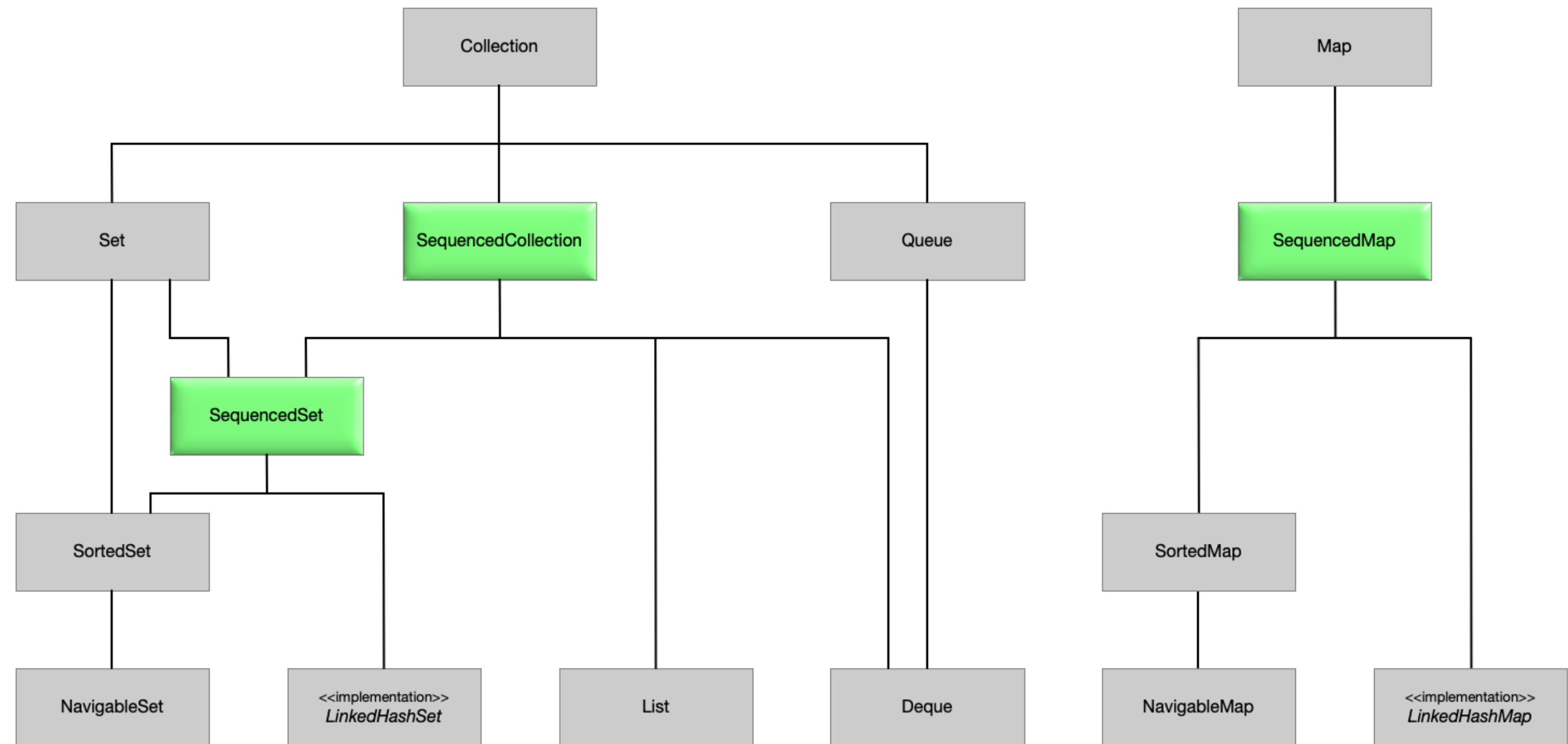


Sequenced Collections

Документация и полезные ссылки

- [JEP 431](#)
- [Sequenced Collections in Java 21](#)

Иерархия классов



Sequenced Collections JEP – Stuart Marks

2022-02-16

Утилитарные классы для работы с коллекциями

Классы Arrays и Collections, Guava и Apache Commons



Полезные ссылки

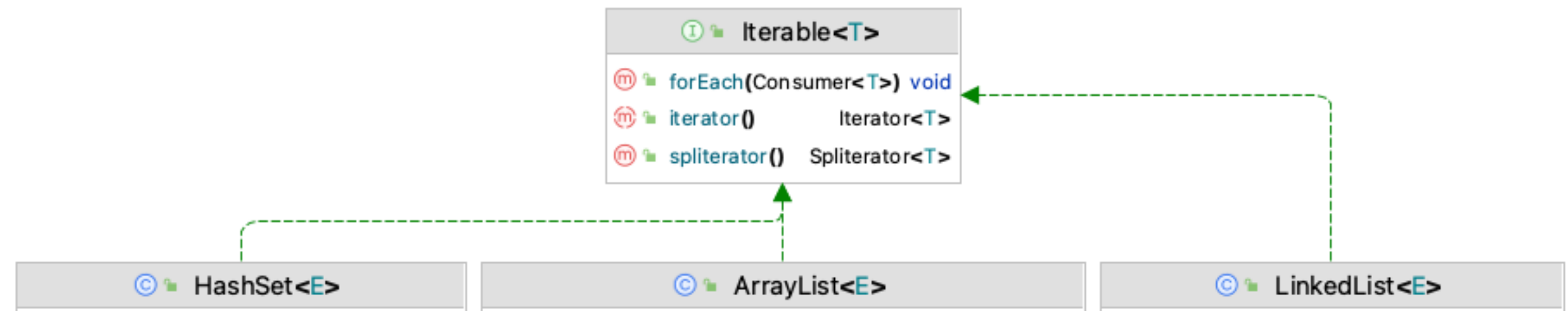
- [Arrays Documentation](#)
- [Collections Documentation](#)
- [Guide to the java.util.Arrays Class](#)
- [Guava Wiki](#)
- [Guava Guide](#)
- [Apache Commons Collections Wiki](#)
- [Apache Commons Collections Guide](#)

ТИНЬКОФФ

Итераторы

Iterable

Иерархия классов

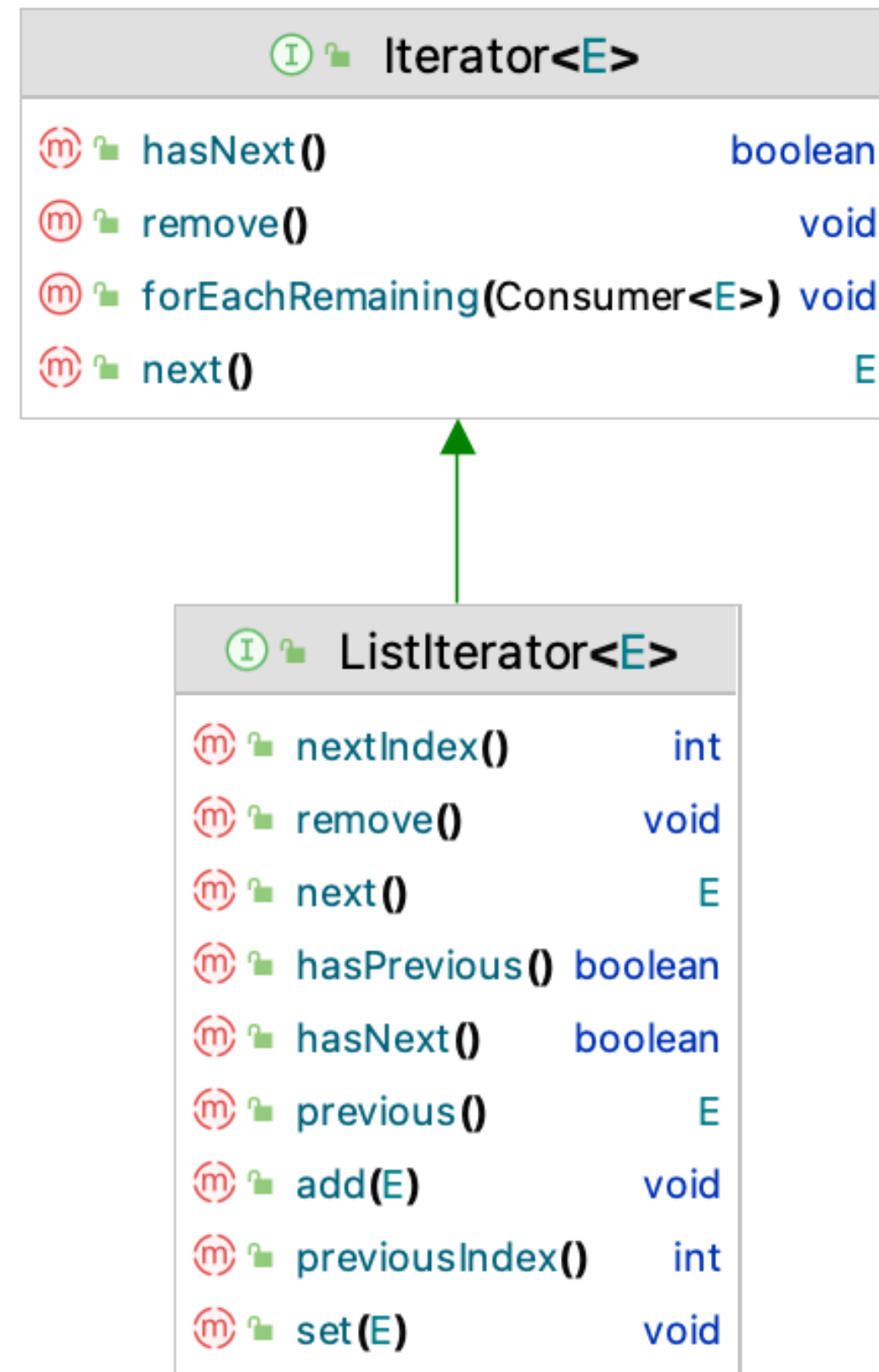


Iterator и ListIterator

Документация и полезные ссылки

- [Документация](#)
- [A Guide to Iterator in Java](#)
- [What's the Difference Between Iterator and ListIterator?](#)

Иерархия интерфейсов



fail-fast и fail-safe поведение

Определение

fail-fast поведение означает, что при возникновении ошибки или состояния, которое может привести к ошибке, система немедленно прекращает дальнейшую работу и уведомляет пользователя об этом. Использование fail-fast подхода позволяет избежать недетерминированного поведения программы в течение времени.

В противоположность fail-fast, итераторы fail-safe не вызывают никаких исключений при изменении данных, потому что они работают с клоном коллекции вместо оригинала.

ConcurrentModificationException

В Java Collections API некоторые итераторы ведут себя как fail-fast и выбрасывают ConcurrentModificationException, если после его создания была произведена модификация коллекции, т.е. добавлен или удален элемент напрямую из коллекции, а не используя методы итератора.

Реализация такого поведения осуществляется за счет подсчета количества модификаций коллекции (modification count):

- при изменении коллекции счетчик модификаций так же изменяется;
- при создании итератора ему передается текущее значение счетчика;
- при каждом обращении к итератору сохраненное значение счетчика сравнивается с текущим, и, если они не совпадают, возникает исключение.

Лямбда-выражения

Функции высшего порядка



Определение

Язык программирования имеет функции первого класса, если функции в нем рассматриваются как объекты первого класса. То есть, их можно передавать в качестве аргументов в другие функции, возвращать как результат других функций, присваивать их переменным и сохранять в структурах данных.

Функции высшего порядка – это функции, которые принимают и/или возвращают другие функции.

Функциональные интерфейсы

Определение

Функциональный интерфейс - это любой интерфейс, в котором определен ровно один абстрактный метод. При этом, кол-во методов с реализацией по умолчанию не ограничено.

В Java такие интерфейсы принято помечать аннотацией `@FunctionalInterface`.

Примеры функциональных интерфейсов

```
44 @FunctionalInterface
45 public interface BiConsumer<T, U> {
46
47     Performs this operation on the given arguments.
48     Params: t – the first input argument
49             U – the second input argument
50
51     void accept(T t, U u);
52
53     Returns a composed BiConsumer that performs, in sequence, this operation followed by the after
54     operation. If performing either operation throws an exception, it is relayed to the caller of the
55     composed operation. If performing this operation throws an exception, the after operation will not
56     be performed.
57     Params: after – the operation to perform after this operation
58     Returns: a composed BiConsumer that performs in sequence this operation followed by the after
59     operation
60     Throws: NullPointerException – if after is null
61
62     default BiConsumer<T, U> andThen(BiConsumer<? super T, ? super U> after) {
63         Objects.requireNonNull(after);
64
65         return (l, r) -> {
66             accept(l, r);
67             after.accept(l, r);
68         };
69     }
70 }
71
72 }
```

Лямбда-выражения



Определение

Лямбда-выражения были добавлены в Java 8 в качестве реализации функций первого класса. В первую очередь это было сделано для упрощения работы со Stream API.



Полезные ссылки

- [Lambda Expressions \(Гайд от Oracle\)](#)
- [Поверхностный обзор лямбд в статье на хабре](#)
- [Lambda Expressions and Functional Interfaces: Tips and Best Practices](#)
- [Java 8 Lambdas vs Anonymous Classes Performance](#)

ТИНЬКОФФ

Stream API

Stream API и его “аналоги”



Полезные ссылки

- [Java Stream API Tutorial](#)
- [Статья на хабре про Stream'ы №1](#)
- [When to use Parallel Streams](#)
- [Тагир Валеев про Stream'ы – раз, два и три](#)
- [Introduction to Spliterator](#)
- [Introduction to StreamEx](#)
- [Spliterator vs Iterator](#)
- [Project Reactor](#)
- [Kotlin Sequences](#)

ТИНЬКОФФ

Вопросы?



ТИНЬКОФФ



Обратная связь

ТИНЬКОФФ

Спасибо!

