Java Pro 2.3

TEL-RAN
by Starta Institute

Дерево. Компаратор. Итератор

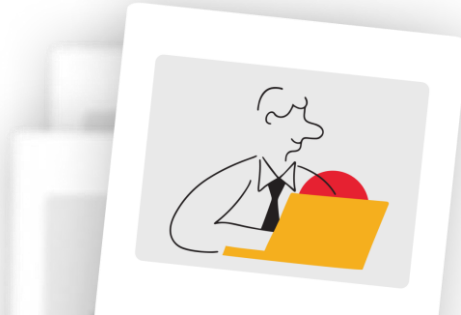
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Основной блок
2. Вопросы по основному блоку
3. Домашняя работа

TEL-RAN
by Starta Institute

TEL-RAN
by Starta Institute

1 ПОВТОРЕНИЕ ИЗУЧЕННОГО

Повторение

TEL-RAN
by Starta Institute

1 В чём особенность множества?

2 Set – это что?

3 Что вы знаете о реализации класса HashSet?

Повторение

TEL-RAN
by Starta Institute

1 В чём особенность множества?

Множество – неупорядоченная коллекция, хранящая только уникальные элементы.

2 Set – это что?

Интерфейс-наследник Collection, обладающий теми же методами и List.of. Может хранить null в некоторых своих реализациях.

3 Что вы знаете о реализации класса HashSet?

Реализация выполнена на основании хеш-таблицы. Хеш-таблица позволяет находить элементы быстрее благодаря тому, что ищет сперва по хешу, а только потом по эквивалентности. Чтобы эффективно хранить в такой таблице объекты собственного класса, в классе нужно переопределить методы *equals()* и *hashCode()* класса.

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. *4*

B. *5*

C. *Ошибка компиляции*

D. *Ошибка выполнения*

```
public static void main(String[] args) {
    Set<String> streets = Set.of(
            "Abbey Road",
            "Champs-Élysées",
            "Hollywood Boulevard",
            "La Rambla“
    );

    streets.add("Abbey Road");
    System.out.println(streets.size());
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. *4*

B. *5*

C. *Ошибка компиляции*

D. *Ошибка выполнения*

```
public static void main(String[] args) {
    Set<String> streets = Set.of(
            "Abbey Road",
            "Champs-Élysées",
            "Hollywood Boulevard",
            "La Rambla“
    );

    streets.add("Abbey Road");
    System.out.println(streets.size());
}
```

```
Exception in thread "main" java.lang.UnsupportedOperationException Create breakpoint
    at java.base/java.util.ImmutableCollections.uoe(ImmutableCollections.java:142)
    at java.base/java.util.ImmutableCollections$AbstractImmutableCollection.add(ImmutableCollections.java:147)
    at Main.main(Main.java:22)
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. *4*

B. *5*

C. *Ошибка компиляции*

D. *Ошибка выполнения*

```
public static void main(String[] args) {
    Set<String> streets = new HashSet<>(
            Set.of(
                    "Abbey Road",
                    "Champs-Élysées",
                    "Hollywood Boulevard",
                    "La Rambla"
            )
    );

    streets.add("Abbey Road");
    System.out.println(streets.size());
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. *4*

B. *5*

C. *Ошибка компиляции*

D. *Ошибка выполнения*

```
public static void main(String[] args) {
    Set<String> streets = new HashSet<>(
            Set.of(
                    "Abbey Road",
                    "Champs-Élysées",
                    "Hollywood Boulevard",
                    "La Rambla"
            )
    );

    streets.add("Abbey Road");
    System.out.println(streets.size());
}
```

Добавление элемента, который уже есть в множестве, приведёт к тому, что элемент не будет добавлен. Размер коллекции останется прежним

Повторение

Что будет выведено в консоль?

A. *4*

B. *3*

C. *Ошибка компиляции*

D. *Ошибка выполнения*

```
public static class NameInfo {
    private final String first;
    private final String last;

    public NameInfo(String first, String last) {
        this.first = first;
        this.last = last;
    }
}

public static void main(String[] args) {
    Set<NameInfo> streets = new HashSet<>(
            Set.of(
                    new NameInfo("Jim", "Kerry"),
                    new NameInfo("Albert", "Einstein"),
                    new NameInfo("Nelson", "Mandela")
            )
    );

    streets.add(new NameInfo("Jim", "Kerry"));
    System.out.println(streets.size());
}
```

Повторение

Что будет выведено в консоль?

A. *4*

B. *3*

C. *Ошибка компиляции*

D. *Ошибка выполнения*

Т.к. в классе NameInfo не переопределны методы hashCode() и equals(), объекты с одинаково заполненными полями всё равно не будут считаться эквивалентными (разные ссылки в памяти)

```
public static class NameInfo {
    private final String first;
    private final String last;

    public NameInfo(String first, String last) {
        this.first = first;
        this.last = last;
    }
}

public static void main(String[] args) {
    Set<NameInfo> streets = new HashSet<>(
            Set.of(
                    new NameInfo("Jim", "Kerry"),
                    new NameInfo("Albert", "Einstein"),
                    new NameInfo("Nelson", "Mandela")
            )
    );

    streets.add(new NameInfo("Jim", "Kerry"));
    System.out.println(streets.size());
}
```

Повторение

TEL-RAN
by Starta Institute

В чём прикол мема?

TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

- Я посадил дерево
- Всё познаётся в сравнении
- Верным путём

Проблема

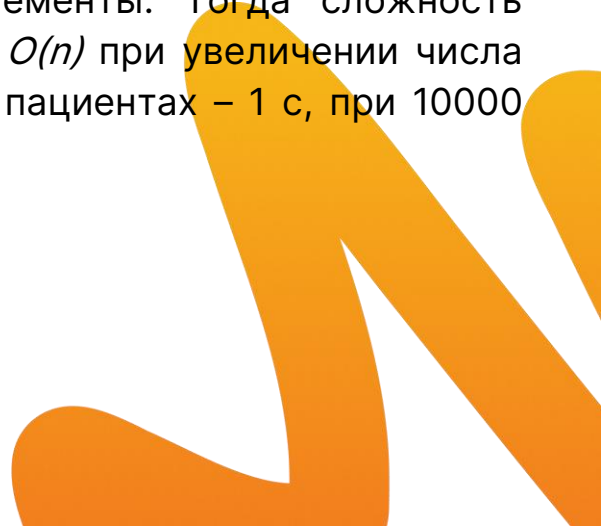
TEL-RAN
by Starta Institute

В прошлый раз мы создали перечень всех пациентов поликлиники.

Предположим, что два эквивалентных пациента будут сравниваться по имени, фамилии, дате рождения, информации о страховании и всем другим полям.

Тогда если хранить пациентов в *HashSet*, то при поиске по полям *имя* и *фамилия* (наиболее частый вариант поиска) придётся каждый элемент множества проверять на совпадение этих полей, т.е. нужно будет обойти все элементы. Тогда сложность алгоритма (скорость выполнения) будет возрастать линейно *O(n)* при увеличении числа пациентов. Например, при 100 пациентах – 100 мс, при 1000 пациентах – 1 с, при 10000 пациентах – 10 с.

Есть ли вариант искать быстрее?

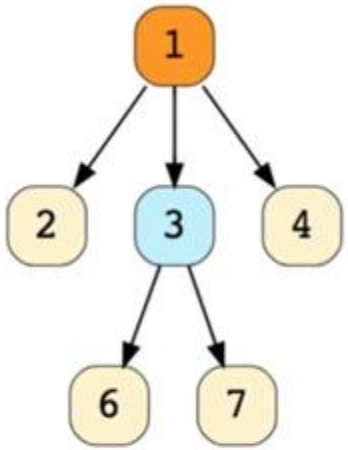


Я посадил дерево

Дерево

TEL-RAN
by Starta Institute

Дерево – это иерархическая структура данных, состоящая из вершин (узлов) и ребер, соединяющих их. Деревья широко используются в искусственном интеллекте и сложных алгоритмах для обеспечения эффективного механизма хранения данных.



Я посадил дерево

Дерево

TEL-RAN
by Starta Institute

Двоичное дерево – иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками.

Двоичное дерево поиска (англ. binary search tree, BST) – двоичное дерево, для которого выполняются следующие дополнительные свойства:

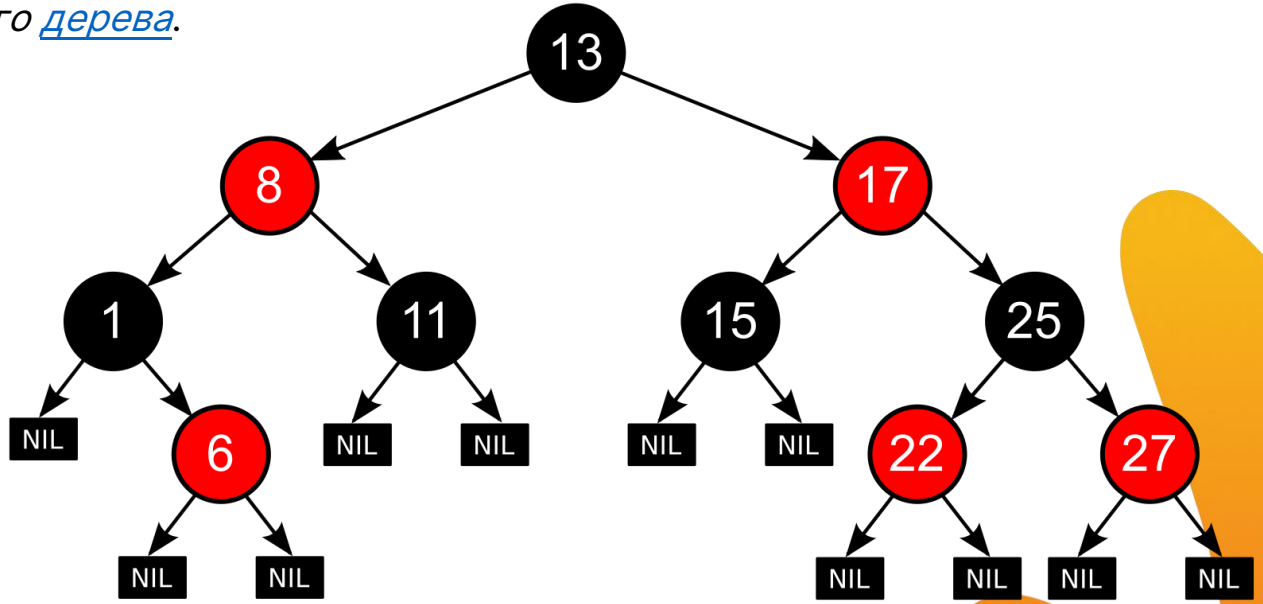
- оба поддерева — левое и правое — являются двоичными деревьями поиска;
- у всех узлов левого поддерева узла X значения ключей данных меньше либо равны, значению ключа узла X;
- у всех узлов правого поддерева произвольного узла X значения ключей данных больше, нежели значение ключа данных самого узла X.

Очевидно, данные в каждом узле должны обладать *ключами*, для которых определена операция *сравнения*.

Я посадил дерево
TreeSet

TEL-RAN
by Starta Institute

О классе **TreeSet** вспоминают в тех случаях, когда множество должно быть упорядочено. Каким образом упорядочивать, определяет разработчик при создании нового *TreeSet*. По умолчанию элементы располагаются в *естественном* порядке. Организованы они в виде *красно-чёрного дерева*.



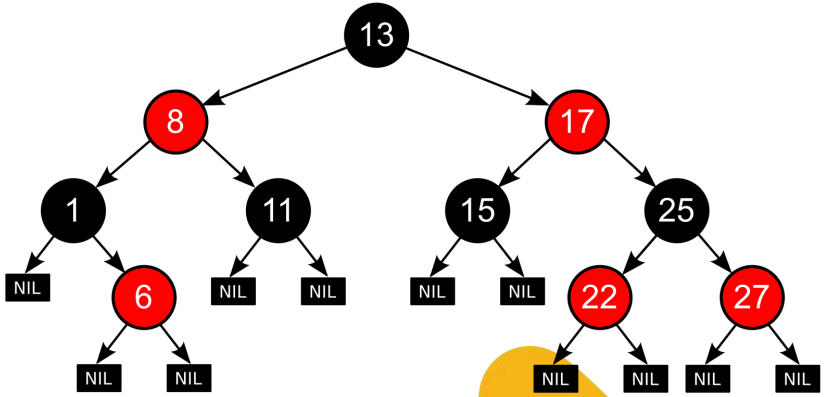

```
Set<Long> trSet = new TreeSet<>();
```

Визуализация работы дерева.

Проблема

TEL-RAN
by Starta Institute

Что такое *естественный* порядок чисел понятно (от меньшего к большему). Понятно, как сравнивать числа друг с другом (больше, меньше, равны), чтобы положить объект в дерево на правильное место.



Но как сравнивать нечисловые сущности – у них ведь есть только метод equals()? Например, пациентов: по росту, по весу, по тяжести заболевания, в алфавитном порядке фамилий и имён, по году? Каков естественный порядок пациентов?

Всё познаётся в сравнении
Comparable

TEL-RAN
by Starta Institute

java.lang.Comparable – это интерфейс, которому следует класс, чтобы его экземпляры можно было сравнивать между собой. Такое сравнение будет выбираться по умолчанию и называется *natural ordering*. Например, такой метод сравнения реализован в классах-обёртках типа Long или классе String (сортировка по алфавиту). В интерфейсе всего один метод, который нужно реализовать в соответствии с *контрактом*.

```
public interface Comparable<T> {
    /**
     * @param   o the object to be compared.
     * @return  a negative integer, zero, or a positive integer as this object
     *          is less than, equal to, or greater than the specified object.
     *
     * @throws NullPointerException if the specified object is null
     * @throws ClassCastException if the specified object's type prevents it
     *         from being compared to this object.
     */
    public int compareTo(T o);
}
```

Всё познаётся в сравнении
Comparable

TEL-RAN
by Starta Institute

Мы не можем сравнивать между собой экземпляры не связанных наследованием классов, а также сравнивать объекты с *null*.

Всё познаётся в сравнении

Контракт сравнения

TEL-RAN
by Starta Institute

Comparable.compareTo() и *Comparator.compare()* возвращают *int* по следующей схеме:

1 отрицательный *int* (первый объект отрицательный, то есть меньше)

2 положительный *int* (первый объект положительный, то есть больший)

3 ноль, если объекты равны

Всё познаётся в сравнении

Пример Comparable

TEL-RAN
by Starta Institute

```
package comparator;
public class Player implements Comparable<Player>{
    private String nickname;
    private int rank;
    private int age;

    public Player(String nickname, int rank, int age) {
        this.nickname = nickname;
        this.rank = rank;
        this.age = age;
    }

    // standard getters and setters

    @Override
    public int compareTo(Player o) {
        return Integer.compare(this.rank, o.rank);
    }
}
```

Интерфейс нужно параметризовать тем же классом

Единственный метод, который нужно переопределить

По умолчанию игроки будут сравниваться по рейтингу. Но если нам нужно отсортировать их по имени или возрасту?

Всё познаётся в сравнении

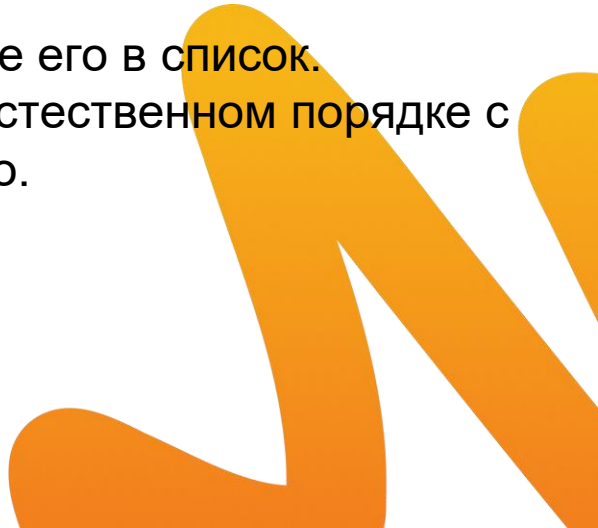
Задание

TEL-RAN
by Starta Institute

1 Измените класс Пациент, чтобы естественный порядок пациентов был по алфавиту.

2 Измените класс FileCabinet, чтобы он хранил множество пациентов в алфавитном порядке для быстрого поиска по фамилии.

3 Получите множество пациентов в main и превратите его в список. Выведите список в консоль. Отсортируйте список в естественном порядке с помощью Collections.sort(). Выведите список повторно.



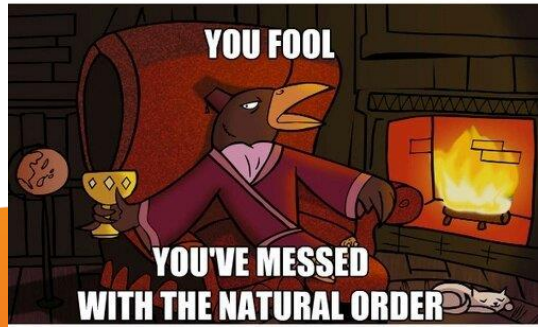
Проблема

TEL-RAN
by Starta Institute

Но если естественный порядок объектов не всегда нас устраивает?

Например, нам нужно формировать группы для ежегодной диспансеризации пациентов по возрасту. В таком случае логично хранить пациентов не по алфавиту, а по году рождения.

Хотелось бы иметь возможность задавать порядок хранения пациентов в TreeSet, а точнее, сравнивать пациентов уже не по имени, а по дате рождения.



Всё познаётся в сравнении
Comparator

TEL-RAN
by Starta Institute

Comparator – это интерфейс, главный метод которого *compare(arg1, arg2)* позволяет сравнивать объекты заданным способом. Он похож на метод *Comparable.compareTo()*. Главное отличие в том, что класс может реализовать только один метод *compareTo()*, а компараторов может быть сколь угодно много.

```java
@FunctionalInterface // Аннотация информирует, что интерфейс можно имплементировать в лямбда-выражениях
public interface Comparator<T> {
    int compare(T o1, T o2); // Основной метод, используемый для сравнения. Его нужно переопределить в своём компараторе
    boolean equals(Object obj); // Сравнение компараторов

    // Многие методы компаратора имеют реализацию по умолчанию
    default Comparator<T> reversed() { return Collections.reverseOrder(this); } // методы для создания обратного порядка сравнения
    public static <T extends Comparable<? super T>> Comparator<T> reverseOrder() { return Collections.reverseOrder(); }

    default Comparator<T> thenComparing(Comparator<? super T> other) { … } // методы с thenComparing нужны для сравнения объектов по нескольким полям
    default Comparator<T> thenComparingInt(…)…
    default Comparator<T> thenComparingLong(…)…
    default Comparator<T> thenComparingDouble(….)…

    public static <T extends Comparable<? super T>> Comparator<T> naturalOrder() { … } // метод для создания компаратора с естественным порядком сортировки

    public static <T> Comparator<T> nullsFirst(Comparator<? super T> comparator) { return new Comparators.NullComparator<>(true, comparator); } // метод возвращает null-safe компаратор, который помещает все null  в начало коллекции
    public static <T> Comparator<T> nullsLast(Comparator<? super T> comparator) { return new Comparators.NullComparator<>(false, comparator); } // аналогично – в конец коллекци

    public static <T, U extends Comparable<? super U>> Comparator<T> comparing(Function<? super T, ? extends U> keyExtractor) { … } // метод, который получает метод сравнения сущностей в качестве параметра

    public static <T> Comparator<T> comparingInt(…)…
    public static <T> Comparator<T> comparingLong(…)…
    public static<T> Comparator<T> comparingDouble(…)…
}
```

Всё познаётся в сравнении

Пример своего Comparator

TEL-RAN
by Starta Institute

ComparatorExample.zip

Всё познаётся в сравнении

Задание

TEL-RAN
by Starta Institute

1 Случайным образом заполните даты рождения пациентов.

2 Создайте второй конструктор для класса FileCabinet, который будет принимать пациентов и компаратор, задающий способ хранения пациентов во внутреннем множестве.

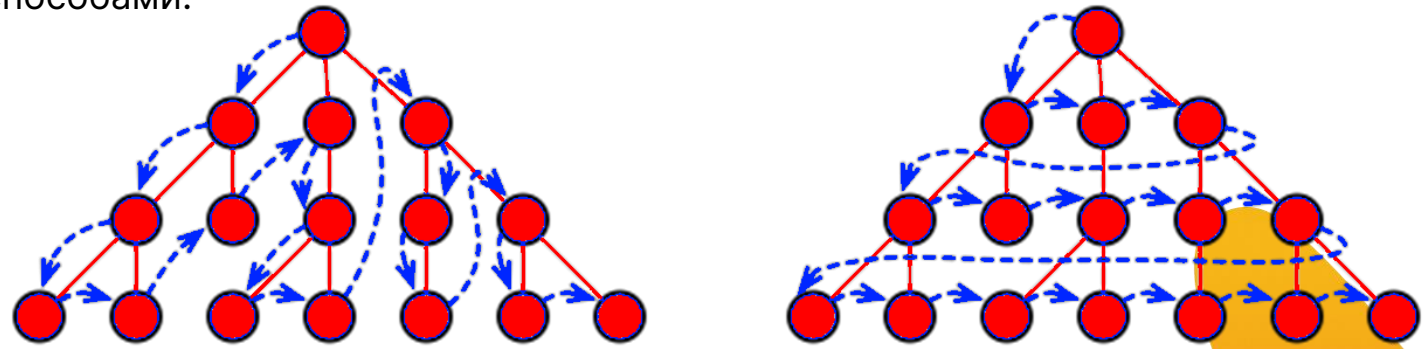
3 Создайте картотеки, которые хранят пациентов по

- фамилии – для быстрого поиска в регистратуре
- дате рождения – для формирования группы пациентов, которых нужно пригласить на плановый осмотр (диспансеризацию) один раз в 3 года.

Проблема

TEL-RAN
by Starta Institute

Одну и ту же коллекцию (особенно нелинейную, как *TreeSet*) можно поэлементно обойти разными способами.



Мы можем добавить разные способы обхода в коллекцию. Но добавляя всё новые алгоритмы в код коллекции, мы понемногу размываем её основную задачу, которая заключается в эффективном хранении данных.

Есть ли способ отдать задачу по обходу коллекции другому, специальному, классу?

Верным путём

Шаблон Итератор

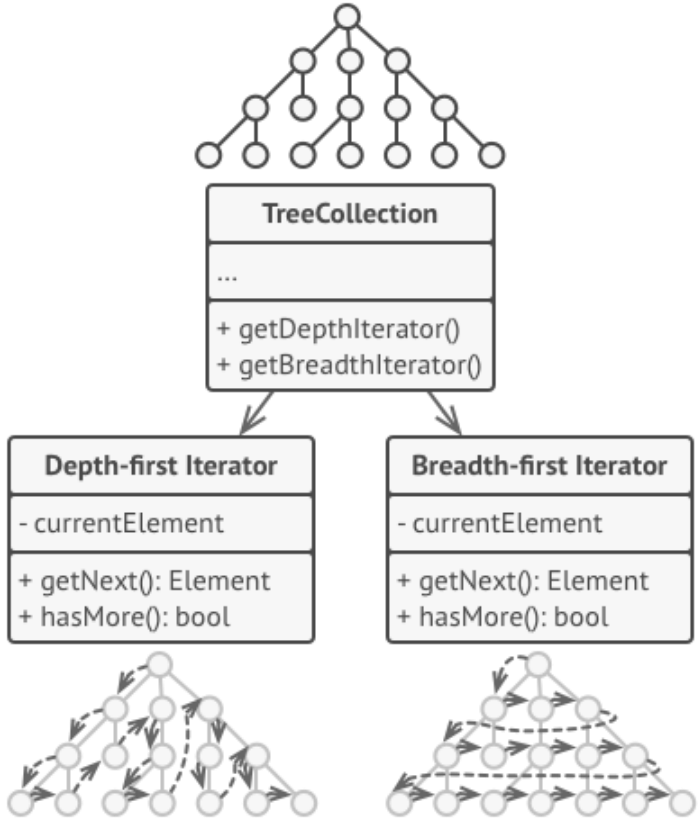
TEL-RAN
by Starta Institute

Итератор – это поведенческий паттерн проектирования, который даёт возможность последовательно обходить элементы составных объектов (массивов, коллекций), не раскрывая их внутреннего представления.

Итераторы содержат код обхода коллекции. Одну коллекцию могут обходить сразу несколько итераторов.

Объект-итератор будет отслеживать состояние обхода, текущую позицию в коллекции и сколько элементов ещё осталось обойти.

Если вам понадобится добавить новый способ обхода, можно создать новую реализацию итератора, не меняя существующий код коллекции.



Верным путём

Интерфейс Iterator

TEL-RAN by Starta Institute

Iterator – это интерфейс, которому следуют классы, позволяющие обходить коллекции. В интерфейсе *Collection* есть метод *Iterator<E> iterator()*. Он возвращает итератор для данной коллекции.

```
public interface Iterator <E> {
        E next();
        boolean hasNext();
        void remove();
}
```

Реализация интерфейса предполагает, что

1 с помощью вызова метода *next()* можно получить следующий элемент;

2 с помощью метода *hasNext()* можно узнать, есть ли следующий элемент, и не достигнут ли конец коллекции.

Если элементы еще имеются, то *hasNext()* вернет значение *true*. *hasNext()* следует вызывать перед методом *next()*, так как при достижении конца коллекции метод *next()* выбрасывает исключение *NoSuchElementException*.

3 метод *remove()* удаляет текущий элемент, который был получен последним вызовом *next()*.

Верным путём

Пример использования Iterator

TEL-RAN
by Starta Institute

Получаем итератор, реализованный в классе ArrayList

Ещё один вариант организации цикла – цикл с участием итератора

Цикл можно записать в функциональном виде

```
public static void main(String[] args) {
    List<String> states = new ArrayList<>();
    states.add("Germany");
    states.add("France");
    states.add("Italy");
    states.add("Spain");

    Iterator<String> iter = states.iterator();
    while (iter.hasNext()) {
        System.out.println(iter.next());
    }
}
```

```
iter.forEachRemaining(System.out::println);
```

Верным путём

Задание

TEL-RAN
by Starta Institute

1 Создайте список пациентов.

2 Обойдите список пациентов с помощью итератора.

Верным путём

Интерфейс ListIterator

TEL-RAN
by Starta Institute

У имплементаций интерфейса *List* есть ещё один итератор – имплементация интерфейса **ListIterator**, расширяющего интерфейс *Iterator*. Данный интерфейс позволяет итерироваться вперёд и назад по коллекции.

```
public interface ListIterator<E> extends Iterator<E>
```

Верным путём

Интерфейс ListIterator

TEL-RAN
by Starta Institute

ListIterator содержит следующие дополнительные методы:

void add(E obj) - вставляет объект *obj* перед элементом, который должен быть возвращен следующим вызовом *next()*;

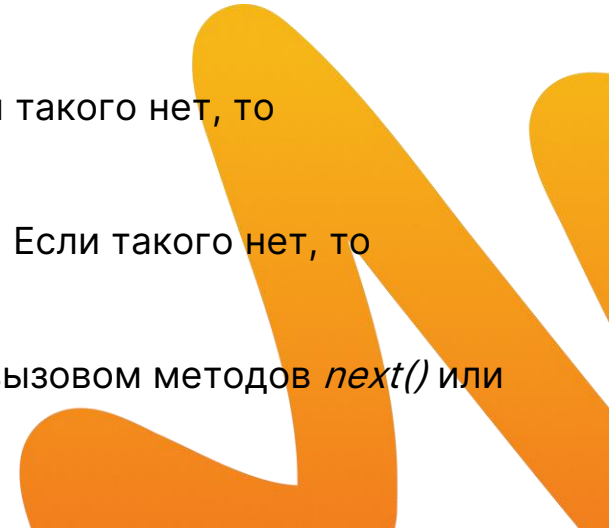
boolean hasPrevious() – возвращает true, если в коллекции имеется предыдущий элемент;

previous() – возвращает текущий элемент и переходит к предыдущему, если такого нет, то генерируется *NoSuchElementException*

int nextIndex() – возвращает индекс следующего элемента. Если такого нет, то возвращается размер списка

int previousIndex() – возвращает индекс предыдущего элемента. Если такого нет, то возвращается число -1

void set(E obj) – присваивает текущему элементу, выбранному вызовом методов *next()* или *previous()*, ссылку на объект *obj*



Верным путём

Как создать свой итератор?

TEL-RAN
by Starta Institute

1. Класс, по которому будет выполняться итерация, должен имплементировать интерфейс *Iterable* (предок интерфейса *Collection*).
2. Создать класс для имплементации интерфейса *Iterator* одним из способов:

- в отдельном классе (если нельзя менять исходный итерируемый класс);
- в виде внутреннего класса итерируемого класса;
- в виде анонимного класса внутри метода *public Iterator<T> iterator()*.

3. Использовать итератор в клиентском коде.

Имплементация интерфейса *Iterable* позволяет использовать для класса цикл *foreach*.

Верным путём

Пример своего итератора

TEL-RAN
by Starta Institute

MyIteratorExample.zip

Верным путём

Как создать свой итератор для массива?

TEL-RAN
by Starta Institute

Вариант 1. Преобразовать массив в *List* с помощью метода *asList()*.

```
String[] arr = { "A", "B", "C" };
Iterator<String> iterator = Arrays.asList(arr).iterator();
iterator.forEachRemaining(System.out::println);
```

Верным путём

Как создать свой итератор для массива?

TEL-RAN
by Starta Institute

Вариант 1. Преобразовать массив в *List* с помощью метода *asList()*.

```
String[] arr = { "A", "B", "C" };
Iterator<String> iterator = Arrays.asList(arr).iterator();
iterator.forEachRemaining(System.out::println);
```

Вариант 2. Реализовать свой итератор.

Применение анонимного класса

```
String[] arr = { "A", "B", "C" };
Iterator<String> iterator = new Iterator<String>() {
    private int i = 0;

    @Override
    public boolean hasNext() { return arr.length > i; }

    @Override
    public String next() { return arr[i++]; }
};
iterator.forEachRemaining(System.out::println);
```

Верным путём

Задание

TEL-RAN
by Starta Institute

1 Создайте итератор для массива. Итератор должен возвращать сначала чётные элементы, затем нечётные.

2 Создайте итератор для множества пациентов, чтобы получать пациентов в порядке уменьшения массы тела (профилактика недоедания у пациентов).

Верным путём

ConcurrentModificationException

TEL-RAN by Starta Institute

Нельзя модифицировать коллекцию напрямую, когда итерируешь по ней. Удалять элементы можно только через методы итератора.

Другой выход – переложить нужные элементы в другую коллекцию.

```
Set<Integer> set = Set.of();
// Возникнет ConcurrentModificationException
Iterator iterator = set.iterator();
while (iterator.hasNext()){
    int n = (int) iterator.next();
    if(n<10)
        set.remove(n);
}

// Возникнет ConcurrentModificationException
for(Integer n : set)
    if(n>10)
        set.remove(n);

// Правильно
Iterator<Integer> iterator = set.iterator();
while (iterator.hasNext()) {
    if (iterator.next() > 10) {
        iterator.remove();
    }
}
```

Верным путём

Исключения в коллекциях

TEL-RAN
by Starta Institute

Итераторы **fail-fast** в случае изменения структуры коллекции во время итерирования вызывают исключение.

Итераторы **fail-safe** не вызывают никаких исключений при изменении структуры, потому что они работают с клоном коллекции вместо оригинала. Такие итераторы используются в многопоточном программировании. Например, итератор коллекции *CopyOnWriteArrayList* и итератор представления *keySet* коллекции *ConcurrentHashMap* являются примерами итераторов fail-safe.

Верным путём

Что ещё за Enumeration?

TEL-RAN by Starta Institute

Паттерн *Итератор* в Java (1.0) был изначально реализован в виде интерфейса *Enumeration*

```
public interface Enumeration<E>
```

Enumeration в два раза быстрее *Iterator* и использует меньше памяти, но *Iterator* потокобезопасен, т.к. не позволяет другим потокам модифицировать коллекцию при переборе. *Enumeration* можно использовать только для *read-only* коллекций. Так же у него отсутствует метод *remove()*;

```
boolean hasMoreElements();

E nextElement();
```

В последствии (в Java 1.2) его заменили на интерфейс *Iterator,* а в сам *Enumeration* добавили метод для перехода к новому интерфейсу *Iterator.*

```
default Iterator<E> asIterator() {
    return new Iterator<>() {
        @Override public boolean hasNext() {
            return hasMoreElements();
        }
        @Override public E next() {
            return nextElement();
        }
    };
```

TEL-RAN
by Starta Institute

3 ВОПРОСЫ ПО ОСНОВНОМУ БЛОКУ

TEL-RAN
by Starta Institute

4 Домашнее задание

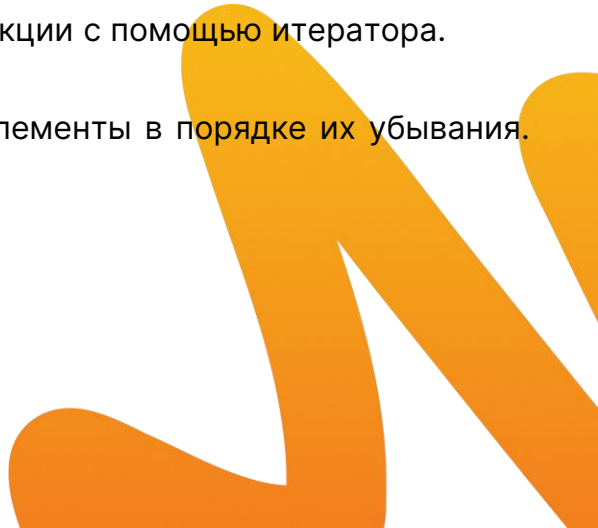
Домашнее задание

TEL-RAN
by Starta Institute

1 Создайте класс FullName полного имени человека (firstName, lastName). Создайте TreeSet с экземплярами этого класса. При создании TreeSet передайте в него компаратор, который сравнивает экземпляры по полю firstName. Выведите множество в консоль.

2 Создайте коллекцию (List или Set), добавьте несколько элементов. Организовать цикл while по коллекции с помощью итератора. Организовать цикл for (не путать с foreach!) по коллекции с помощью итератора.

3 Создайте итератор по массиву целых чисел, который будет выводить элементы в порядке их убывания. Исходный массив не должен при этом измениться.



Дополнительная практика

TEL-RAN
by Starta Institute

Создайте свой класс для хранения информации о входящем/исходящих вызове телефона (направление звонка, номер вызываемого/вызывающего абонента, был ли принят звонок, дата и время). Создайте класс, хранящий журнал вызовов. Создайте итератор для обхода набора звонков в журнале:

- сначала непринятые звонки в хронологическом порядке;
- все остальные звонки в хронологическом порядке.

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute