



**Java**

# В предыдущих сериях

- Static методы и переменные в Java – механизм, иногда использующийся, но в целом нарушающий ООП
- Передача по ссылке и по значению – в Java примитивы передаются по значению, а объекты – по значению ссылки
- Неизменяемые объекты – объекты, состояние которых невозможно изменить (все обертки над примитивами Java)
- Класс Object – предок всех классов в Java. Методы equals, toString и hashCode – часто переопределяются

# Глава 5.1

## Классические структуры данных

# Структуры данных

- Набор однотипных данных
- Основные структуры данных присутствуют в большинстве языков
- Обладают свойствами, делающими их удобными для определенных операций
- Каждая структура данных имеет свои плюсы и минусы

# Массивы

- Лежат в памяти целым «куском»
- Элементы проиндексированны
- Чтобы вставить элемент в середину, нужно «сдвинуть» элементы справа
- Вставка в конец очень быстрая
- Получение элемента по индексу очень быстрое
- Если не хватает размера, нужно создать новый массив, и скопировать его данные из старого

1	5	6	42	3	2	null	null
---	---	---	----	---	---	------	------

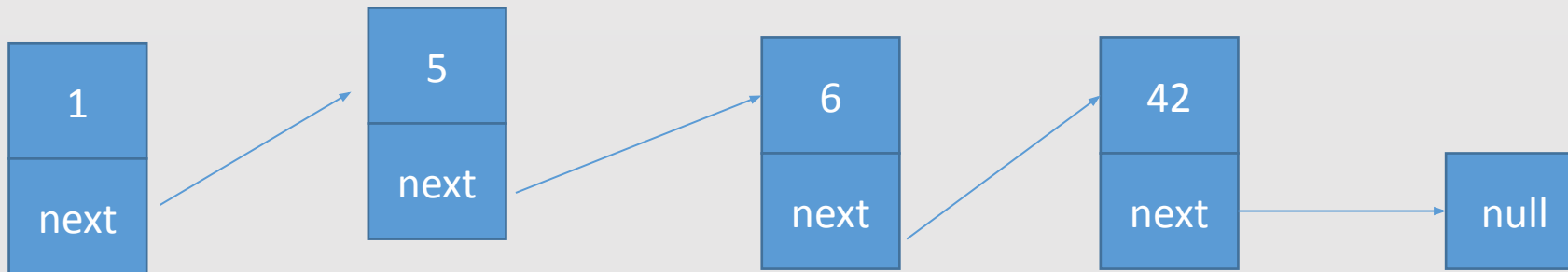
# Оценим сложности стандартных операций

- Получение по индексу
- Вставка вконец
- Вставка в середину и в начало
- Удаление последнего элемента
- Удаление элемента из середины

1	5	6	42	3	2	null	null
---	---	---	----	---	---	------	------

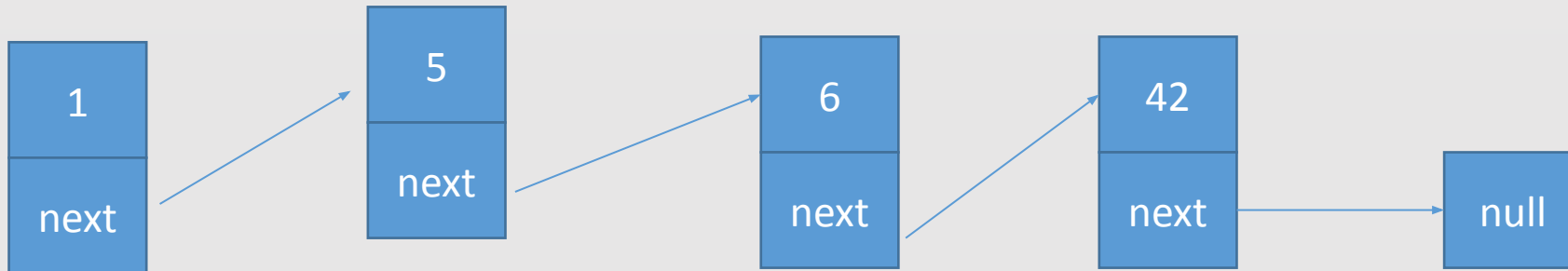
# СВЯЗНЫЕ СПИСКИ

- Состоят из узлов - Node
- Каждая нода имеет как минимум ссылку на следующий элемент(односвязный список)
- Как правило, эффективен только когда надо много вставлять в середину



# Оценим сложности стандартных операций

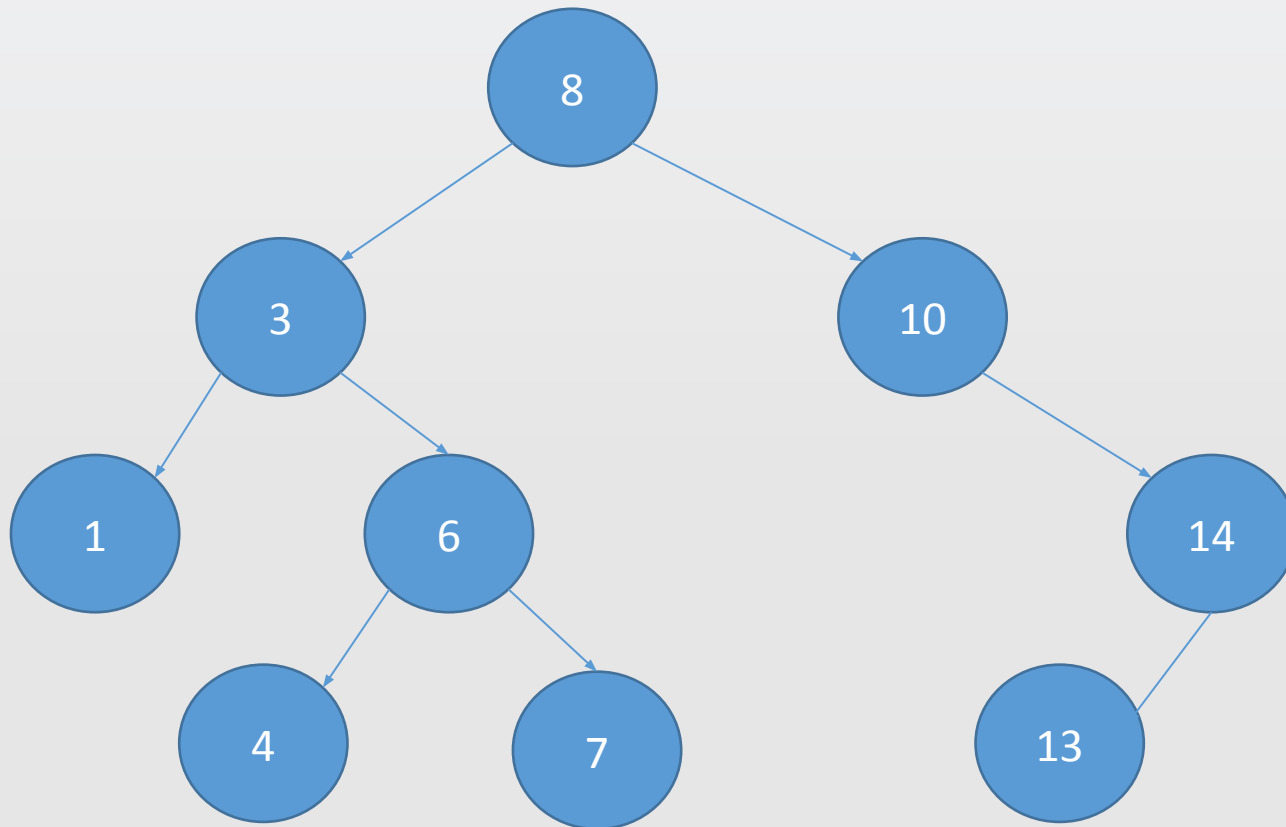
- Получение по индексу
- Вставка вконец
- Вставка в середину и в начало
- Удаление последнего элемента
- Удаление элемента из середины





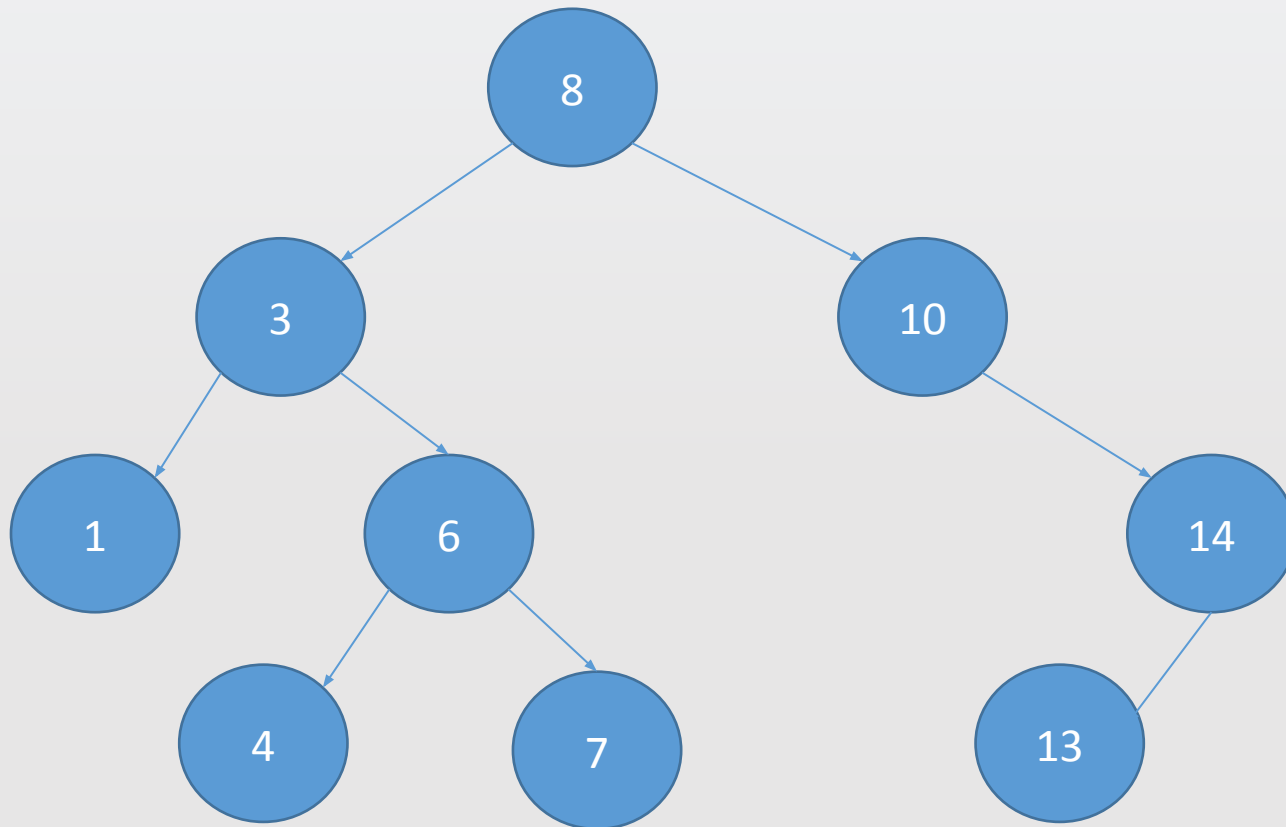
# Деревья

- Чаще всего используются для поиска
- Некоторые умеет автобалансироваться



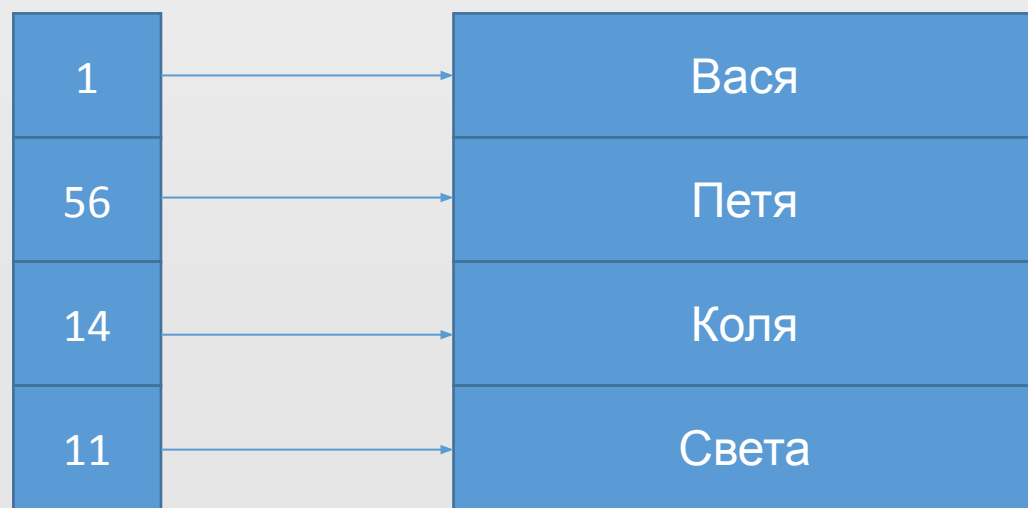
# Бинарное дерево

- Оценим сложность поиска



# Ассоциативный массив

- Ключ – значение
- Как правило используется для получение элемента по ключу
- Оценивать не будем (пока 😊)



# Глава 5.1.1

## Интерфейсы

### Comparable и Comparator

# Сравнение объектов на «больше» и «меньше» в Java

- В Java часто приходится сравнивать объекты не только на равенство, но и на «больше» и «меньше»
- Существует 2 способа сравнивать объекты в Java – реализовывать интерфейс Comparable или Comparator

# Интерфейс Comparable

```
public interface Comparable<T> {  
    @Contract(pure = true)  
    public int compareTo( @NotNull T o);  
}
```

compareTo возвращает:

- Ноль, если два объекта равны
- число  $>0$ , если первый объект (на котором вызывается метод) больше, чем второй (который передается в качестве параметра);
- число  $<0$ , если первый объект меньше второго

# Интерфейс Comparable

```
public abstract class Vehicle implements Movable, PassengerTransport, Comparable<Vehicle> {  
  
    private static final int MAX_PASSENGERS_COUNT = 3;  
    private static int VEHICLES_CREATED = 0;  
  
    public static int getVehiclesCreated() {  
        return VEHICLES_CREATED;  
    }  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    private final String serialNumber;  
  
    @Override  
    public int compareTo(Vehicle o) {  
        return serialNumber.compareTo(o.serialNumber);  
    }  
}
```

«Дженерик», в данном случае говорит, что мы сравниваем Vehicle

# Интерфейс Comparator

```
public abstract class Vehicle implements Movable, PassengerTransport, Comparable<Vehicle> {  
  
    private static final int MAX_PASSENGERS_COUNT = 3;  
    private static int VEHICLES_CREATED = 0;  
  
    public static int getVehiclesCreated() {  
        return VEHICLES_CREATED;  
    }  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    private final String serialNumber;  
  
    public String getSerialNumber() {  
        return serialNumber;  
    }  
}
```

Метод получения или  
«getter» для  
серийного номера

```
import java.util.Comparator;  
  
public class VehicleComparator implements Comparator<Vehicle> {  
  
    @Override  
    public int compare(Vehicle o1, Vehicle o2) {  
        return o1.getSerialNumber().compareTo(o2.getSerialNumber());  
    }  
}
```



# Вопросы и ответы



# Глава 5.2

## Коллекции в Java

# Коллекции

- Очень часто при разработке приходится хранить наборы одинаковых данных (мы уже встречали массивы)
- Обычные массивы не очень удобны, т.к. они не расширяются автоматически, да и не имеют какого-то удобного API
- Коллекции в Java используются как раз для хранения массивов данных.
- Есть 2 основные ветки в Java Collection Framework.

# Коллекции

- Основа первой ветки – интерфейс Iterable
- Iterable можно воспринимать как свойство “перечисляемый”, может отдать iterator с помощью метода iterator()
- Все что Iterable можно использовать в цикле foreach (начиная с 5 Java)
- В более старых версиях «пройтись» по каждому элементу структуры данных можно было с помощью Iterator
- Заметим, что в Iterator нет операции получения по

# Iterator

```
public interface Iterator<E>
```

An iterator over a collection. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

**Since:**

1.2

**See Also:**

Collection, ListIterator, Iterable

## Method Summary

All Methods

Instance Methods

Abstract Methods

Default Methods

Modifier and Type

Method and Description

default void

**forEachRemaining**(Consumer<? super E> action)

Performs the given action for each remaining element until all elements have been processed or the action throws an exception.

boolean

**hasNext**()

Returns true if the iteration has more elements.

E

**next**()

Returns the next element in the iteration.

default void

**remove**()

Removes from the underlying collection the last element returned by this iterator (optional operation).

# Iterator

```
public static void main(String[] args) {
```

```
    Collection<String> list = getCollection();  
    Iterator<String> iterator = list.iterator();  
    while(iterator.hasNext()){  
        System.out.println(iterator.next());  
    }  
}
```

Так жили в Java в  
доисторические  
времена

```
private static List<String> getCollection() {  
    List<String> list = new ArrayList<>();  
    list.add("one");  
    list.add("two");  
    list.add("three");  
    return list;  
}
```

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ...
```

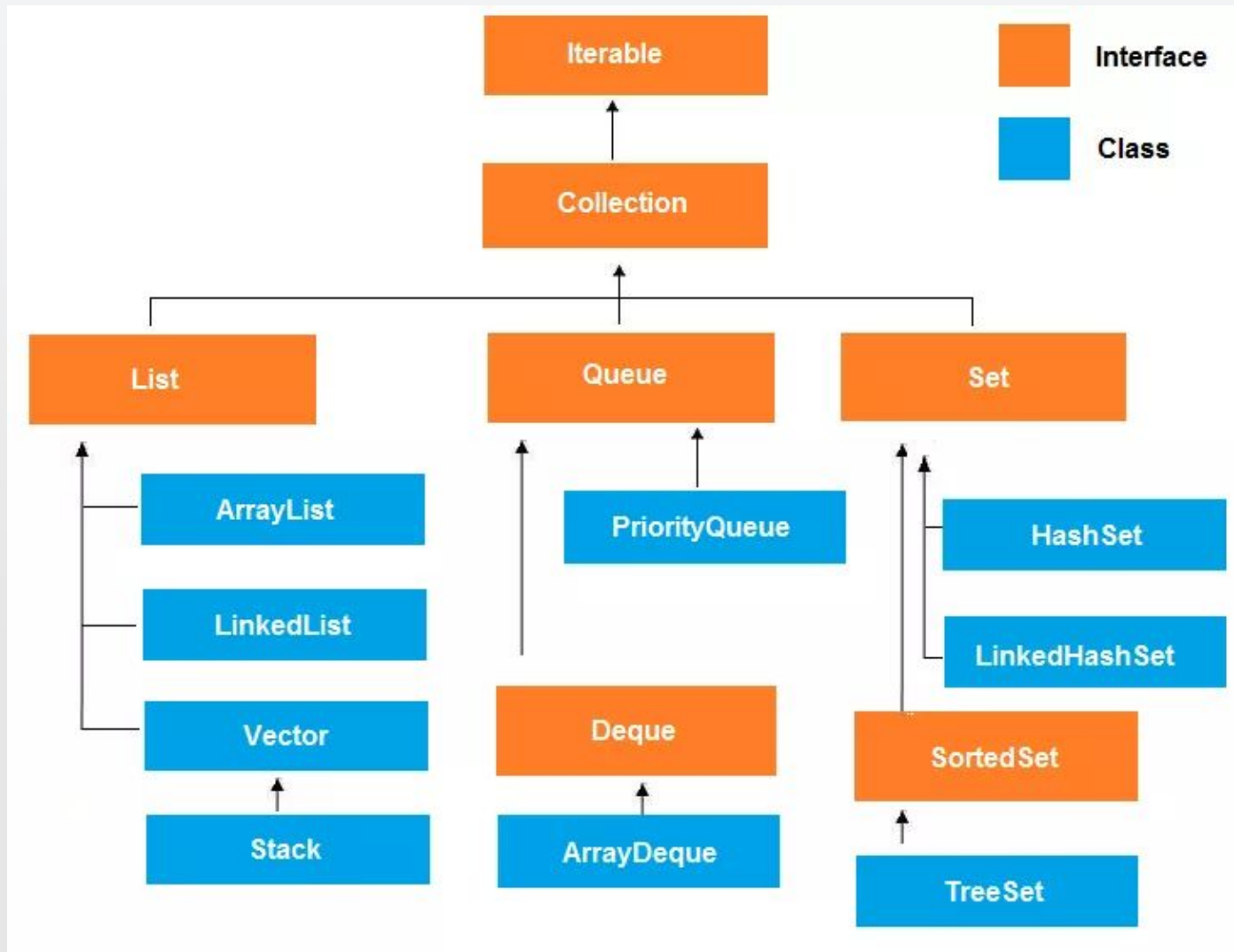
```
one  
two  
three
```

```
Process finished with exit code 0
```

# Коллекция - интерфейс

- Коллекция добавляет операции add, contains
- Так же в коллекциях появляется remove конкретного элемента
- Основные интерфейсы наследники коллекций – List, Set, Queue

# Коллекция - интерфейс





# Set

- Множество (то есть элементы уникальны)
- Хранит каждый элемент 1 раз (проверяется с помощью equals)
- Можно пройтись по всем элементам
- Можно проверить, содержит ли Set существующий элемент
- **Нельзя** получить элемент по индексу

# Set

```
public static void main(String[] args) {  
  
    Set<String> collection = new HashSet<>();  
    collection.add("one");  
    collection.add("two");  
    collection.add("three");  
    collection.add("two");  
    System.out.println(collection.size());  
}
```

---

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java"
```

```
3
```

```
Process finished with exit code 0
```

# Set: популярные реализации

- HashSet – самая популярная реализация. Использует хеш код для ускорения производительности
- LinkedHashSet – поддерживает порядок вставки
- TreeSet – наследует SortedSet, внутри красно-черное дерево. Туда можно положить только элементы, которые можно сравнивать (реализуют Comparable) или нужно передать специальный Comparator.

# List

- List – список. Основная фишка – получение элементов по индексу
- Две самые известные реализации – ArrayList и LinkedList
- Чаще всего используют ArrayList

# List

```
public static void main(String[] args) {  
  
    List<String> collection = new ArrayList<>();  
    collection.add("one");  
    collection.add("two");  
    collection.add("three");  
    collection.add("two");  
    System.out.println(collection.size());  
    System.out.println(collection.get(2));  
}
```

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ..
```

```
4
```

```
three
```

```
Process finished with exit code 0
```

# List: популярные реализации

- ArrayList – самая популярная реализация. Внутри – массив.
- Сложности операций – такие, как у массива
- Автоматически расширяется при достижении предела размера внутреннего массива
- Используется в 99% случаев

# List: популярные реализации

- LinkedList – связный список
- Сложности алгоритмов как у связного списка
- Имеет смысл использовать, только когда нужно много добавлять в середину
- Очень популярный вопрос на собеседовании – разница между ArrayList и LinkedList

# Queue (кueue) - очередь

- Очередь
- Сохраняет принцип – первый пришел первый ушел
- Популярная реализация - PriorityQueue



# Queue

```
public static void main(String[] args) {  
  
    Queue<String> collection = new PriorityQueue<>();  
    collection.add("one");  
    collection.add("two");  
    collection.add("three");  
  
    System.out.println(collection.poll());  
    System.out.println(collection.poll());  
    System.out.println(collection.poll());  
    System.out.println(collection.poll());  
  
}
```

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ...
```

```
one  
three  
two  
null
```

```
Process finished with exit code 0
```

# Фильтрация элементов коллекции: безопасные способы

- `removeIf`
- Создать новую коллекцию, и положить туда нужные элементы
- Фичи java 8 + (пока мы про них не знаем)
- Итератором
- Нельзя – в `forEach`! (`ConcurrentModificationException`)

# Вопросы и ответы



# Глава 5.2.1

## Utility - классы

# Utility класс

- Элемент «процедурного программирования»
- По сути – набор процедур
- Использовать надо с осторожностью
- Закрывает `final` наследования модификатором `final`
- Нельзя создать сущность (для этого делаем приватный конструктор)

# Utility класс

final class – закрыт он наследования

Приватный конструктор по умолчанию не даст создать инстанс

```
public final class CollectionBenchmarkUtils {  
  
    private CollectionBenchmarkUtils() {}  
  
    public static final int BENCHMARK_SIZE = 10000000;  
  
    public static void testInsertionInTheEnd(Collection<Integer> collection) {  
        System.out.printf("Testing insertion, benchmark size = %s, collection type %s%n", BENCHMARK_SIZE, collection.getClass().getSimpleName());  
        long startTime = System.currentTimeMillis();  
        for (int i = 0; i <= BENCHMARK_SIZE; i++) {  
            collection.add(i);  
        }  
        long endTime = System.currentTimeMillis();  
        System.out.printf("benchmark took %s milliseconds%n", endTime - startTime);  
    }  
}
```

# Вопросы и ответы



# Глава 5.3

## Практика. Бенчмарк реализаций интерфейса Collection



# Домашнее задание

- Написать перформанс тесты для следующих случаев
- Добавление элементов в ArrayList, LinkedList, HashSet, TreeSet
- Добавление по фиксированному индексу (2 теста, в начало и в конец) в ArrayList, LinkedList.
- Заполнить HashSet и TreeSet целыми числами. Проверить разницу в скорости по методу contains (вызывать контейнс в цикле, а не один раз)
- Сравнить скорость contains для HashSet, ArrayList, LinkedList
- Написать комментарий, почему отличается производительность в том или ином случае