ORACLE

JDK 8: Молот Лямбд

Что нового в библиотеках

Сергей Куксенко

sergey.kuksenko@oracle.com, @kuksenk0

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: про доклады

JDK 8: Я, лямбда:

- доклад про лямбды как самостоятельную языковую фичу
  - lambda expressions
  - method references

JDK 8: Молот лямбд: (← вы здесь)

- доклад про то, что лямбды ещё изменили в Java и JDK
  - more λ-accepting methods in JDK
  - default methods in interfaces
  - static methods in interfaces
  - streams (a.k.a. bulk collection operations)

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: вопросы

- Что?

Slide 5/45. Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: вопросы

- Что?
- Как?

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: вопросы

- Что?
- Как?
- Зачем?

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: вопросы

- Что?
- Как?
- Зачем?
- ~~Почему?~~

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: вопросы

- Что?
- Как?
- Зачем?
- ~~Почему?~~
- ~~Почему бы не?~~

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: λ samples code

https://github.com/shipilev/jdk8-lambda-samples

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Введение: Мотивация

У нас много поводов менять код в JDK:

- введение лямбд существенно упрощает многие API
- введение default method'ов упрощает эволюцию библиотек
- введение статических методов в интерфейсах избавляет от мелких утильных классов

Нам важно первыми понаступать на грабли, перед тем, как давать фичи девелоперам.

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: Everything is better with λ

Многие места в библиотеке выглядели бы лучше:

- `Iterable.forEach(Consumer<T>)`
- `Collection.removeIf(Predicate<E>)`
- `ThreadLocal.withInitial(Supplier<T>)`
- `Comparator.comparing(Function<T,U>)`
- `Map.computeIfAbsent(K, Function<K,V>)`
- `AtomicInteger.updateAndGet(IntUnaryOp)`

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
}

Comparator<M> m = new Comparator<M>() {
  @Override
  public int compareTo(M m1, M m2) {
    int v = m1.last.compareTo(m2.last);
    if (v != 0) {
      return v;
    } else {
      return m1.first.compareTo(m2.first);
    }
  }
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Java™ ORACLE®

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
}

Comparator<M> m = new Comparator<M>() {
  @Override
  public int compareTo(M m1, M m2) {
    int v = m1.last.compareTo(m2.last);
    return (v != 0) ?
      v : m1.first.compareTo(m2.first);
  }
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
}

Comparator<M> m = (m1, m2) -> {
  int v = m1.last.compareTo(m2.last);
  return (v != 0) ?
    v : m1.first.compareTo(m2.first);
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Java™ ORACLE®

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
}

Comparator<M> m = (m1, m2) -> {
  int v = C.comparing(x -> x.last)
                  .compare(m1, m2);
  return (v != 0) ?
     v : C.comparing(x -> x.first)
                  .compare(m1, m2);
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
}

Comparator<M> m =
      C.comparing(x -> x.last)
       .thenComparing(x -> x.first);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: Comparators

```
public class M {
  final String first;
  final String last;
  String getFirst() { return first; }
  String getLast()  { return last; }
}

Comparator<M> m =
    C.comparing(M::getLast)
     .thenComparing(M::getFirst);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Point λ-fications: call for action

- в JDK уже порядочное количество таких точечных методов
- сейчас самое время добить остатки!
- если есть что-то, что было бы глупо упустить?

Пробуйте билды, пишите на:
`lambda-dev@openjdk.java.net`

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: проблема

Backward compatibility:

- compatiblity – священная корова Java
- нельзя менять байткод, если это ломает приложения
- нельзя вводить новые идиомы, если это ломает ожидания
- нельзя даже поменять serialization form!
- нельзя менять публичный API, если это ломает приложения

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Java™ ORACLE®

Default methods: проблема

Нельзя так просто взять и добавить метод в интерфейс:

```
interface Collection<T> {
    /** @since 1.8 */
    void removeAll(Predicate<T> p);
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: проблема

Нельзя так просто взять и добавить метод в интерфейс:

```
interface Collection<T> {
    /** @since 1.8 */
    void removeAll(Predicate<T> p);
}

class GooAbbaCollection<T>
        implements Collection<T> {
    // compiling with JDK 8:
```

Java™ ORACLE

Slide 19/45. Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: проблема

Нельзя так просто взять и добавить метод в интерфейс:

```
interface Collection<T> {
    /** @since 1.8 */
    void removeAll(Predicate<T> p);
}

class GooAbbaCollection<T>
        implements Collection<T> {
    // compiling with JDK 8: FAIL
    // compiling with JDK 7:
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Java™ ORACLE®

Default methods: проблема

Нельзя так просто взять и добавить метод в интерфейс:

```
interface Collection<T> {
    /** @since 1.8 */
    void removeAll(Predicate<T> p);
}

class GooAbbaCollection<T>
        implements Collection<T> {
    // compiling with JDK 8: FAIL
    // compiling with JDK 7: OK
    // ...and then removeAll():
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: проблема

Нельзя так просто взять и добавить метод в интерфейс:

```
interface Collection<T> {
    /** @since 1.8 */
    void removeAll(Predicate<T> p);
}

class GooAbbaCollection<T>
        implements Collection<T> {
    // compiling with JDK 8: FAIL
    // compiling with JDK 7: OK
    // ...and then removeAll(): FAIL
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: решение

```
interface Collection<T> {
    /** @since 1.8 */
    default void removeAll(Predicate<T> p){
       // fallback implementation goes here
    }
}

class GooAbbaCollection<T>
          implements Collection<T> {
    // compiling with JDK 8: OK
    // compiling with JDK 7: OK
    // ...then removeAll(): OK
}
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

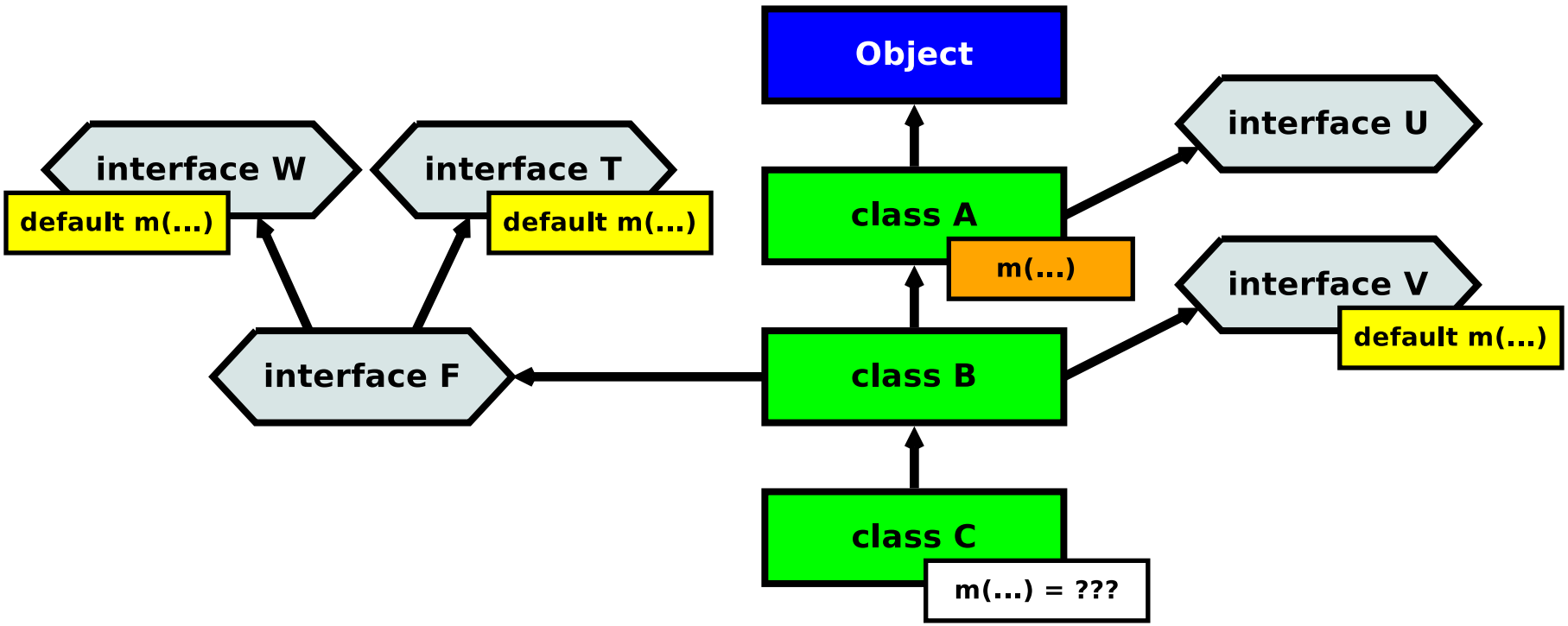
Default methods: свойства

```
interface Collection<T> {
  default void removeAll(Predicate<T> p){ };
}
```

- default-методы (почти) ничем не отличаются от виртуальных
- вызываются invokeinterface
- видны через Reflection
- могут наследоваться

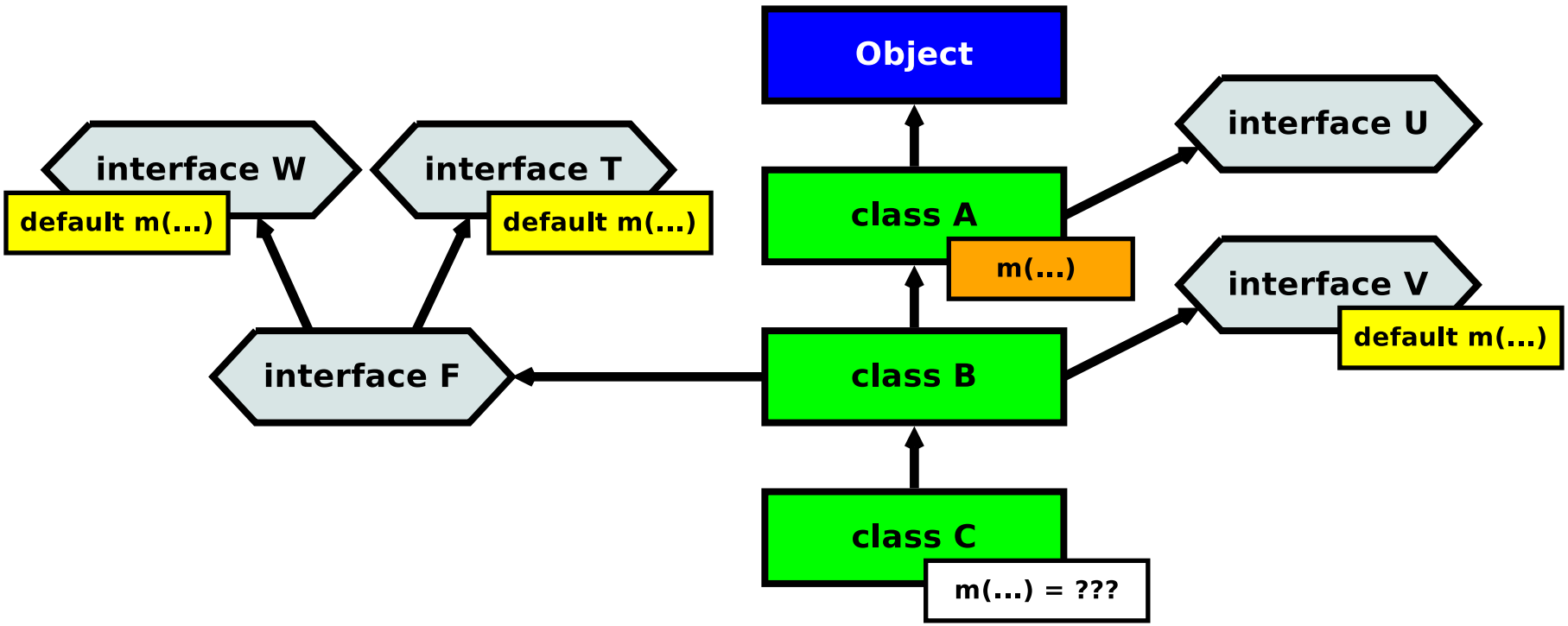
 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: наследование



Slide 22/45. Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: наследование



Всегда выигрывает конкретная реализация из иерархии классов.

Slide 22/45. Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default methods: abuse

- обеспечивают наследование *поведения*
- иногда используется для «*trait*»-ов

```
interface Op {
    boolean isStateful();
}
interface StatelessOp extends Op {
    default boolean isStateful() {
        return false;
    }
}
interface StatefulOp extends Op {
    default boolean isStateful() {
        return true;
    }
}
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Static methods

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Static methods: симметрия

- обычный джавовый стиль: interface + utility class:
  - `Collection` + `Collections`
  - `Predicate` + `Predicates` ← в JDK 8?
  - `Function` + `Functions` ← в JDK 8?
  - ...
- можно ли внести `static`-методы в интерфейс?
  - уже есть `static`-константы
  - `default`-методы уже вносят в интерфейсы `instance`-методы

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Static methods: пример

- позволяют тащить утилиты с собой
- например, фабрики:

```
public interface Ticket {
    String qDublin();
    static Ticket random() {
        return () -> "toDublin";
    }
}

assertEquals("toDublin",
             Ticket.random().qDublin());
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Мотивация

Зачем нам какие-то Stream'ы, когда и так всё хорошо?

```
public void printGroups(List<People> people) {
    Set<Group> groups = new HashSet<>();
    for (Person p : people) {
        if (p.getAge() >= 65)
            groups.add(p.getGroup());
    }
    List<Group> sorted = new ArrayList<>(groups);
    Collections.sort(sorted, new Comparator<Group>() {
        public int compare(Group a, Group b) {
            return Integer.compare(a.getSize(), b.getSize())
        }
    });
    for (Group g : sorted)
        System.out.println(g.getName());
}
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Мотивация

Было бы круто не городить километры одинакового кода:

```
public void printGroups(List<People> people) {
    people.stream()
        .filter(p -> p.getAge() > 65)
        .map(p -> p.getGroup())
        .distinct()
        .sorted(comparing(g -> g.getSize())
        .map(g -> g.getName())
        .forEach(n -> System.out.println(n));
}
```

Ещё хочется:

- работа со стандартными классами
- параллелизм?

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Дизайн

- Большинство кода укладывается в простой паттерн:

$$source$$

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Дизайн

- Большинство кода укладывается в простой паттерн:

$$source \rightarrow op$$

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Дизайн

- Большинство кода укладывается в простой паттерн:

$$source \rightarrow op \rightarrow op$$

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Дизайн

- Большинство кода укладывается в простой паттерн:

$$source \rightarrow op \rightarrow op \rightarrow op \rightarrow$$

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Дизайн

- Большинство кода укладывается в простой паттерн:

$$source \rightarrow op \rightarrow op \rightarrow op \rightarrow gangnamstyle$$

Java™ ORACLE

Slide 30/45. Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Дизайн

- Большинство кода укладывается в простой паттерн:

$$source \rightarrow op \rightarrow op \rightarrow op \rightarrow sink$$

- «sources»: collections, iterators, channels, ...
- «operations»: filter, map, reduce, ...
- «sinks»: collections, locals, ...

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: sources

- стандартные классы как источники?
  - + классы, которые ещё не написаны
  - + классы, написанные пользователями
- `Collection` не подходит
  - не всё же сначала в коллекцию заворачивать?
- `Iterable` не подходит
  - адовы проблемы с последовательными итераторами
  - методы для операций пачкают интерфейс
- `Stream` подходит
  - отдельная штука с нужной нам семантикой
  - «пачкаем» классы только методом `stream()`

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Stream

- Stream = «sequence of elements»
- выполнение операций отложено до последнего
- не мутирует источник
- одноразовый

```
public void printGroups(List<People> people) {
    Stream<People> s1 = people.stream();
    Stream<People> s2 = s1.filter(p -> p.getAge() > 65);
    Stream<Group>  s3 = s2.map(p -> p.getGroup());
    Stream<Group>  s4 = s3.distinct();
    Stream<Group>  s5 = s4.sorted(comparing(g -> g.getSize()));
    Stream<String> s6 = s5.map(g -> g.getName());
    s6.forEach(n -> System.out.println(n));
}
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Stream Sources

- коллекции:

```
List<T> list; Stream<T> s = list.stream();
```

- генераторы:

```
Stream<Integer> s =
    Stream.generate(() -> x++);
```

- утилиты:

```
IntStream s = IntStream.range(0, 100);
```

- etc:

```
Stream<String> s = bufferedReader.lines();
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Operations

- операции описывают действия над потоком
- два главных типа:
  1. *intermediate*: `Stream` $\rightarrow$ `Stream`
  2. *terminal*: `Stream` $\rightarrow$ PROFIT!

```
public void printGroups(List<People> people) {
   Stream<People> s = people.stream();

   Stream<String> s1 =
     s.filter(p -> p.getAge() > 65)
      .map(p -> p.getGroup())
      .distinct()
      .sorted(comparing(g -> g.getSize())
      .map(g -> g.getName());

   s1.forEach(n -> System.out.println(n));
}
```

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Sinks

- терминальные операции дают результат
- в зависимости от того, что уже в наборе операций:
  1. вычисляют lazily или eagerly
  2. параллельно или последовательно
- главные sink'и:
  1. *reducers*: `reduce`, `findFirst/Any`, etc
  2. *collectors*: сложить в коллекцию
  3. *forEach*: сделать действие над элементами
  4. *iterator*: вытаскивать результаты по одному

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Sinks/Reducers

- берут поток и дают некоторый скаляр:

```
int s =
    stream.reduce(0, (x, y) -> x + y);
```

Slide 36/45. Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Sinks/Reducers

- берут поток и дают некоторый скаляр:

```
int s =
    stream.reduce(0, (x, y) -> x + y);
```

- некоторые отдают Optional<T>, чтобы отличать пустоту:

```
Optional<Integer> o =
    stream.reduce((x, y) -> x + y);
Integer i =
    stream.reduce(0, (x, y) -> x + y);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Sinks/Collectors

- складывают содержимое потока:

```
List<Integer> list =
    IntStream.range(0, 100).boxed()
        .collect(Collectors.toList());
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Sinks/Collectors

- складывают содержимое потока:

```
List<Integer> list =
  IntStream.range(0, 100).boxed()
      .collect(Collectors.toList());
```

- могут принимать сложные коллекции:

```
Map<Integer,Integer> map =
  IntStream.range(0, 1000).boxed()
      .collect(
          Collectors.toConcurrentMap(
            (k) -> k % 42,
            Functions.identity(),
            Collectors.lastWinsMerger()
          )
      );
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Sinks, forEach/iterator

- делают действие над каждым элементом потока:

```
IntStream.range(0, 100)
    .forEach(System.out::println);
```

- можно вытаскивать элементы из потока по очереди:

```
Iterator<Integer> =
   IntStream.range(0, 100).iterator();
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Lazy vs. Eager

- Lazy: тянет элементы из источника по одному
  - iterator
- Eager: обрабатывает весь поток разом
  - почти все reducer'ы
  - почти все collector'ы
  - forEach

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: Short-circuting

- некоторые операции могут «бросить» поток
- даже в eager-режиме!
- получают смысл операции над бесконечными потоками

- примеры: `findFirst`, `findAny`

```
int v = Stream.generate(() -> x++)
                .findFirst().get();
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: parallelism

- многие источники хорошо бьются на части
- многие операции хорошо параллелизуются
- библиотека делает всю работу за нас
- «под капотом» используется `ForkJoinPool`
- но: нужно эксплицитно просить библиотеку

```
int v = list.parallelStream()
                .reduce(Math::max)
                .get();
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: explicit parallelism

Q: Почему не имплицитно?

A: Выигрыш сильно зависит от:

- N – количества элементов в источнике
- Q – стоимости операции над элементом
- P – доступного паралеллизма на машине
- C – количества конкуррентных клиентов

Точно знаем только N.

Неплохо представляем себе P.

Умеем худо-бедно справляться с C.

Q очень сложно оценить.

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Streams: демо

Parallel vs. Sequential

https://github.com/shipilev/jdk8-lambda-samples/blob/internal/src/main/java/com/oracle/streams/SeqParBench.java

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ресурсы

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ресурсы: Полезные ссылки

- Project Lambda:
  http://openjdk.java.net/projects/lambda/
- Binary builds:
  http://jdk8.java.net/lambda
- Mailing list:
  lambda-dev@openjdk.java.net
- Talk samples:
  https://github.com/shipilev/jdk8-lambda-samples

ORACLE

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.