ORACLE

JDK 8: Я, Лямбда

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.

Введение

Введение: про доклады

JDK8: Я, лямбда: (← вы здесь)

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

JDK8: Молот лямбд:

- 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

Введение: вопросы

- Что?

Введение: вопросы

- Что?
- Как?

Введение: вопросы

- Что?
- Как?
- Зачем?

Введение: вопросы

- Что?
- Как?
- Зачем?
- ~~Почему?~~

Введение: вопросы

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

Введение: λ samples code

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

Lambda

Lambda - ?

Lambda:

- анонимная функция

λ : JDK N, where N $<$ 8

```
new Comparator<Integer>() {

  @Override
  public int compare(Integer x, Integer y) {
    return (x < y) ? -1
                   : (x > y) ? 1 : 0;
    }

};
```

λ : JDK N, where N $\geqslant$ 8

```
(x, y) -> (x < y) ? -1
                  : (x > y) ? 1 : 0
```

Lambda:

- ~~анонимная функция~~
- выражение, описывающее анонимную функцию

λ : functional interface

Функциональный интерфейс

Интерфейс, содержащий единственный абстрактный метод.

- a.k.a. SAM (Single Abstract Method)
- e.g.
  - java.lang.Runnable
  - java.util.Comparator
  - java.awt.event.ActionListener

Lambda:

- ~~анонимная функция~~
- ~~выражение, описывающее анонимную функцию~~
- выражение, описывающее анонимную функцию, результатом исполнения которого является *некоторый объект*, реализующий требуемый функциональный интерфейс

Lambda:

- ~~анонимная функция~~
- ~~выражение, описывающее анонимную функцию~~
- выражение, описывающее анонимную функцию, результатом исполнения которого является *объект* неизвестной природы, реализующий требуемый функциональный интерфейс

λ : examples

```
// don't try this at home
Comparator<Integer> cmp = (x, y) -> x - y;
```

```
// BinaryOperator<T> - T apply(T t, T u)
BinaryOperator<Integer> sub =
                                (x, y) -> x - y;
```

```
// BiFunction<T, U, R> - R apply(T t, U u)
BiFunction<Integer,Integer,Integer> biSub =
                                (x, y) -> x - y;
```

ORACLE

syntax: все вокруг →

```
Comparator<Integer> cmp =

        (x, y) ->
                (x < y) ? -1 : (x > y) ? 1 : 0;
```


syntax: явное указание типов

```
Comparator<Integer> cmp =

    (Integer x, Integer y) ->
            (x < y) ? -1 : (x > y) ? 1 : 0;
```


syntax: без аргументов

```
// Supplier<T> - T get();

Supplier<Integer> ultimateAnswerFactory =

                        () -> 42;
```

ORACLE

syntax: с единственным аргументом

```
// Function<T, R> - R apply(T t);

Function<String, Integer> f0 =

        (String s) -> Integer.parseInt(s);

Function<String, Integer> f1 =

                (s) -> Integer.parseInt(s);

Function<String, Integer> f2 =

                    s -> Integer.parseInt(s);
```

ORACLE

syntax: все ли тут правильно?

```
Comparator<Integer> cmp =

    (x, y) -> (x < y) ? -1
                      : (x == y) ? 0 : 1;
```

syntax: все ли тут правильно?

```
Comparator<Integer> cmp =

    (x, y) -> (x < y) ? -1
                      : (x == y) ? 0 : 1;
```

syntax: блок как тело

```
Comparator<Integer> cmp =

    (x, y) -> (x < y) ? -1
                      : (x == y) ? 0 : 1;


Comparator<Integer> rightCmp = (a, b) -> {
     int x = a;
     int y = b;
     return (x < y) ? -1
                    : (x == y) ? 0 : 1;
};
```

ORACLE

syntax: блок как тело

```
// Supplier<T> - T get();

Supplier<Integer> deepThought  =

() -> {
     long toThinkInMillis =
          TimeUnit.DAYS.toMillis(2737500000L);
     Thread.sleep(toThinkInMillis);
     return 42;
};
```


syntax: lambda без результата

```
// Consumer<T>  -  void accept(T t);

Consumer<String> c =
        s -> { System.out.println(s);};

Arrays.asList("Foo", "Bar").forEach(c);

Arrays.asList("Foo", "Bar", "Baz")
        .forEach(s -> System.out.println(s));
```

ORACLE

syntax: variable hiding запрещен!

```
public void foo() {
    int x = 42;

    Comparator<Integer> cmp =
        (x, y) -> (x < y) ? -1
                          : (x > y) ? 1 : 0;
    ...
```

Compile Error!

syntax: variable hiding запрещен!

```
public void foo() {
    int x = 42;

    Comparator<Integer> cmp = (a, b) -> {
        int x = a;
        int y = b;
        return (x < y) ? -1
                       : (x == y) ? 0 : 1;
    };
    ...
```

Compile Error!

usage: Где?

оператор присваивания
(правая часть)

e.g.

```
FI f = () -> 42;
```


usage: Где?

return

e.g. `return () -> 42;`

usage: Где?

аргумент
метода/конструктора

e.g `foo(1 , () -> 42)`

usage: Где?

инициализатор массива

```
e.g.  FI[] fis = { () -> 41, () -> 42 };
```

ORACLE

usage: Где?

cast

e.g. `(FI)() -> 42`

capture: in anonymous classes

```
public Comparator<Integer> makeComparator(){
  final int less = -1;
  final int equal = 0;
  final int greater = 1;

  return new Comparator<Integer>(){
    @Override
    public int compare(Integer x, Integer y){
      return (x < y) ? less
                     : (x > y) ? greater
                               : equal;
    }
  };
}
```

capture: in lambdas

```
public Comparator<Integer> makeComparator(){
   int less = -1;
   int equal = 0;
   int greater = 1;

   return (x, y) ->
                  (x < y) ? less
                          : (x > y) ? greater
                                    : equal;
}
```


capture: effective final

lambda может использовать любые внешние переменные, если они являются effective final

Effective final

Локальная переменная, не меняющая своего значения

a.k.a. final переменная «de facto»

capture: effective final

```
public void foo() {
    int cnt = 0;

    Supplier<Integer> counter = () -> cnt++;
    ...
```

Compile Error!

capture: bonus

Effective final in anonymous classes

```
public Comparator<Integer> makeComparator(){
   int less = -1;
   int equal = 0;
   int greater = 1;

   return new Comparator<Integer>(){
      @Override
      public int compare(Integer x, Integer y){
         return (x < y) ? less
                        : (x > y) ? greater
                                  : equal;
      }
   };
}
```

ORACLE

capture: рекурсивные лямбды

```
public class Fibonacci {
  //IntUnaryOperator  - int applyAsInt(int)

  IntUnaryOperator fib_instance =
    (n) -> (n < 2) ? n :
           fib_instance.applyAsInt(n - 1) +
           fib_instance.applyAsInt(n - 2);

  static IntUnaryOperator fib_static =
    (n) -> (n < 2) ? n :
           fib_static.applyAsInt(n - 1) +
           fib_static.applyAsInt(n - 2);
```


capture: рекурсивные лямбды

```
public IntUnaryOperator makeLocalFib() {
  IntUnaryOperator fib_local =
      (n) -> (n < 2) ? n :
             fib_local.applyAsInt(n - 1) +
             fib_local.applyAsInt(n - 2);

      return fib_local;
}
```

Compile Error!

Method Reference

example: good?

```
Comparator<Integer> cmp =

        (x, y) -> (x < y) ? -1
                          : (x > y) ? 1 : 0;
```

example: do not reinvent the wheel

```
//Integer:
//  public static int compare(int x, int y)

Comparator<Integer> cmp =

        (x, y) -> Integer.compare(x, y);
```


method reference: example

```
//Integer:
//  public static int compare(int x, int y)

Comparator<Integer> cmp = Integer::compare;
```

method reference: bounded

```
//PrintStream - public void println(String s)

//Consumer<T> - void accept(T t);

Arrays.asList("Foo", "Bar", "Baz")
      .forEach(System.out::println);
```

method reference: bounded

```
// Predicate<T>  -  boolean test(T t);

Predicate<String> newMatcher(String pattern)
{
    return pattern::equalsIgnoreCase;
}

assertTrue(newMatcher("true").test("TruE"))

assertTrue(newMatcher("false").test("FalsE"))

assertFalse(newMatcher("true").test("FalsE"))
```


method reference: bounded

```
// Predicate<T>  -  boolean test(T t);

Predicate<String> isTrue =
        "true"::equalsIgnoreCase;

assertTrue(isTrue.test("TruE"))

assertFalse(isTrue.test("FalsE"))
```

ORACLE

method reference: unbounded

```
// Comparator<T>:
//      int compare(T o1, T o2);

// Integer:
//      int compareTo(Integer anotherInteger)

Comparator<Integer> cmp = Integer::compareTo;
```


method reference: example

```
// Function<T, R> - R apply(T t);

Function<String, Integer> f0 =
                                Integer::parseInt;
```

method reference: to constructor

```
// Function<T, R> - R apply(T t);

Function<String, Integer> f0 =
                                Integer::parseInt;

// Integer - public Integer(String s)

Function<String, Integer> f1 = Integer::new;
```

ORACLE

method reference: to constructor

```
public class Counter {
    private int c = 0;
    public Counter()        { this(0); }
    public Counter(int c) { this.c = c; }
    public int inc()        { return ++c; }
    public int get()        { return c;   }
}

Supplier<Counter> f = Counter::new;
assertEquals(0, f.get().get());
assertEquals(1, f.get().inc());
```


method reference: to constructor

```
public class Counter {
    private int c = 0;
    public Counter()      { this(0); }
    public Counter(int c) { this.c = c; }
    public int inc()      { return ++c; }
    public int get()      { return c;   }
}

Function<Integer, Counter> f = Counter::new;
assertEquals(1,  f.apply(1).get());
assertEquals(42, f.apply(42).get());
```


Serialization

example: unsigned int set

```
NavigableSet<Integer> set = new TreeSet<>(
    (x, y) -> Integer.compareUnsigned(x, y)
);

set.addAll(Arrays.asList(-100, 0, 100));

assertEquals(0,  set.first());

assertEquals(-100, set.last());
```

example: unsigned int set

```
NavigableSet<Integer> set = new TreeSet<>(
    (x, y) -> Integer.compareUnsigned(x, y)
);

try {
    ObjectOutputStream stream = ...
    ...
    stream.writeObject(set);
} catch (NotSerializableException e) {
    we are here :(
} ...
```


serialization: ЧТО ДЕЛАТЬ?

```
NavigableSet<Integer> set = new TreeSet<>(
    (x, y) -> Integer.compareUnsigned(x, y)
);

try {
    ObjectOutputStream stream = ...
    ...
    stream.writeObject(set);
} catch (NotSerializableException e) {
    we are here :(
} ...
```


serialization: «type intersection»

```
NavigableSet<Integer> set = new TreeSet<>(
    (Comparator<Integer> & Serializable)
    (x, y) -> Integer.compareUnsigned(x, y)
);

try {
    ObjectOutputStream stream = ...
    ...
    stream.writeObject(set);
} catch (NotSerializableException e) {
    we are not here :)
} ...
```

serialization: marker interface

Маркер-интерфейс

Интерфейс, не содержащий абстрактных методов.

- a.k.a. ZAM (Zero Abstract Methods)

serialization: marker interface

Маркер-интерфейс

Интерфейс, не содержащий абстрактных методов.

- a.k.a. ZAM (Zero Abstract Methods)

type intersection

$(SAM\ \&\ ZAM_0\ \&\ ZAM_1\ \&\ ...\ \&\ ZAM_k)$

serialization: method reference

```
NavigableSet<Integer> set = new TreeSet<>(
    (Comparator<Integer> & Serializable)
    Integer::compareUnsigned
);

try {
    ObjectOutputStream stream = ...
    ...
    stream.writeObject(set);
} catch (NotSerializableException e) {
    we are not here :)
} ...
```

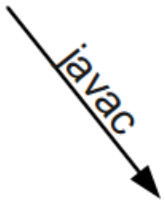
ORACLE

Детали реализации


```
void m() {
    int y = 3;
    Function<Integer,Integer> f = x -> x + y;
    f.apply(2);
}
```

Naïve desugaring

```
void m() {
    int y = 3;
    Function<Integer,Integer> f = x -> x + y;
    f.apply(2);
}
```



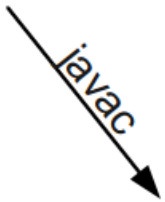

```
void m() {
    int y = 3;
    Function<Integer,Integer> f = new A$1(y);
    f.apply(2);
}

class A$1 implements Function<Integer,Integer> {
    private final int y;
    A$1(int y) { this.y = y; }

    public Integer apply(Integer x) {
        return x + y;
    }
}
```


Project Lambda ABI

```
void m() {
    int y = 3;
    Function<Integer,Integer> f = x -> x + y;
    f.apply(2);
}
```

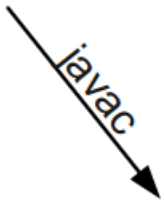



```
static Integer lambda$1(int y, Integer x) {
    return x + y;
}

void m() {
    int y = 3;
    Function<Integer,Integer> f = λ-factory;
    f.apply(2);
}
```


Project Lambda ABI

```
void m() {
    int y = 3;
    Function<Integer,Integer> f = x -> x + y;
    f.apply(2);
}
```




```
static Integer lambda$1(int y, Integer x) {
    return x + y;
}

void m() {
    int y = 3;
    Function<Integer,Integer> f =
            INDY[ j.l.i.LambdaMetaFactory,
                  MT[Function.apply],
                  MH[lambda$1]
                ](y);
    f.apply(2);
}
```


INDY mechanics

```
Function<Integer,Integer> f =
    INDY[ j.l.i.LambdaMetaFactory,
          MT[Function.apply],
          MH[lambda$1]
        ](y);
```

INDY mechanics

```
Function<Integer,Integer> f =
    INDY[ j.l.i.LambdaMetaFactory,
          MT[Function.apply],
          MH[lambda$1]
        ](y);
```


```
1) (once) execute
     j.l.i.LambdaMetaFactory(MT[Function.apply],MH[lambda$1]);

2) (once) store result to
     vmstatic CallSite CS;

3) execute
     Function<Integer,Integer> f = CS.get().invoke(y);
```

HotSpot implementation

```
j.l.i.LambdaMetaFactory(MT[Function.apply],MH[lambda$1])
```

HotSpot implementation

```
j.l.i.LambdaMetaFactory(MT[Function.apply],MH[lambda$1]) {
    generate(...);
```


```
class A$1 implements Function<Integer,Integer> {
    private final int y;
    A$1(int y) { this.y = y; }

    public Integer apply(Integer x) {
        return lambda$1(y, x);
    }
}
```

HotSpot implementation

```
j.l.i.LambdaMetaFactory(MT[Function.apply],MH[lambda$1]) {
    generate(...);
    return new CallSite(MH[A$1#new]);
}
```

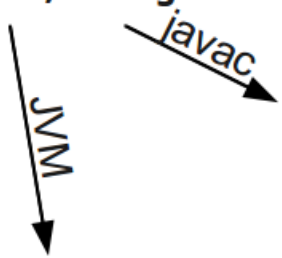
```
class A$1 implements Function<Integer,Integer> {
    private final int y;
    A$1(int y) { this.y = y; }

    public Integer apply(Integer x) {
        return lambda$1(y, x);
    }
}
```

```
CS.get().invoke(y);  ~  return new A$1(y);
```

Non-capturing lambda

```
void m2() {
    Function<Integer,Integer> f = x -> x + 3;
    f.apply(2);
}
```




```
static Integer lambda$23(Integer x) {
    return x + 3;
}
```

```
static A$23$INSTANCE = new A$23();

class A$23 implements Function<Integer,Integer>{

    A$23() {}

    public Integer apply(Integer x) {
        return lambda$23(x);
    }
}
```

```
CS.get().invoke();  ~  return A$23$INSTANCE;
```


Ресурсы

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/kuksenko/jdk8-lambda-samples

ORACLE

