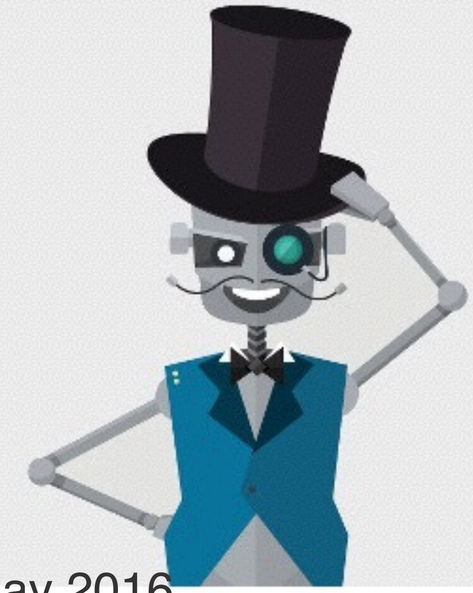
Beyond Lambdas, the aftermath

@DanielSawano
@DanielDeogun



Deogun - Sawano, 11-13 May 2016

geecon

About us…

Daniel Deogun

Daniel Sawano

omega point.

Stockholm - Gothenburg - Malmoe - Umea - New York

omega point.

[inert code here]

Optionals 1

```
7   final Oracle oracle = new Oracle();
8
9   String _() {
10      final Advise advise = currentAdvise();
11
12      if (advise != null) {
13          return advise.cheap();
14      }
15      else {
16          return oracle.advise().expensive();
17      }
18  }
```

omega point.

Optionals 1

```
25  final Oracle oracle = new Oracle();
26
27  String _() {
28      final Advise advise = currentAdvise();
29
30      return Optional.ofNullable(advise)
31                     .map(Advise::cheap)
32                     .orElse(oracle.advise().expensive());
33  }
```

omega point.

Optionals 1

```
25  final Oracle oracle = new Oracle();
26
27  String _() {
28      final Advise advise = currentAdvise();
29
30      return Optional.ofNullable(advise)
31                     .map(Advise::cheap)
32                     .orElseGet( () -> oracle.advise().expensive());
33  }
```

omega point.

Optionals 2

```
26  String _(final Optional<String> optOfSomeValue) {
27
28      return optOfSomeValue.map(v -> calculate(v))
29                           .filter(someCriteria())
30                           .map(v -> transform(v))
31                           .orElseGet(() -> completelyDifferentCalculation());
32
33  }
```

omega point.

Optionals 2

```
26  String _(final Optional<String> optOfSomeValue) {
27
28      if (optOfSomeValue.isPresent()) {
29          final String calculatedValue = calculate(optOfSomeValue.get());
30          if (someCriteria().test(calculatedValue)) {
31              return transform(calculatedValue);
32          }
33      }
34
35      return completelyDifferentCalculation();
36
37  }
```

omega point.

Optionals 2

```
26  String _() {
27      return value()
28              .flatMap(v -> firstCalculation(v))
29              .orElseGet(() -> completelyDifferentCalculation());
30  }
31
32  Optional<String> value() {
33      return Optional.of(someValue());
34  }
35
36  Optional<String> firstCalculation(final String v) {
37      return Optional.of(calculate(v))
38                      .filter(someCriteria())
39                      .map(value -> transform(value));
40  }
```

omega point.

Optionals 3

```
27  <T> void _(final Optional<T> argument) {
28      argument.map(a -> doSomething(a));
29  }
```

omega point.

Optionals 3

```
25  <T> void _(final T argument) {
26      if (argument != null) {
27          doSomething(argument);
28      }
29  }
```

omega point.

Optionals 3

```
26  <T> void _(final T argument) {
27      doSomething(notNull(argument));
28  }
```

omega point.

Streams 1

```
30  @Test
31  public void _() {
32
33      final Stream<String> stream = elements().stream()
34                                                .sorted();
35
36      final String result = stream.collect(joining(","));
37
38      assertEquals("A,B,C", result);
39
40  }
41
42  static List<String> elements() {
43      return asList("C", "B", null, "A");
44  }
```

Streams 1

```
31  @Test
32  public void _() {
33
34      final Stream<String> stream = elements().stream()
35                                              .filter(Objects::nonNull)
36                                              .sorted();
37
38      final String result = stream.collect(joining(","));
39
40      assertEquals("A,B,C", result);
41
42  }
43
44  static List<String> elements() {
45      return asList("C", "B", null, "A");
46  }
```

Streams 2

```
27  @Test
28  public void _() {
29
30      final long idToFind = 6;
31      final Predicate<Item> idFilter = item -> item.id().equals(idToFind);
32
33      service().itemsMatching(idFilter)
34               .findFirst()
35               .ifPresent(Support::doSomething);
36
37  }
```

omega point.

Streams 2

```
28  @Test
29  public void _() {
30
31      final long idToFind = 6;
32      final Predicate<Item> idFilter = item -> item.id().equals(idToFind);
33
34      service().itemsMatching(idFilter)
35               .reduce(toOneItem())
36               .ifPresent(Support::doSomething);
37
38  }
39
40  BinaryOperator<Item> toOneItem() {
41      return (item, item2) -> {
42          throw new IllegalStateException("Found more than one item with the same id");
43      };
44  }
```

omega point.

Streams 3

```
29  private final UserService userService = new UserService();
30  private final OrderService orderService = new OrderService();
31
32  @Test
33  public void _() {
34      givenALoggedInUser(userService);
35
36      itemsToBuy().stream()
37                  .map(item -> new Order(item.id(), currentUser().id()))
38                  .forEach(orderService::sendOrder);
39
40      System.out.println(format("Sent %d orders", orderService.sentOrders()));
41  }
42
43  User currentUser() {
44      final User user = userService.currentUser();
45      validState(user != null, "No current user found");
46      return user;
47  }
```

omega point.

Streams 3

```
29  private final UserService userService = new UserService();
30  private final OrderService orderService = new OrderService();
31
32  @Test
33  public void _() {
34      givenALoggedInUser(userService);
35
36      final User user = currentUser();
37      itemsToBuy().parallelStream()
38                  .map(item -> new Order(item.id(), user.id()))
39                  .forEach(orderService::sendOrder);
40
41      System.out.println(format("Sent %d orders", orderService.sentOrders()));
42  }
43
44  User currentUser() {
45      final User user = userService.currentUser();
46      validState(user != null, "No current user found");
47      return user;
48  }
```

omega point.

Lambdas 1

```
28  static Integer numberOfFreeApples(final User user,
29                                    final Function<User, Integer> foodRatio) {
30      return 2 * foodRatio.apply(user);
31  }
32
33  @Test
34  public void _() {
35
36      final Function<User, Integer> foodRatioForVisitors = u -> u.age() > 12 ? 2 : 1;
37
38      final int numberOfFreeApples = numberOfFreeApples(someUser(), foodRatioForVisitors);
39
40      System.out.println(format("Number of free apples: %d", numberOfFreeApples));
41
42  }
```

omega point.

Lambdas 1

```
29  @Test
30  public void _() {
31
32      final Function<User, Integer> foodRatioForVisitors = u -> u.age() > 12 ? 2 : 1;
33      final Function<User, Integer> age = User::age;
34
35      final int numberOfFreeApples_1 = numberOfFreeApples(someUser(), foodRatioForVisitors);
36      final int numberOfFreeApples_2 = numberOfFreeApples(someUser(), age); // This is a bug!
37
38      System.out.println(format("Number of free apples (1): %d", numberOfFreeApples_1));
39      System.out.println(format("Number of free apples (2): %d", numberOfFreeApples_2));
40
41  }
```

Lambdas 1

```
28      @FunctionalInterface
29      interface FoodRatioStrategy {
30
31          Integer ratioFor(User user);
32      }
33
34      static Integer numberOfFreeApples(final User user,
35                                        final FoodRatioStrategy ratioStrategy) {
36          return 2 * ratioStrategy.ratioFor(user);
37      }
38
39      @Test
40      public void _() {
41
42          final FoodRatioStrategy foodRatioForVisitors = user -> user.age() > 12 ? 2 : 1;
43          final Function<User, Integer> age = User::age;
44
45          final Integer numberOfFreeApples_1 = numberOfFreeApples(someUser(), foodRatioForVisitors);
46          //final Integer numberOfFreeApples_2 = numberOfFreeApples(someUser(), age);
47
48          System.out.println(format("Number of free apples (1): %d", numberOfFreeApples_1));
49      }
```

omega point.

Lambdas 2

```
25  @Test
26  public void should_build_tesla() {
27
28      assertEquals(1000, new TeslaFactory().createTesla().engine().horsepower());
29
30  }
31
32  @Test
33  public void should_build_volvo() {
34
35      assertEquals(250, new VolvoFactory().createVolvo().engine().horsepower());
36
37  }
```

Lambdas 3

```
29  @Test
30  public void _() {
31
32      final List<Integer> values = asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
33
34      allEvenNumbers(values);
35
36      System.out.println("Hello");
37
38  }
39
40  static List<Integer> allEvenNumbers(final List<Integer> values) {
41      return values.stream()
42                   .filter(Support::isEven)
43                   .collect(toList());
44  }
```

omega point.

Lambdas 3

```
31  @Test
32  public void _() {
33
34      final List<Integer> values = asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
35
36      final Supplier<List<Integer>> integers = () -> allEvenNumbers(values);
37
38      System.out.println(integers.get());
39
40  }
41
42  static List<Integer> allEvenNumbers(final List<Integer> values) {
43      return values.stream()
44                   .filter(Support::isEven)
45                   .collect(toList());
46  }
```

omega point.

Lambdas 4

```
24  private final String pattern;
25
26  public _14(final String pattern) {
27      this.pattern = pattern;
28  }
29
30  public List<String> allMatchingElements(final List<String> elements) {
31      return elements.stream()
32                     .filter(e -> e.contains(pattern))
33                     .collect(toList());
34  }
```

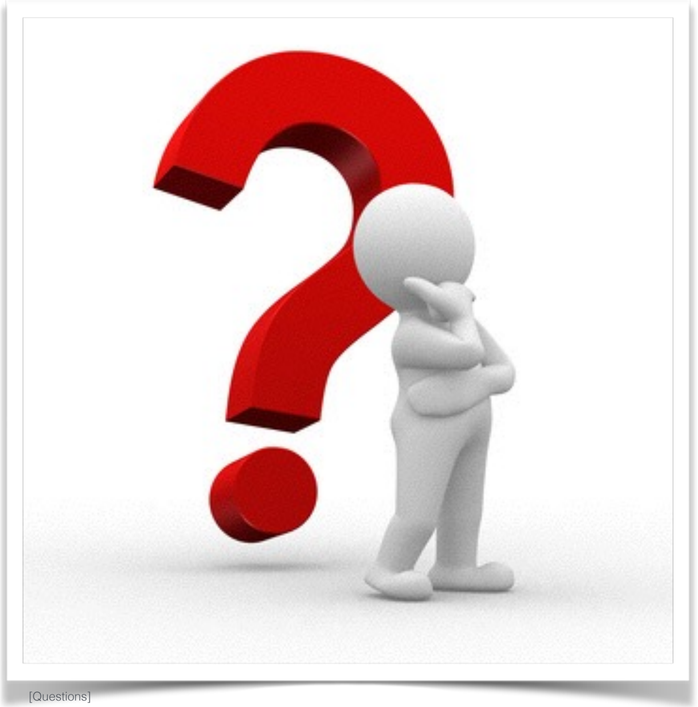
omega point.

Lambdas 4

```
25  private final String pattern;
26
27  public _14(final String pattern) {
28      this.pattern = pattern;
29  }
30
31  public List<String> allMatchingElements(final List<String> elements) {
32      return elements.stream()
33                     .filter(matches(pattern))
34                     .collect(toList());
35  }
36
37  private Predicate<String> matches(final String pattern) {
38      return e -> e.contains(pattern);
39  }
```

Q&A

omega point.



[Questions]

omega point.

Code examples can be found here:
https://github.com/sawano/beyond-lambdas-the-aftermath

omega point.

Thank you!
@DanielSawano @DanielDeogun