
ORACLE®

Where far art thou…() -> Lambda

Coherence SIG, London

Harvey Raja
Consulting Member Technical Staff
Coherence
July, 2015

ORACLE

Safe Harbor Statement

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.

Persistence

1. Java 8
2. Lambdas
3. Infiltrating Lambdas
4. Streams
5. Continuous Demo…

Java 8

- Released 03/2014!
- Lambdas
- Streams
- Improvements to G1
- Default methods
- Date / Time improvements
- Nashorn

Lambdas

- Introduce functions as a first class citizen to the language
- Facilitates functional programming constructs
- Any interface or class with a Single Abstract Method can be defined as a Lambda
- Last expression is the return argument (non-void methods)
- Surrounding arguments can be captured
- A reference to a method (including constructors) can be captured as a Lambda

Lambdas

```
Supplier<String>          s = () -> "Yikes";
Consumer<String>          c = sIn -> System.out.println(sIn);
Function<String, Integer> f = sIn -> Integer.parseInt(sIn);
Predicate<String>         p = sIn -> s.equals("Yikes");
```

```
BiConsumer<String, Integer>          bc    = (sIn, nIn) -> System.out.println(sIn + nIn);
BiFunction<String, Integer, Integer> bf    = (sIn, nIn) -> Integer.parseInt(sIn) + nIn;
BinaryOperator<Integer>              bfInt = (nIn1, nIn2) -> nIn1 + nIn2;
BiPredicate<String, Integer>         bp    = (sIn, nIn) -> sIn.equals("Yikes") && nIn == 42;
```

```
Supplier<String>                    mhGetConstant  = Person::getConstant;
Function<Person, String>            mhGetFirstName = Person::getFirstName;
Supplier<Person>                    mhCons         = Person::new;
BiFunction<String, Integer, Person> mhCons2        = Person::new;
```

Lambdas

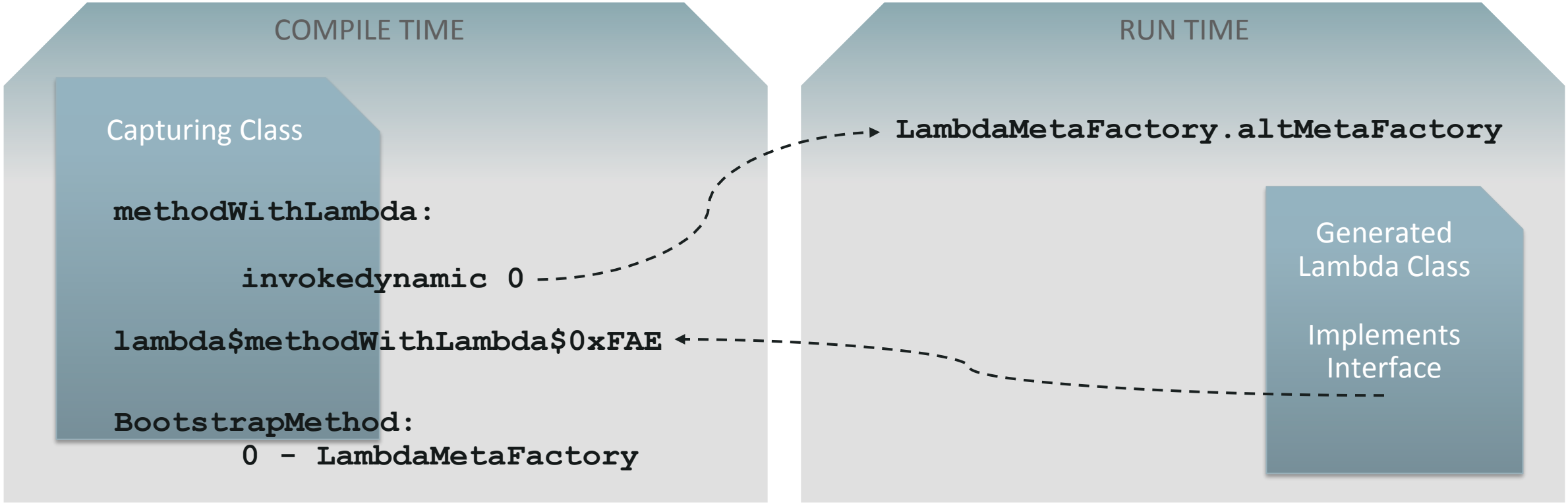
```
String            sMatch = "Yikes";
Predicate<String> p2     = sIn -> s.equals(sMatch);
```

```
List<String> listHarNames = new ArrayList<>(listNames.size());
listNames.forEach(sName ->
    {
    if (sName.startsWith("Har"))
        {
        listHarNames.add(sName);
        }
    });
```

```
Map<String, Long> map = new HashMap<>();
map.computeIfAbsent("—|—", sKey -> veryExpensiveOp(sKey));
```

Lambdas – under the hood

- Compiler converts lambda expression into a number of parts:



Lambdas – under the hood

- InvokeDynamic is beautiful
  - Introduces the notion of linkage time
  - Allows for different types of call sites to be returned allowing the JVM to optimize for ConstantCallSites
- LambdaMetaFactory dynamically generates a class implementing the SAM
- SAM implementation will call generated synthetic method on capturing class
- The class is defined ‘anonymously’

Lambdas – under the hood

- Lambdas can be serialized but must implement Serializable
- A Serializable lambda adds behaviour to the class:
  - writeReplace/readResolve
  - SerializedLambda
- Metadata for the lambda and captured arguments are serialized
- Type conversion as interface types can be narrowed at the call site / captured class
  - Slightly more involved for primitives – boxed & unboxed

Hardware and Software
Engineered to Work Together


ORACLE®