Проект Amber

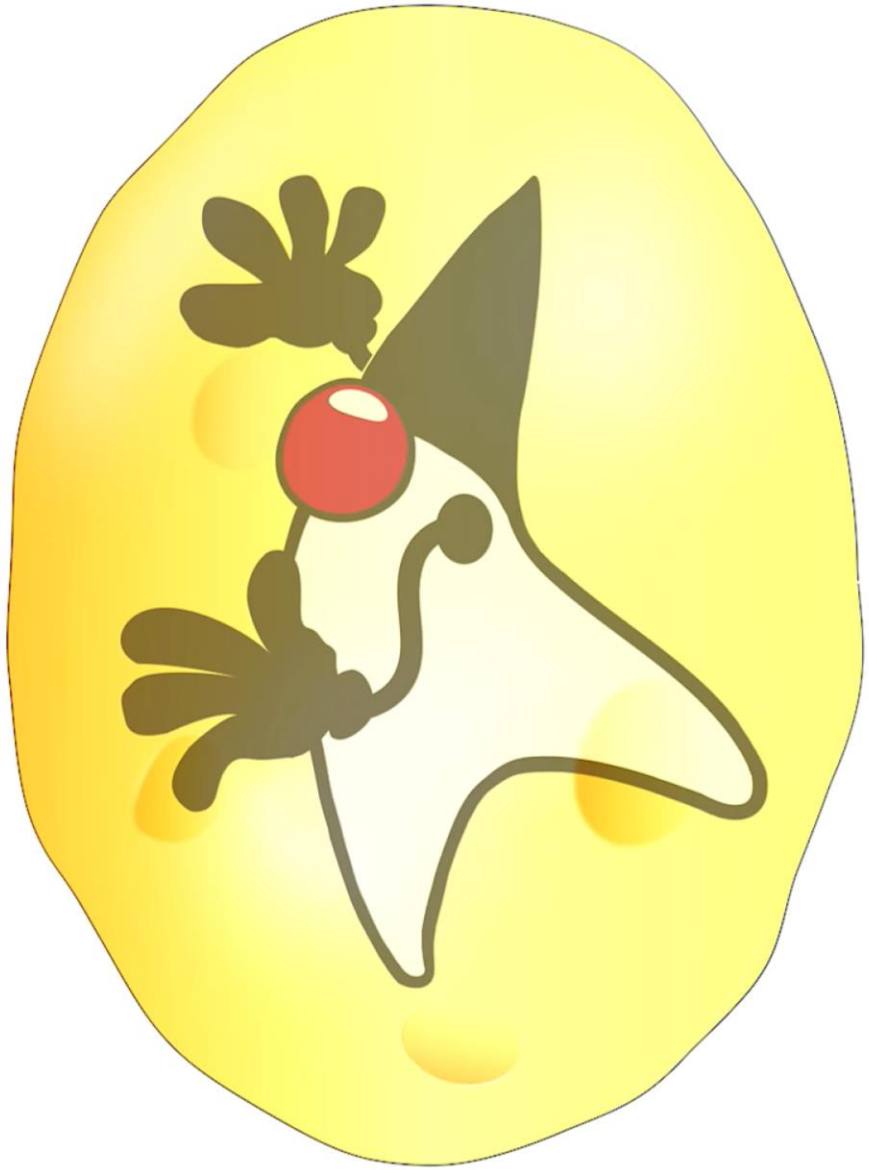
Вести с полей

Тагир Валеев

Safe harbor statement

Всё, что будет сказано, реализовано только в *экспериментальной версии* языка Java. Никто не гарантирует, что хоть что-то из этого попадёт в официальный релиз. Любые детали реализации могут измениться произвольным образом или вообще исчезнуть.

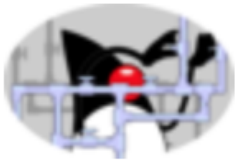
Я вас предупредил!



Constants in Amber with Vicente Romero and Paul Sandoz
https://www.youtube.com/watch?v=XbEC0jW1fDY

Где брать?

http://hg.openjdk.java.net/amber/



OpenJDK / amber
repositories

Качать только это

amber	Amber prototype	amber-dev@openjdk.java.net
amber10-old	[READ-ONLY] Amber prototype (old topology)	amber-dev@openjdk.java.net
corba		
hotspot		
jaxp		
jaxws		
jdk		
langtools		
nashorn		

Старые репозитории до слияния, сюда не смотреть

Amber JEPs

JEP	JIRA Issue	Branch	Name
286	8151454	lvti	Local-Variable Type Inference
301	8170351	enhanced-enums	Enhanced Enums
302	8170361	lambda-leftovers	Lambda Leftovers
303	8178320	condy-folding	Intrinsics for the LDC and INVOKEDYNAMIC Instructions
309	8177279	condy	Dynamic constants in the JVM
305	8181287	pattern	Pattern Matching

JEP 286:
Local Variable Type Inference

```java
int i = 2017;
List<String> list = Arrays.asList("Joker", "JPoint");
String s = " Hello ";
System.out.println(s.trim() + ", " + list.get(0) + "-" + i);
```

```java
int i = 2017;
List<String> list = Arrays.asList("Joker", "JPoint");
String s = " Hello ";
System.out.println(s.trim() + ", " + list.get(0) + "-" + i);
```

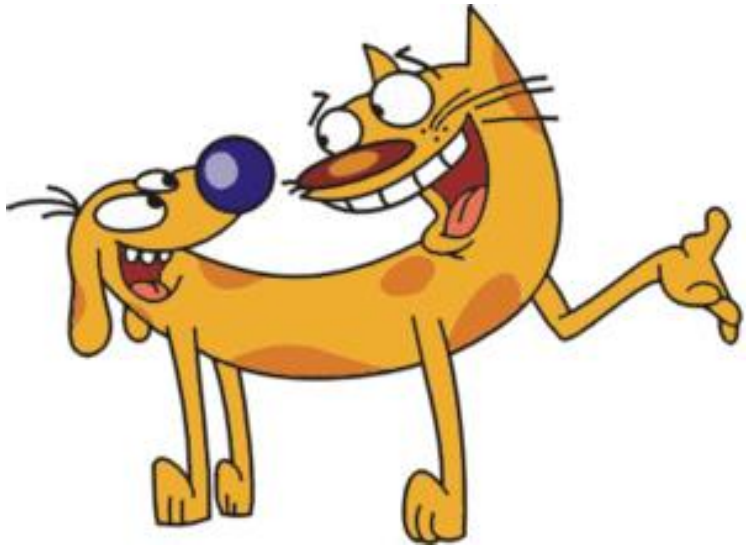
```java
var i = 2018;
var list = Arrays.asList("Joker", "JPoint");
var s = " Hello ";
System.out.println(s.trim() + ", " + list.get(0) + "-" + i);
```

```
<T> void var(T t, Consumer<? super T> cons) {
  cons.accept(t);
}

var(2017, i ->
var(Arrays.asList("Joker", "JPoint"), list ->
var(" Hello ", s ->
    System.out.println(s.trim() + ", " + list.get(0) + "-" + i)
)));
```

```
var i = 2018;
var list = Arrays.asList("Joker", "JPoint");
var s = " Hello ";
System.out.println(s.trim() + ", " + list.get(0) + "-" + i);
```

```
var a;
var b = null;
var c = {1, 2, 3};
var[] d = new int[] {4, 5, 6};
var e = 5, f = 6;
var g = String::trim;
var h = () -> { };
```

```
interface Animal {}
interface Cat extends Animal { void meow(); }
interface Dog extends Animal { void bark(); }

void makeSound(Animal animal) {
  var catDog = (Cat & Dog) animal;
  catDog.meow();
  catDog.bark();
}
```



```
void anonymousObject() {
  var object = new Object() {
    int x, y;

    void print() {
      System.out.println(x + ":" + y);
    }
  };
  object.x = 1;
  object.y = 2;
  object.print();
}
```

```
void lockSample() {
  var result = new Object() {
    int count;
    String name;
  };
  runWithLock(() -> {
    result.count = calculateCount();
    result.name = generateName();
  });
  // Используем result.count, result.name уже без блокировки
}
```

```
int var = var();
```

```
int var = var();

variable var = var();
```

```
int var = var();

variable var = var();

variable_with_inferred_type var = var();
```

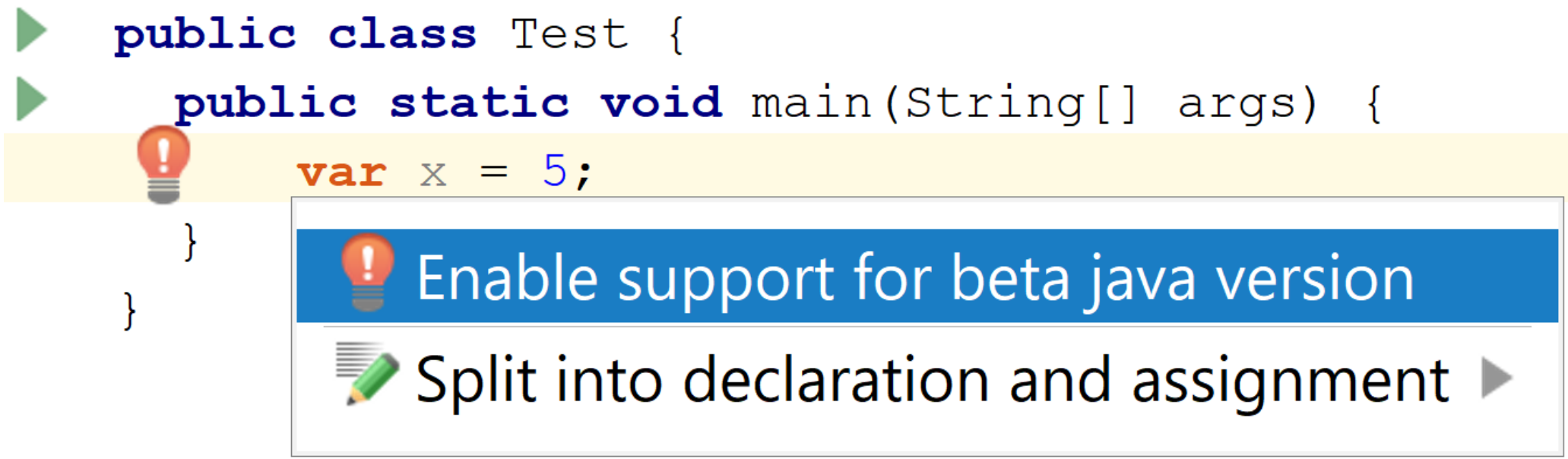
```
int var = var();

variable var = var();

variable_with_inferred_type var = var();

$variable_with_inferred_type$ var = var();
```

```
int var = var();

variable var = var();

variable_with_inferred_type var = var();

$variable_with_inferred_type$ var = var();

$%^& var = var();
```

Restricted keywords

```
var var = var();
```

```
class var {}

class MyClass extends var {}

class MyClass2 extends com.example.var {}

class MyMap<var, val> extends HashMap<var, val> {}
```


public class Test {
public static void main(String[] args) {
var x = 5;
}
}
Enable support for beta java version
Split into declaration and assignment

JEP 301
Enhanced Enums

```
enum Conf {
  JOKER {
    void getWetUnderSpbRain() {
      System.out.println("You're really wet!");
    }
  },
  JBREAK {
    void getFrozenInSiberia() {
      System.out.println("You're frozen!");
    }
  }
}
```

```
enum Conf {
  JOKER {
    void getWetUnderSpbRain() {
      System.out.println("You're really wet!");
    }
  },
  JBREAK {
    void getFrozenInSiberia() {
      System.out.println("You're frozen!");
    }
  }
}

Conf.JOKER.getWetUnderSpbRain();
Conf.JBREAK.getFrozenInSiberia();
```

```
abstract static class Conf {
  private Conf() {}
  static class Joker extends Conf {
    private Joker() {}
    void getWetUnderSpbRain() {
      System.out.println("You're really wet!");
    }
  }
  static class JBreak extends Conf {
    private JBreak() {}
    void getFrozenInSiberia() {
      System.out.println("You're frozen!");
    }
  }
  public static final Joker JOKER = new Joker();
  public static final JBreak JBREAK = new JBreak();
}
```

```
enum Converter<T> {
  INTEGER<Integer> {
    Integer fromString(String s) { return Integer.valueOf(s); }
  },
  DOUBLE<Double> {
    Double fromString(String s) { return Double.valueOf(s); }
  },
  DATE<LocalDate> {
    LocalDate fromString(String s) { return LocalDate.parse(s); }
  };

  abstract T fromString(String s);
}
```

Было: `Converter$1`

Стало: `Converter$$INTEGER`

Было: `Converter$1`

Стало: `Converter$$INTEGER`

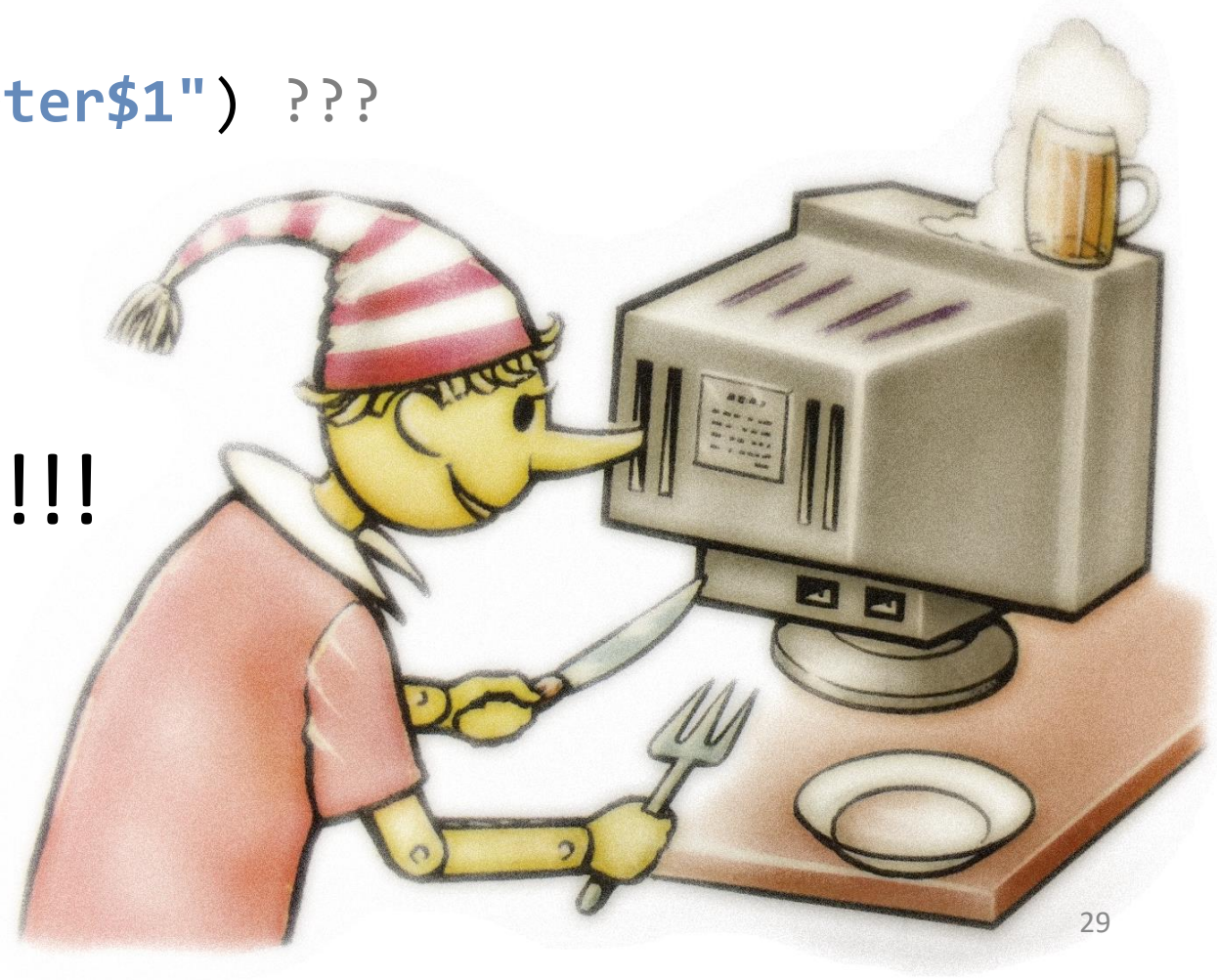
Делаешь `Class.forName("Converter$1")` ???

Было: `Converter$1`

Стало: `Converter$$INTEGER`

Делаешь `Class.forName("Converter$1")` ???

САМ СЕБЕ
ЗЛОБНЫЙ БУРАТИНО!!!



```
EnumSet.noneOf(Converter.class);
```

```
EnumSet.noneOf(Converter.class);

EnumSet<Converter> enumSet = EnumSet.noneOf(Converter.class);
```

Type parameter ‘Converter’ is not within its bound; should extend ‘java.lang.Enum<Converter>’

```
EnumSet.noneOf(Converter.class);

EnumSet<Converter> enumSet = EnumSet.noneOf(Converter.class);
```

Type parameter ‘Converter’ is not within its bound; should extend ‘java.lang.Enum<Converter>’

```
class EnumSet<E extends Enum<E>> extends AbstractSet<E>
```

“F-bound”

```
EnumSet.noneOf(Converter.class);

EnumSet<Converter> enumSet = EnumSet.noneOf(Converter.class);
```

Type parameter ‘Converter’ is not within its bound; should extend ‘java.lang.Enum<Converter>’

```
class EnumSet<E extends Enum<E>> extends AbstractSet<E>
```

“F-bound”

```
Converter extends Enum<Converter>
```

```
EnumSet.noneOf(Converter.class);

EnumSet<Converter<?>> enumSet =
          EnumSet.<Converter<?>> noneOf(Converter.class);
```

Type argument ‘Converter<?>’ is not within bounds of type-variable E

```
class EnumSet<E extends Enum<E>> extends AbstractSet<E>
```

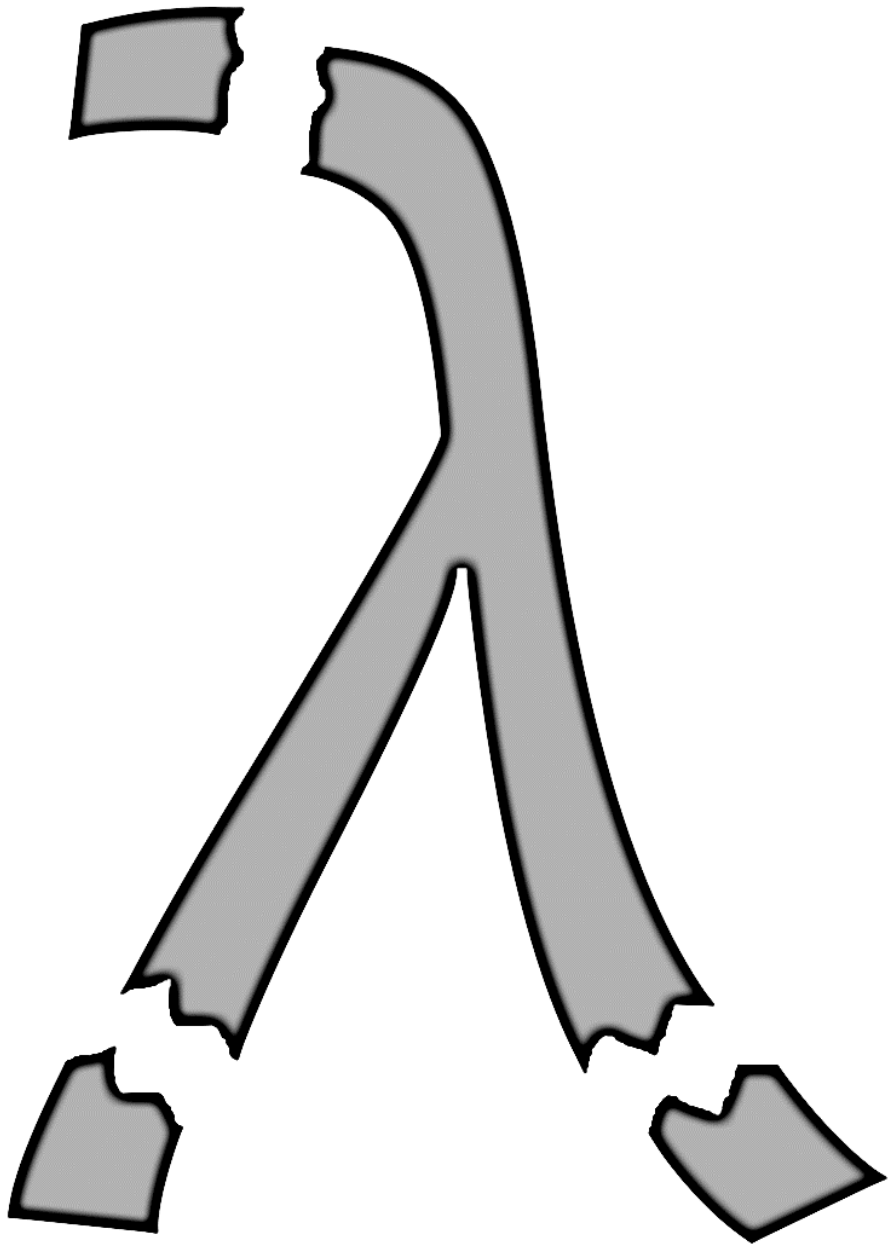
“F-bound”

```
Converter<?> extends Enum<Converter<?>>
```

Amber JEPs

JEP	JIRA Issue	Branch	Name
286	8151454	lvti	Local-Variable Type Inference
~~301~~	~~8170351~~	~~enhanced-enums~~	~~Enhanced Enums~~
302	8170361	lambda-leftovers	Lambda Leftovers
303	8178320	condy-folding	Intrinsics for the LDC and INVOKEDYNAMIC Instructions
309	8177279	condy	Dynamic constants in the JVM
305	8181287	pattern	Pattern Matching

JEP 302
Lambda Leftovers



Lambda Leftovers

1. Подчерк = игнорируемый параметр

```
BiFunction<Integer, String, String> biss =
        (i, _) -> String.valueOf(i);
```

2. Перекрытие переменных параметрами лямбды

```
map.computeIfAbsent(key, key -> key.length());
```

3. Использование возвращаемого типа лямбды для разрешения неоднозначностей

```
void m(Predicate<String> p);
void m(Function<String, String> f);

default void call() {
  m(s -> false);
}
```

Подчерки

```
BinaryOperator<String> op = (_, _) -> "foo";
```

Подчерки

```
BinaryOperator<String> op = (_, _) -> "foo";
```

```
try {
  Files.delete(Paths.get("/etc/passwd"));
} catch (IOException _) { }
```

Подчерки

```
BinaryOperator<String> op = (_, _) -> "foo";
```

```
try {
  Files.delete(Paths.get("/etc/passwd"));
} catch (IOException _) { }
```

```
long count(Iterable<?> iterable) {
  long count = 0;
  for (Object _ : iterable) count++;
  return count;
}
```

Подчерки

```
@Override
void doSomething(int _, String _) {
  // Do nothing
}
```

Подчерки

```
interface SomethingDoer {
  void doSomething(int _, String _);
}
```

Подчерки

```
interface SomethingDoer {
  void doSomething(int _, String _);
}
```

```
native void doSomething(int _, String _);
```

http://mail.openjdk.java.net/pipermail/amber-dev/2017-June/001370.html

Подчерки

```
BinaryOperator<String> op = (_, _) -> "foo";
```

```
try {
  Files.delete(Paths.get("/etc/passwd"));
} catch (IOException _) { }
```

```
long count(Iterable<?> iterable) {
  long count = 0;
  for (Object _ : iterable) count++;
  return count;
}
```

```
@Override
void doSomething(int _, String _) {
  // Do nothing
}
```

Переиспользование имён в параметрах

```
void add(Map<String, List<String>> multiMap,
         String key, String value) {
  multiMap.computeIfAbsent(key, k -> new ArrayList<>())
          .add(value);
}
```

Переиспользование имён в параметрах

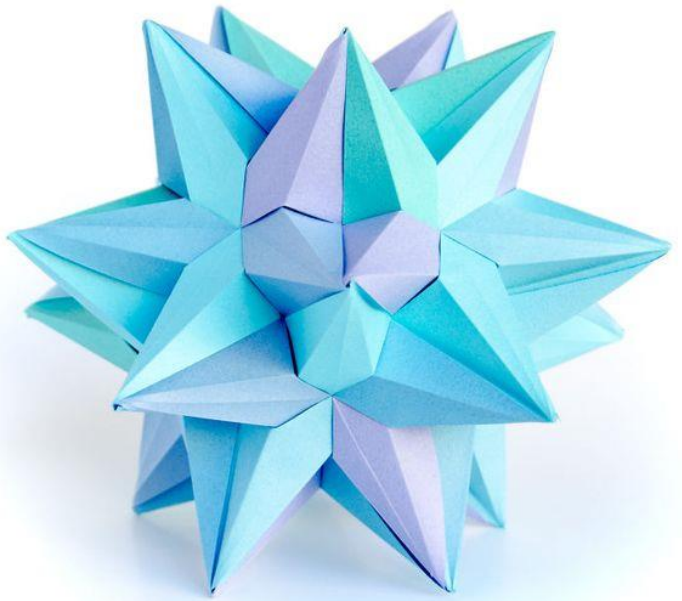
```
void add(Map<String, List<String>> multiMap,
         String key, String value) {
  multiMap.computeIfAbsent(key, k -> new ArrayList<>())
          .add(value);
}
```

```
void add(Map<String, Container> multiMap,
         String key, String value) {
  multiMap.computeIfAbsent(key, _ -> new Container(key, false))
          .accept(value);
}
```

Переиспользование имён в параметрах

```
void add(Map<String, Container> multiMap,
         String key, String value) {
  multiMap.computeIfAbsent(key, key -> new Container(key, false))
          .accept(value);
}
```

```
interface a{
  a a(a a);
}

public static void main(String[] args) {
  a a = a -> a.a(a);
  a.a((a a) -> a);
}
```



JEP 303:
Constant folding (bytecode intrinsics for LDC & invokedynamic)

LDC/LDC_W/LDC2_W

- ✓LDC (0x12), LDC_W (0x13): push constant on the stack:
  - ✓CONSTANT_Integer -> `int`
  - ✓CONSTANT_Float -> `float`
  - ✓CONSTANT_String -> `java.lang.String`
  - ✓CONSTANT_Class -> `java.lang.Class`
  - ✓CONSTANT_MethodType -> `java.lang.invoke.MethodType`
  - ✓CONSTANT_MethodHandle -> `java.lang.invoke.MethodHandle`
- ✓LDC2_W (0x14): push 2-slot constant on the stack:
  - ✓CONSTANT_Long -> `long`
  - ✓CONSTANT_Double -> `double`

https://docs.oracle.com/javase/specs/jvms/se9/html/jvms-6.html#jvms-6.5.ldc

```
public class Ldc {
  public static final MethodHandle SUM;

  static {
    try {
      SUM = MethodHandles.lookup().findStatic(Ldc.class, "sum",
              MethodType.methodType(int.class, int.class, int.class));
    } catch (NoSuchMethodException | IllegalAccessException e) {
      throw new RuntimeException(e);
    }
  }

  private static int sum(int x, int y) { return x + y; }

  public static void main(String[] args) throws Throwable {
    System.out.println((int)SUM.invokeExact(2, 2));
  }
}
```

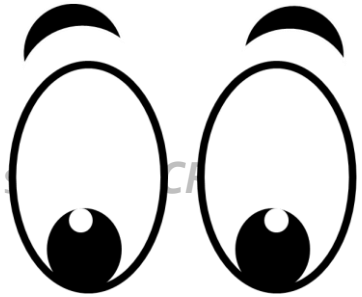
```
stack=9, locals=1, args_size=0
   0: invokestatic  #6   // Method java/lang/invoke/MethodHandles.lookup
   3: ldc           #7   // class test/Ldc
   5: ldc           #8   // String sum
   7: getstatic     #9   // Field java/lang/Integer.TYPE:Ljava/lang/Class;
  10: getstatic     #9   // Field java/lang/Integer.TYPE:Ljava/lang/Class;
  13: iconst_1
  14: anewarray     #10  // class java/lang/Class
  17: dup
  18: iconst_0
  19: getstatic     #9   // Field java/lang/Integer.TYPE:Ljava/lang/Class;
  22: aastore
  23: invokestatic  #11  // Method java/lang/invoke/MethodType.methodType
  26: invokevirtual #12  // Method java/lang/invoke/MethodHandles$Lookup.findStatic
  29: putstatic     #3   // Field SUM:Ljava/lang/invoke/MethodHandle;
  32: goto          45
  35: astore_0
  36: new           #15  // class java/lang/RuntimeException
  39: dup
  40: aload_0
  41: invokespecial #16  // Method java/lang/RuntimeException."<init>"
  44: athrow
  45: return
Exception table:
   from    to  target type
       0    32    35   Class java/lang/NoSuchMethodException
       0    32    35   Class java/lang/IllegalAccessException
```

Зачем мне всё это???

```java
import java.lang.invoke.*;
import static java.lang.invoke.ClassRef.CR_int;

public class Ldc {
  public static final MethodHandle SUM = Intrinsics.ldc(MethodHandleRef.of(
    Kind.STATIC, ClassRef.of("Ldc"), "sum", CR_int, CR_int, CR_int));

  private static int sum(int x, int y) { return x+y; }

  public static void main(String[] args) throws Throwable {
    System.out.println((int)SUM.invokeExact(2, 2));
  }
}
```

```
$ javac -XDdoConstantFold Ldc.java
```

```
stack=1, locals=0, args_size=0
   0: ldc           #6   // MethodHandle REF_invokeStatic Ldc.sum:(II)I
   2: putstatic     #3   // Field SUM:Ljava/lang/invoke/MethodHandle;
   5: return
```



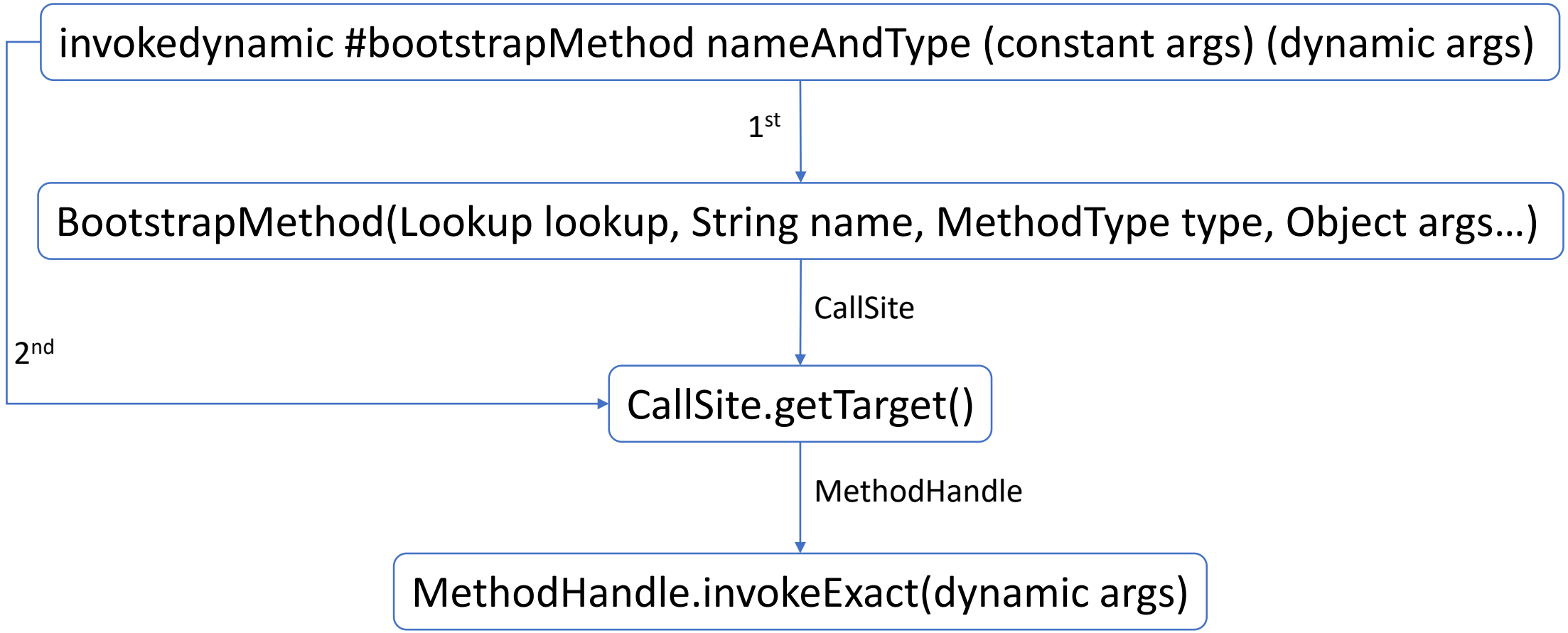
```
import java.lang.invoke.*;
import static java.lang.invoke.ClassRef.CR

public class Ldc {
  public static final MethodHandle SUM = Intrinsics.ldc(MethodHandleRef.of(
    Kind.STATIC, ClassRef.of("Ldc"), "sum", CR_int, CR_int, CR_int));

  private static int sum(int x, int y) { return x+y; }

  public static void main(String[] args) throws Throwable {
    System.out.println((int)SUM.invokeExact(2, 2));
  }
}

$ javac -XDdoConstantFold Ldc.java

stack=1, locals=0, args_size=0
   0: ldc           #6   // MethodHandle REF_invokeStatic Ldc.sum:(II)I
   2: putstatic     #3   // Field SUM:Ljava/lang/invoke/MethodHandle;
   5: return
```

```java
/**
 * Instructs the compiler to generate an {@code ldc} instruction for the given
 * {@code Constable} instance. A compiler error will be issued if it cannot
 * be proved that the argument is a constant
 * @param <T> the type to which this constant pool entry resolves
 * @param constant a constant to be ldc'ed
 * @return the constant wrapped inside the {@code Constable} object
 */
public static <T> T ldc(ConstantRef<T> constant) {
    throw new UnsupportedOperationException("no reflective access");
}
```

```java
/**
 * Return a {@code MethodHandleRef} corresponding to an
 * invocation of a static method
 * @param kind One of: STATIC, VIRTUAL, INTERFACE_VIRTUAL, SPECIAL, CONSTRUCTOR
 * @param clazz the class containing the method
 * @param name the name of the method (ignored if kind=CONSTRUCTOR)
 * @param returnType the return type of the method
 * @param paramTypes the parameter types of the method
 * @return the {@code MethodHandleRef}
 */
@TrackableConstant
public static MethodHandleRef of(Kind kind, ClassRef clazz, String name,
                                 ClassRef returnType, ClassRef... paramTypes) {
    return of(kind, clazz, name, MethodTypeRef.of(returnType, paramTypes));
}
```

```
public class Ldc {
  static final MethodTypeRef RETURNS_INT =
          MethodTypeRef.of(CR_int);
  static final MethodTypeRef INT_BINARY_OP =
          RETURNS_INT.insertParameterTypes(0, CR_int, CR_int);

  public static final MethodHandle SUM = Intrinsics.ldc(
    MethodHandleRef.of(Kind.STATIC, ClassRef.of("Ldc"), "sum", INT_BINARY_OP));

  private static int sum(int x, int y) { return x + y; }

  public static void main(String[] args) throws Throwable {
    System.out.println((int)SUM.invokeExact(2, 2));
  }
}
```

InvokeDynamic (indy)



JUG.RU

https://www.youtube.com/watch?v=DgshYDTpS9I

Invokedynamic: Deep Dive

Vladimir Ivanov, @iwan0www
HotSpot JVM Compiler
Oracle

11.03.2015

InvokeDynamic в Java

✓Компиляция лямбд (Java 8)

```
(a, b) -> a + b
```

✓Компиляция ссылок на методы (Java 8)

```
Integer::sum
```

✓Компиляция конкатенации строк (Java 9) – слава @shipilëv!

```
"Hello " + x + "!"
```

```
static class Counter {
  final String name;
  final AtomicLong count = new AtomicLong();

  public Counter(String name) { this.name = name; }

  void add() { System.out.println(name + "#" + count.incrementAndGet()); }
}


static final Map<String, Counter> ourCounters = new ConcurrentHashMap<>();

Counter getCounter(String name) {
  return ourCounters.computeIfAbsent(name, Counter::new);
}
```

```
public static void main(String[] args) {
    for(int i=0; i<3; i++) {
      getCounter("one").add();
      getCounter("two").add();
    }
    getCounter("one").add();
}
```

```
one#1
two#1
one#2
two#2
one#3
two#3
one#4
```

```
static class Counter {
  final String name;
  final AtomicLong count = new AtomicLong();

  public Counter(String name) { this.name = name; }

  void add() { System.out.println(name + "#" + count.incrementAndGet()); }
}
```

```
static CallSite counter(MethodHandles.Lookup lookup, String name, MethodType type) {
  Counter counter = new Counter(name);
  return new ConstantCallSite(Intrinsics.ldc(
              MethodHandleRef.of(Kind.VIRTUAL, ClassRef.of("Counter"), "add", CR_void)
        ).bindTo(counter));
}
```

```
public class Indy {
  static class Counter {…}
  static CallSite counter(MethodHandles.Lookup lookup, String name, MethodType type) {…}

  static final BootstrapSpecifier BSM = BootstrapSpecifier
        .of(MethodHandleRef.ofIndyBootstrap(ClassRef.of("Indy"), "counter",
                                  ClassRef.of("java.lang.invoke.CallSite")));

  public static void main(String[] args) throws Throwable {
    for(int i=0; i<3; i++) {
      Intrinsics.invokedynamic(BSM, "one");
      Intrinsics.invokedynamic(BSM, "two");
    }
    Intrinsics.invokedynamic(BSM, "one");
  }
}
```

```
public class Indy {
  static class Counter {…}
  static CallSite counter(MethodHandles.Lookup lookup, String name, MethodType type) {…}

  static final BootstrapSpecifier BSM = BootstrapSpecifier
        .of(MethodHandleRef.ofIndyBootstrap(ClassRef.of("Indy"), "counter",
                                  ClassRef.of("java.lang.invoke.CallSite")));

  public static void main(String[] args) throws Throwable {
    for(int i=0; i<3; i++) {
      Intrinsics.invokedynamic(BSM, "one");
      Intrinsics.invokedynamic(BSM, "two");
    }
    Intrinsics.invokedynamic(BSM, "one");
  }
}
```

```
public class Indy {
  static class Counter {…}
  static CallSite counter(MethodHandles.Lookup lookup, String name, MethodType type) {…}
  static final BootstrapSpecifier BSM = …;

  public static void main(String[] args) throws Throwable {
    for(int i=0; i<3; i++) {
      Intrinsics.invokedynamic(BSM, "one");
      Intrinsics.invokedynamic(BSM, "two");
    }
    Intrinsics.invokedynamic(BSM, "one");
  }
}
```

```
one#1
two#1
one#2
two#2
one#3
two#3
one#1
```

JEP 309:
Constant dynamic

"One person's constant is another person's variable"
Susan Gerhart

LDC/LDC_W/LDC2_W

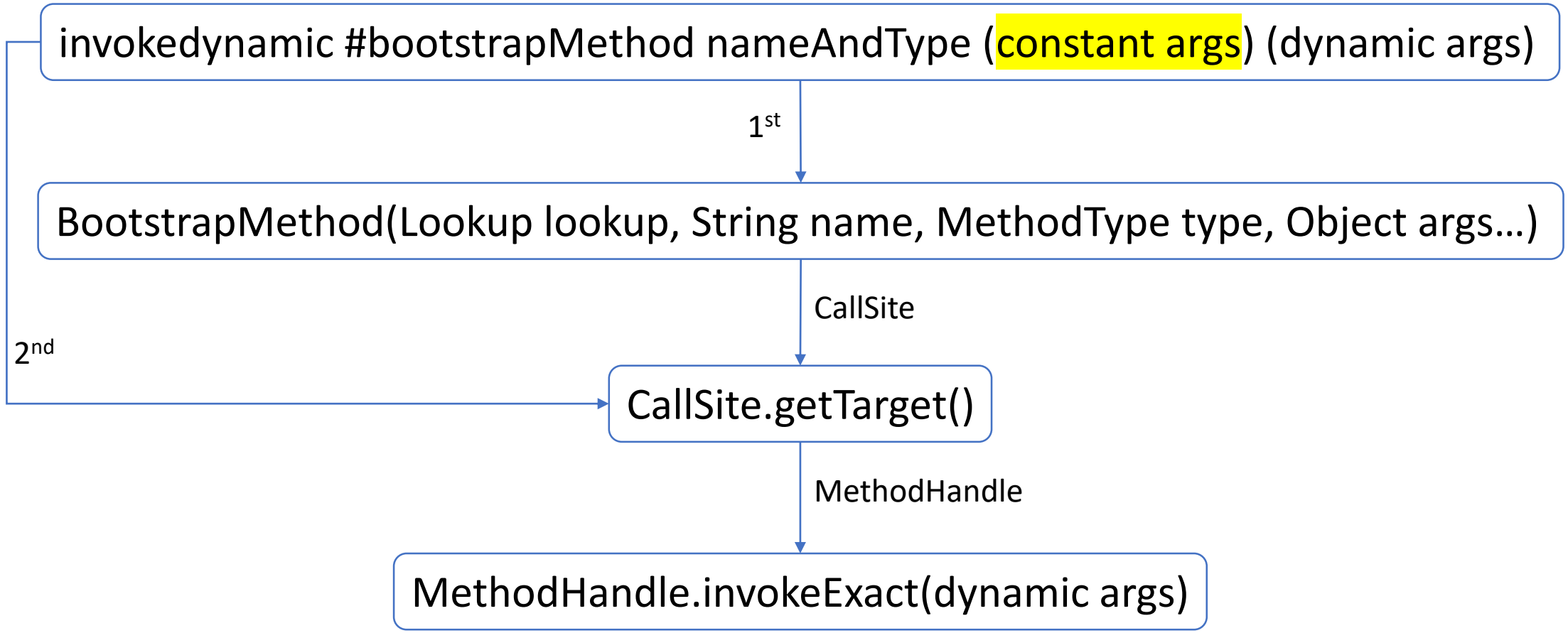
- ✓LDC (0x12), LDC_W (0x13): push constant on the stack:
  - ✓CONSTANT_Integer -> `int`
  - ✓CONSTANT_Float -> `float`
  - ✓CONSTANT_String -> `java.lang.String`
  - ✓CONSTANT_Class -> `java.lang.Class`
  - ✓CONSTANT_MethodType -> `java.lang.invoke.MethodType`
  - ✓CONSTANT_MethodHandle -> `java.lang.invoke.MethodHandle`
- ✓LDC2_W (0x14): push 2-slot constant on the stack:
  - ✓CONSTANT_Long -> `long`
  - ✓CONSTANT_Double -> `double`

https://docs.oracle.com/javase/specs/jvms/se9/html/jvms-6.html#jvms-6.5.ldc

Constant pool: чего нельзя

- ✓null → `aconstnull`
- ✓Enum constant → `getstatic EnumType.CONSTANT`
- ✓int.class → `getstatic Integer.TYPE`
- ✓Чистая функция → `invokedynamic …`
- ✓Список других констант (List.of(xyz))
- ✓Мало ли что ещё!

InvokeDynamic (indy)



Constant dynamic (condy)

- ✓Новый тип констант в пуле (CONSTANT_Dynamic):
  - ✓Имя
  - ✓Тип
  - ✓Bootstrap method
  - ✓Константные аргументы
- ✓Bootstrap вызывается один раз

https://github.com/forax/cplisp

By **Rémi Forax**

cplisp

A Lisp that is able to compile part of itself into the constant pool of a Java .class

It uses the new constant dynamic (condy) constant pool constant, so if you want to run a .class, you need to get the source of the project amber

```
hg clone http://hg.openjdk.java.net/amber/amber
```

then update to move to the condy branch

```
hg update condy
```

then compile the whole jdk

```
configure
make images
```

```
public static void main(java.lang.String[]);
descriptor: ([Ljava/lang/String;)V
flags: (0x0009) ACC_PUBLIC, ACC_STATIC
Code: stack=1, locals=1, args_size=1
   0: ldc #26 // ConstantDynamic #1:eval:Ljava/lang/Object;
   2: pop
   3: return
```

(eval (print hello cplisp))

BootstrapMethods:
0: #21 REF_invokeStatic fr/umlv/cplisp/CpLisp.bsm
Method arguments:
  #10 print
  #12 hello
  #14 cplisp

1: 0: #21 REF_invokeStatic fr/umlv/cplisp/CpLisp.bsm
Method arguments:
  #8 eval
  #24 #0:print:Ljava/lang/Object;

```
import java.lang.invoke.*;

public class Condy {
  public static void main(String[] args) {
    System.out.println((Object)Intrinsics.ldc(ConstantRef.ofNull()));
  }
}
```

```
public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: (0x0009) ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=1, args_size=1
         0: getstatic     #2      // Field java/lang/System.out:Ljava/io/PrintStream;
         3: ldc           #3      // ConstantDynamic #0:_:Ljava/lang/Object;
         5: invokevirtual #4      // Method java/io/PrintStream.println:(Ljava/lang/Object;)V
         8: return

BootstrapMethods:
  0: #19 REF_invokeStatic java/lang/invoke/ConstantBootstraps.defaultValue
    Method arguments:
```

```
public class Condy {
  static Pattern patternBS(MethodHandles.Lookup lookup, String name, Class<?> type,
                           String pattern) {
    return Pattern.compile(pattern);
  }

  static final MethodHandleRef PATTERN_CONDY = MethodHandleRef.ofCondyBootstrap(
      ClassRef.of("Condy"), "patternBS", ClassRef.of("java.util.regex.Pattern"), CR_String);

  public static void main(String[] args) {
    for(String arg : args) {
      Pattern p = Intrinsics.ldc(DynamicConstantRef.of(
                  BootstrapSpecifier.of(PATTERN_CONDY, "[,;]")));
      System.out.println(Arrays.toString(p.split(arg)));
    }
  }
}
```

```
public class Condy {
  static Pattern patternBS(MethodHandles.Lookup lookup, String name, Class<?> type,
                                                        String pattern) {
    return Pattern.compile(pattern);
  }

  static final MethodHandleRef PATTERN_CONDY = MethodHandleRef.ofCondyBootstrap(
      ClassRef.of("Condy"), "patternBS", ClassRef.of("java.util.regex.Pattern"), CR_String);

  public static void main(String[] args) {
    for(String arg : args) {
      Pattern p = Intrinsics.ldc(DynamicConstantRef.of(
                  BootstrapSpecifier.of(PATTERN_CONDY, "[,;]")));
      System.out.println(Arrays.toString(p.split(arg)));
    }
  }
}
```

```
public class Condy {
  static Pattern patternBS(MethodHandles.Lookup lookup, String name, Class<?> type,
                                                        String pattern) {
    return Pattern.compile(pattern);
  }

  static final MethodHandleRef PATTERN_CONDY = MethodHandleRef.ofCondyBootstrap(
      ClassRef.of("Condy"), "patternBS", ClassRef.of("java.util.regex.Pattern"), CR_String);

  public static void main(String[] args) {
    for(String arg : args) {
      Pattern p = Intrinsics.ldc(DynamicConstantRef.of(
                  BootstrapSpecifier.of(PATTERN_CONDY, "[,;]")));
      System.out.println(Arrays.toString(p.split(arg)));
    }
  }
}
```

```
public class Condy {
  static Pattern patternBS(MethodHandles.Lookup lookup, String name, Class<?> type,
                                                        String pattern) {
    return Pattern.compile(pattern);
  }

  static final MethodHandleRef PATTERN_CONDY = MethodHandleRef.ofCondyBootstrap(
      ClassRef.of("Condy"), "patternBS", ClassRef.of("java.util.regex.Pattern"), CR_String);

  public static void main(String[] args) {
    for(String arg : args) {
      Pattern p = Intrinsics.ldc(DynamicConstantRef.of(
                  BootstrapSpecifier.of(PATTERN_CONDY, "[,;]")));
      System.out.println(Arrays.toString(p.split(arg)));
    }
    Pattern p2 = Intrinsics.ldc(DynamicConstantRef.of(
                 BootstrapSpecifier.of(PATTERN_CONDY, "[,;]")));
    System.out.println(Arrays.toString(p2.split("Joker;JPoint,JBreak")));
  }
}
```

```
public class Condy {
  static final MethodHandleRef LAMBDA_CONDY = MethodHandleRef.ofCondyBootstrap(
          ClassRef.of("java.lang.invoke.LambdaMetafactory"), "metafactory",
          CR_Object, CR_MethodType, CR_MethodHandle, CR_MethodType);

  static final MethodTypeRef INT_BINARY_FN = MethodTypeRef.of(CR_int, CR_int, CR_int);

  static final MethodHandleRef INT_SUM = MethodHandleRef.of(Kind.STATIC,
          CR_Integer, "sum", INT_BINARY_FN);

  public static void main(String[] args) {
    IntBinaryOperator sum = Intrinsics.ldc(DynamicConstantRef.of(BootstrapSpecifier.of(
       LAMBDA_CONDY, INT_BINARY_FN, INT_SUM, INT_BINARY_FN),
       "applyAsInt", ClassRef.of("java.util.function.IntBinaryOperator")));
    System.out.println(sum.applyAsInt(2, 2));
  }
}
```

```
public class Condy {
  static final MethodHandleRef LAMBDA_CONDY = MethodHandleRef.ofCondyBootstrap(
          ClassRef.of("java.lang.invoke.LambdaMetafactory"), "metafactory",
          CR_Object, CR_MethodType, CR_MethodHandle, CR_MethodType);

  static final MethodTypeRef INT_BINARY_FN = MethodTypeRef.of(CR_int, CR_int, CR_int);

  static final MethodHandleRef INT_SUM = MethodHandleRef.of(Kind.STATIC,
          CR_Integer, "sum", INT_BINARY_FN);

  public static void main(String[] args) {
    IntBinaryOperator sum = Intrinsics.ldc(DynamicConstantRef.of(BootstrapSpecifier.of(
       LAMBDA_CONDY, INT_BINARY_FN, INT_SUM, INT_BINARY_FN),
       "applyAsInt", ClassRef.of("java.util.function.IntBinaryOperator")));
    System.out.println(sum.applyAsInt(2, 2));
  }
}
```

```
public class Condy {
  static final MethodHandleRef LAMBDA_CONDY = MethodHandleRef.ofCondyBootstrap(
          ClassRef.of("java.lang.invoke.LambdaMetafactory"), "metafactory",
          CR_Object, CR_MethodType, CR_MethodHandle, CR_MethodType);

  static final MethodTypeRef INT_BINARY_FN = MethodTypeRef.of(CR_int, CR_int, CR_int);

  static final MethodHandleRef INT_SUM = MethodHandleRef.of(Kind.STATIC,
          CR_Integer, "sum", INT_BINARY_FN);

  public static void main(String[] args) {
    IntBinaryOperator sum = Intrinsics.ldc(DynamicConstantRef.of(BootstrapSpecifier.of(
       LAMBDA_CONDY, INT_BINARY_FN, INT_SUM, INT_BINARY_FN),
       "applyAsInt", ClassRef.of("java.util.function.IntBinaryOperator")));
    System.out.println(sum.applyAsInt(2, 2));
  }
}
```

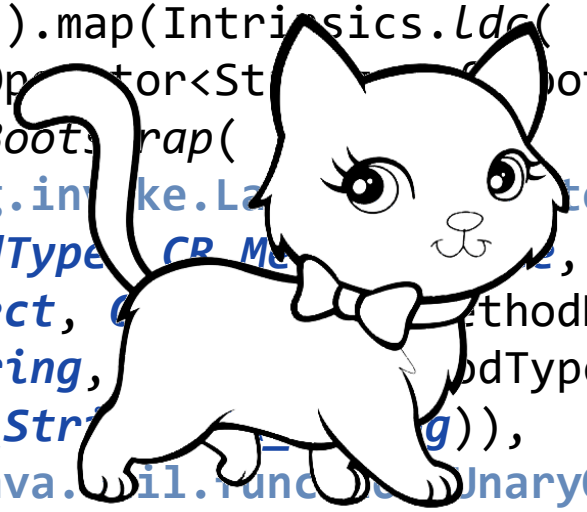
```
public class Condy {
  public static void main(String[] args) {
    IntBinaryOperator sum = Integer::sum;
    System.out.println(sum.applyAsInt(2, 2));
    IntBinaryOperator sum2 = Integer::sum;
    System.out.println(sum2.applyAsInt(2, 2));
  }
}
```

«Презентация без JMH-бенчмарка – что брачная ночь без невесты»

```java
@Benchmark
public String processStringNormal() {
  return Optional.of("  foo  ").map(String::trim).orElse("");
}
```

```
@Benchmark
public String processStringNormal() {
  return Optional.of("  foo  ").map(String::trim).orElse("");
}

@Benchmark
public String processStringCondy() {
  return Optional.of("  foo  ").map(Intrinsics.ldc(
    DynamicConstantRef.<UnaryOperator<String>>of(BootstrapSpecifier.of(
      MethodHandleRef.ofCondyBootstrap(
        ClassRef.of("java.lang.invoke.LambdaMetafactory"), "metafactory",
          CR_Object, CR_MethodType, CR_MethodHandle, CR_MethodType),
      MethodTypeRef.of(CR_Object, CR_Object), MethodHandleRef.of(
          Kind.VIRTUAL, CR_String, "trim", MethodTypeRef.of(CR_String)),
          MethodTypeRef.of(CR_String, CR_String)),
      "apply", ClassRef.of("java.util.function.UnaryOperator")))).orElse("");
}
```

```
@Benchmark
public String processStringNormal() {
  return Optional.of("  foo  ").map(String::trim).orElse("");
}

@Benchmark
public String processStringCondy() {
  return Optional.of("  foo  ").map(Intrinsics.ldc(
    DynamicConstantRef.<UnaryOperator<St      ootstrapSpecifier.of(
      MethodHandleRef.ofCondyBoots    rap(
        ClassRef.of("java.lang.in    ke.La      tory"), "metafactory",
          CR_Object, CR_MethodType    CR_M      , CR_MethodType),
      MethodTypeRef.of(CR_Object,     thodHandleRef.of(
          Kind.VIRTUAL, CR_String,      odTypeRef.of(CR_String)),
          MethodTypeRef.of(CR_Str       g)),
      "apply", ClassRef.of("java.    il.Func      UnaryOperator")))).orElse("");
}
```



```
public java.lang.String processStringNormal();
  Code:
    0: ldc           #2       // String "  foo  "
    2: invokestatic  #3       // Method java/util/Optional.of
    5: invokedynamic #4,  0   // InvokeDynamic #0:apply:()Ljava/util/function/Function;
   10: invokevirtual #5       // Method java/util/Optional.map
   13: ldc           #6       // String ""
   15: invokevirtual #7       // Method java/util/Optional.orElse
   18: checkcast     #8       // class java/lang/String
   21: areturn

public java.lang.String processStringCondy();
  Code:
    0: ldc           #2       // String "  foo  "
    2: invokestatic  #3       // Method java/util/Optional.of
    5: ldc           #9       // Dynamic #1:apply:Ljava/util/function/UnaryOperator;
    7: invokevirtual #5       // Method java/util/Optional.map
   10: ldc           #6       // String ""
   12: invokevirtual #7       // Method java/util/Optional.orElse
   15: checkcast     #8       // class java/lang/String
   18: areturn
```

```
# JMH version: 1.19
# VM version: JDK 10-internal, VM 10-internal+0-adhoc.lan.amber
# VM invoker: /mnt/storage/amber/build/linux-x86_64-normal-server-release/jdk/bin/java
# VM options: <none>
Benchmark                  Mode  Cnt   Score   Error  Units
Condy.processStringCondy   avgt   10  28.114 ± 1.657  ns/op
Condy.processStringNormal  avgt   10  28.065 ± 1.507  ns/op
```

```
# JMH version: 1.19
# VM version: JDK 10-internal, VM 10-internal+0-adhoc.lan.amber
# VM invoker: /mnt/storage/amber/build/linux-x86_64-normal-server-release/jdk/bin/java
# VM options: -Xint
Benchmark                  Mode  Cnt     Score     Error  Units
Condy.processStringCondy   avgt  100  3675.217 ±  86.479  ns/op
Condy.processStringNormal  avgt  100  3897.361 ± 102.085  ns/op
```

```
# JMH version: 1.19
# VM version: JDK 10-internal, VM 10-internal+0-adhoc.lan.amber
# VM invoker: /mnt/storage/amber/build/linux-x86_64-normal-server-release/jdk/bin/java
# VM options: <none>
Benchmark                    Mode  Cnt   Score   Error  Units
Condy.processStringCondy     avgt   10  28.114 ± 1.657  ns/op
Condy.processStringNormal    avgt   10  28.065 ± 1.507  ns/op
```

```
# JMH version: 1.19
# VM version: JDK 10-internal, VM 10-internal+0-adhoc.lan.amber
# VM invoker: /mnt/storage/amber/build/linux-x86_64-normal-server-release/jdk/bin/java
# VM options: -Xint
Benchmark                    Mode  Cnt     Score     Error  Units
Condy.processStringCondy     avgt  100  3675.217 ±  86.479  ns/op
Condy.processStringNormal    avgt  100  3897.361 ± 102.085  ns/op
```

«А пять старушек – уже пять рублей!»

JEP 305:
Pattern Matching

Pattern (образец)

- ✓`null`
- ✓Константа (`1, 5.0, "foo"`)
- ✓Тип+новая переменная (`Integer x`)
- ✓Тип+(образцы для компонент) (`Map.Entry(1, String s)`)

Использование образцов:

- ✓ `<expression> __matches <pattern>`
- ✓
```
switch(<expression>) {
  case <pattern1>: …; break;
  case <pattern2>: …; break;
  default: …; break;
}
```
- ✓
```
return switch(<expression>) {
  case <pattern1> -> <result1>
  case <pattern2> -> <result2>
  default -> <default_result>
};
```

__matches

```
if (x __matches 1) {
  System.out.println("x is one!");
}
```

__matches

```
void test(Object x) {
  if (x __matches 1) {
    System.out.println("x is one!");
  }
}
```

__matches

```
void test(Object x) {
  if (x __matches 1) {
    System.out.println("x is one!");
  }
}
```

```
void test(Object x) {
  if (x instanceof Integer && ((Integer) x) == 1) {
    System.out.println("x is one!");
  }
}
```

__matches

```
void test(Object x) {
  if (x __matches 1) {
    System.out.println("x is one!");
  }
}
```

```
void test(Object x) {
  if (Integer.valueOf(1).equals(x)) {
    System.out.println("x is one!");
  }
}
```

__matches

```
final int __matches = 1;
if (__matches __matches __matches) {
  System.out.println("Matches!");
}
```

__matches

```
final boolean __matches = false;
if ((__matches __matches __matches) __matches __matches) {
  System.out.println("Matches!");
}
```

__matches

```
void test(Object x) {
  if (x __matches String s) {
    System.out.println(s.trim());
  }
}
```

```
void test(Object x) {
  if (x instanceof String) {
    String s = (String) x;
    System.out.println(s.trim());
  }
}
```

__matches

```
void test(Object x) {
  if (!(x __matches String s)) {
    System.out.println("not string");
    return;
  } else {
    System.out.println(s.trim());
  }
}
```

__matches

```
void test(Object x) {
  if (!(x __matches String s)) {
    System.out.println("not string");
    return;
  }
  System.out.println(s.trim());
}
```

__matches

```
boolean isNonEmptyString(Object obj) {
  return obj __matches String s && !s.isEmpty();
}
```

__matches

```
int firstInt(Object o1, Object o2) {
  return (o1 __matches Integer i || o2 __matches Integer i)
          ? i : 0;
}
```

__matches

```
void intOrLong(Object obj) {
  if (obj __matches Integer n || obj __matches Long n) {
    System.out.println(n.longValue()); // n is Number?
  }
}
```

switch

```
static void testSwitch(Object obj) {
  switch(obj) {
    case 1:
       System.out.println("One"); break;
    case 2:
       System.out.println("Two"); break;
    case Integer i:
       System.out.println("Other integer: " + i); break;
    case Number n:
       System.out.println("Non-integer number: " + n); break;
    default:
       System.out.println("Other"); break;
  }
}
```

switch

```
static void testSwitch(Object obj) {
  switch(obj) {
    case 1:
       System.out.println("One"); break;
    case 2:
       System.out.println("Two"); break;
    case Integer i:
       System.out.println("Other integer: " + i); break;
    case Number n:
       System.out.println("Non-integer number: " + n); break;
    case null:
    default:
       System.out.println("Other"); break;
  }
}
```

Хочется ещё?

- ✓Pattern Matching with Brian Goetz @BrianGoetz
  https://youtu.be/n3_8YcYKScw
  http://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html
  http://cr.openjdk.java.net/~briangoetz/amber/pattern-match-translation.html
  http://cr.openjdk.java.net/~briangoetz/amber/switch-rehab.html
- ✓Constants in Amber with Vicente Romero and Paul Sandoz
  https://youtu.be/XbEC0jW1fDY
  http://cr.openjdk.java.net/~briangoetz/amber/constables.html
- ✓Mailing lists
  http://mail.openjdk.java.net/mailman/listinfo/amber-spec-observers
  http://mail.openjdk.java.net/mailman/listinfo/amber-dev

Спасибо
за внимание

—

https://twitter.com/tagir_valeev
https://habrahabr.ru/users/lany
https://github.com/amaembo
tagir.valeev@jetbrains.com