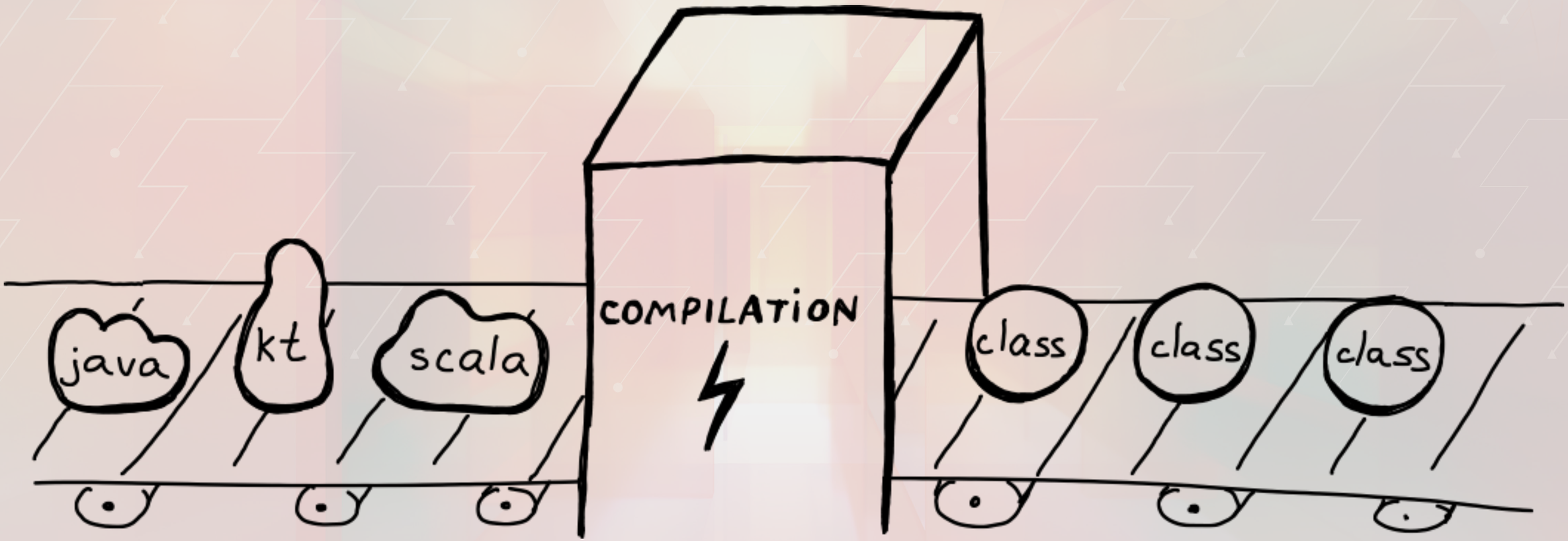
The Hidden Dynamic Life of Java

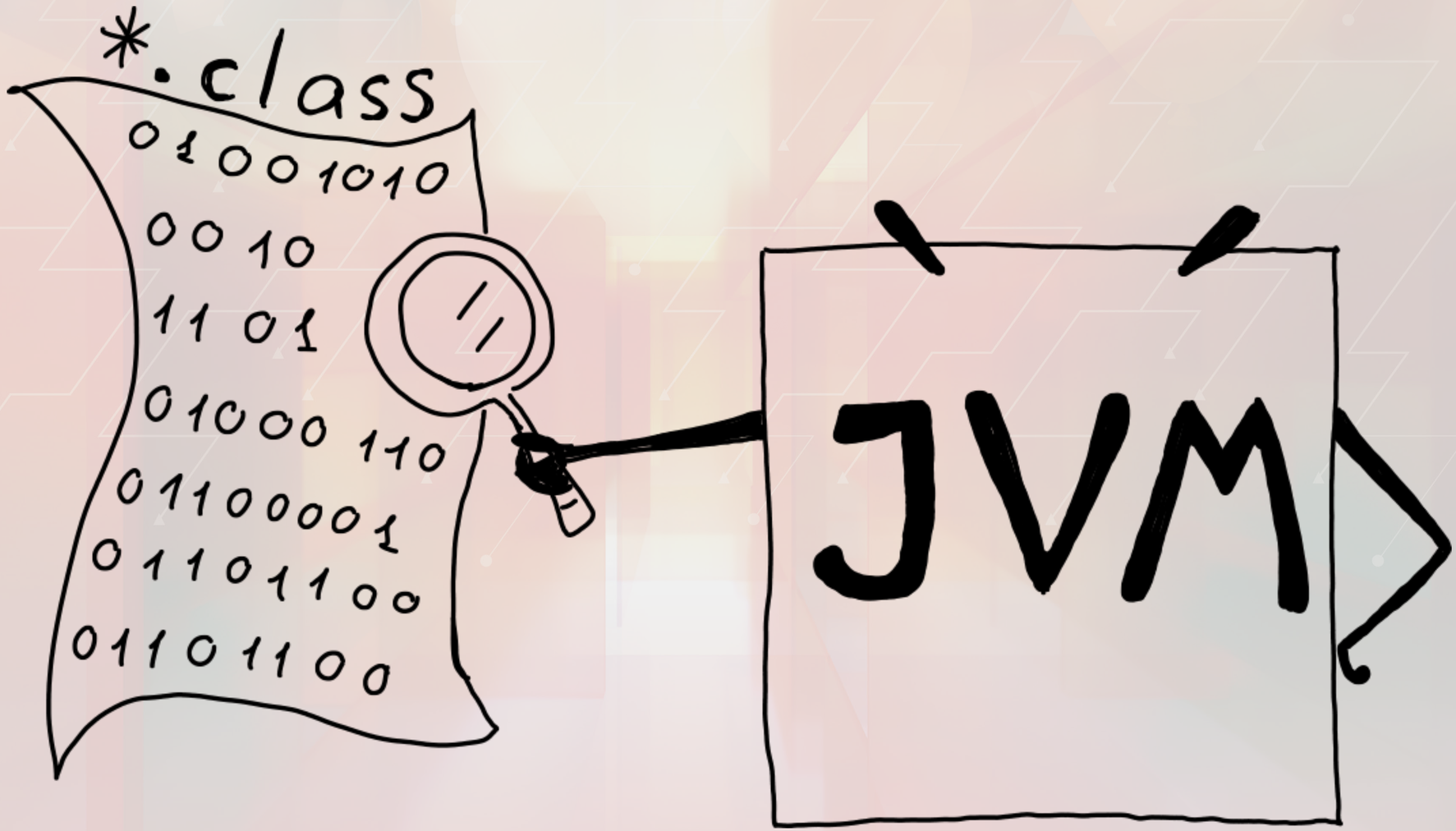
Nataliia Dziubenko

#jfall.

Convenience of the standard bytecode format

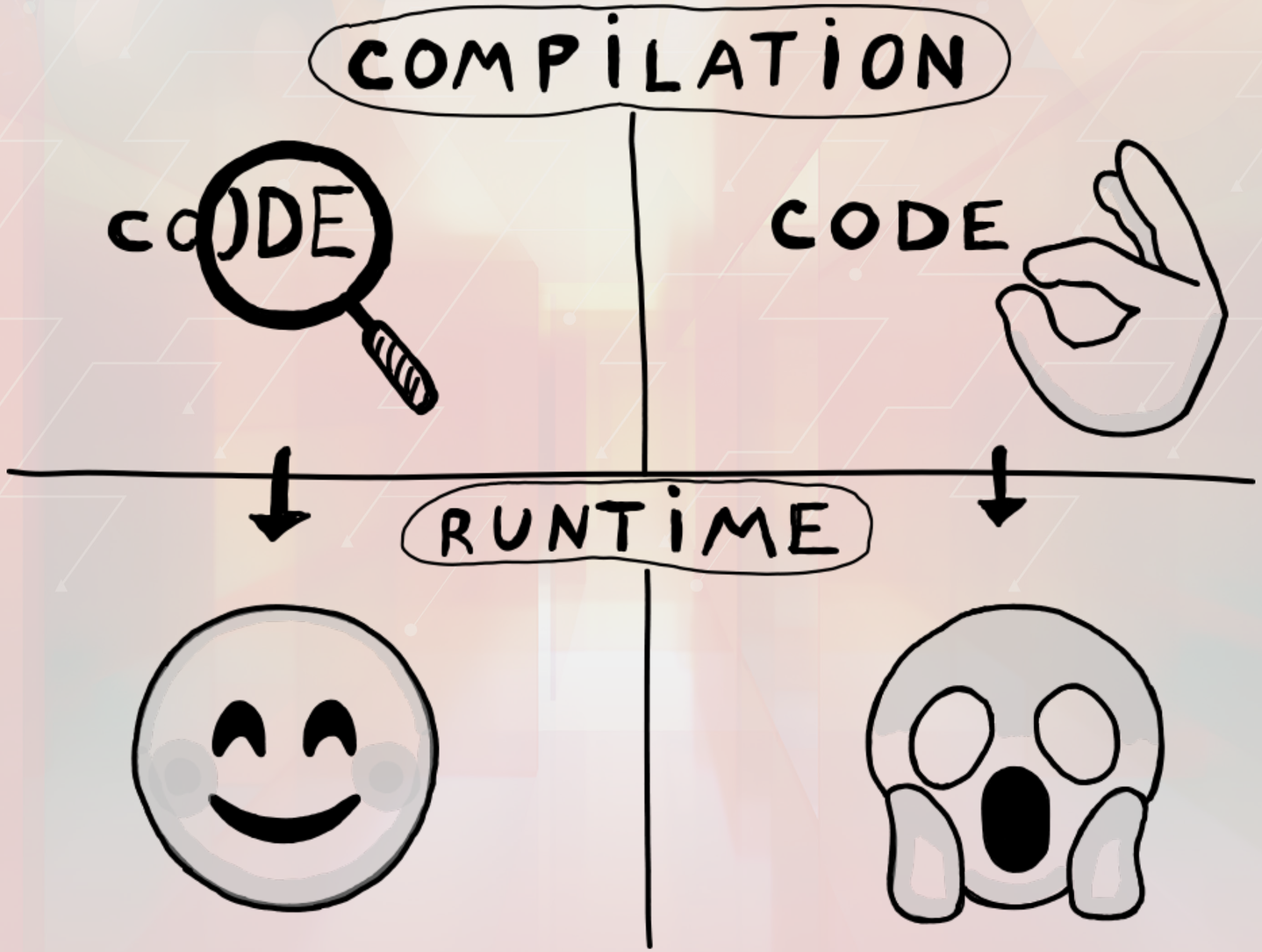


Convenience of the standard bytecode format



It was all for Java!

Statically vs dynamically typed languages



Dynamically typed languages on JVM

Jython

For example

```
def absolute(num):
    return num.__abs__()



print(absolute(-1))
print(absolute("a"))
```

Possible workaround

- LANGUAGE-SPECIFIC TYPES

```
public static SuperType abs(SuperType num) {
    return num.abs();
}

public static void main(String[] args) {
    SuperType result1 = abs(SuperType.of(-2));
    SuperType result2 = abs(SuperType.of("a"));
}
```

Other workarounds

- REFLECTION
- EXTRA INTERPRETER

The new way

How is indy different?

INVOKEVIRTUAL

```
0: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
3: ldc           #13                 // String hello J-Fall!
5: invokevirtual #15                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

INVOKESTATIC

```
0: bipush        115
2: invokestatic  #7                  // Method java/lang/String.valueOf:(I)Ljava/lang/String;
```

INVOKEDYNAMIC

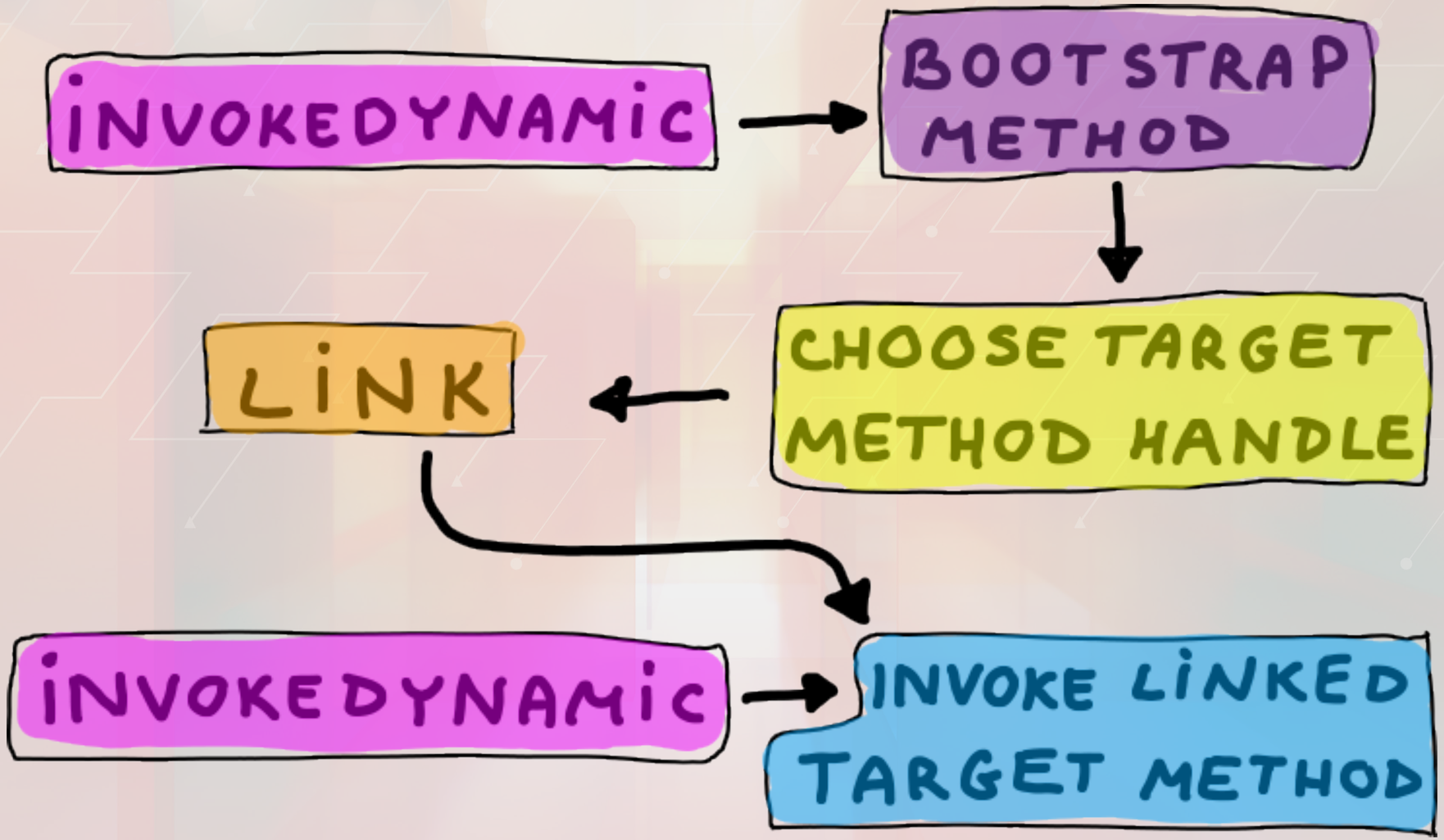
```
3: iload_1
4: invokedynamic #7,  0              // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
```

Bootstrap method

```
3: iload_1
4: invokedynamic #7,  0                  // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #31 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #29 It\'s \u0001 years!
```

Bootstrap method




HMM COOL, BUT WE ARE JAVA DEVS
DYNAMIC LANGUAGES ARE SCARY

String concatenation

```
String a = "a";
String b = "b";
String test = a + b;
```

```
         6: aload_1
         7: aload_2
         8: invokedynamic #11,  0              // InvokeDynamic #0:makeConcatWithConstants:
(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #43 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #49 \u0001\u0001
```

String concatenation

```
    // Fast-path binary concatenations
    if (elements.size() == 2) {
        // Two arguments
        String s0 = elements.get(0);
        String s1 = elements.get(1);

        if (mt.parameterCount() == 2 &&
                !mt.parameterType( num: 0).isPrimitive() &&
                !mt.parameterType( num: 1).isPrimitive() &&
                s0 == null &&
                s1 == null) {
            return simpleConcat();
```

```
MethodHandle simpleConcat = JLA.stringConcatHelper( name: "simpleConcat",
        methodType(String.class, Object.class, Object.class));
```

```
static String simpleConcat(Object first, Object second)
```

Pattern matching for switch


```
public static void main(String[] args) {
    test( obj: 115);
    test( obj: "a");
}

2 usages
public static void test(Object obj) {
    switch (obj) {
        case String s                    -> System.out.println("string: " + s);
        case Integer num                 -> System.out.println("int to long: " + num.longValue());
        default                          -> System.out.println("default");
    }
}
```

Pattern matching for switch

```
 9: aload_1
10: iload_2
11: invokedynamic #27,  0              // InvokeDynamic #0:typeSwitch:(Ljava/lang/Object;I)I
16: lookupswitch  { // 2
               0: 44
               1: 64
         default: 89
    }
```

```
BootstrapMethods:
  0: #81 REF_invokeStatic java/lang/runtime/SwitchBootstraps.typeSwitch:(Ljava/lang/invoke/MethodHandles$Lookup;
Ljava/lang/String;Ljava/lang/invoke/MethodType;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #31 java/lang/String
      #8 java/lang/Integer
```

Lambdas

```
Stream.of( ...values: "a", "b").forEach(System.out::println);
```

```
       25: invokedynamic #31,  0             // InvokeDynamic #0:accept:(Ljava/io/PrintStream;)
Ljava/util/function/Consumer;
       30: invokeinterface #35,  2           // InterfaceMethod java/util/stream/Stream.forEach
:(Ljava/util/function/Consumer;)V
```

```
BootstrapMethods:
  0: #63 REF_invokeStatic java/lang/invoke/LambdaMetafactory.metafactory:(Ljava/lang/invoke/MethodHandles$Look
up;Ljava/lang/String;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodHandle;
Ljava/lang/invoke/MethodType;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #53 (Ljava/lang/Object;)V
      #55 REF_invokeVirtual java/io/PrintStream.println:(Ljava/lang/String;)V
      #62 (Ljava/lang/String;)V
```

Why?

COMPLEXITY

SPACE

ABSTRACT AWAY DECISION MAKING

IMPROVED IMPLEMENTATION

NO NEED TO RECOMPILE

WHY NOT USE NORMAL METHOD CALLS ?

Conclusions

- PREDICTABILITY
- PERFORMANCE

btw, what about AOT?

- javac USAGES SUPPORTED

btw, what about AOT?

- javac USAGES SUPPORTED
- kotlin compiler DOES THE SAME

btw, what about AOT?

- javac USAGES SUPPORTED
- kotlin compiler DOES THE SAME
- BE CAREFUL AS A COMPILER CREATOR

btw, what about AOT?

- javac USAGES SUPPORTED
- kotlin compiler DOES THE SAME
- BE CAREFUL AS A COMPILER CREATOR
- DON'T BE CREATIVE WITH BYTECODE MODIFICATION

Some fun reading

ORACLE

Medium

𝕏@worth_exploring

Thanks for your attention

Please rate my session in the J-Fall app

#jfall.