Во что компилируется Java

Баринов Юрий

План

#1 Немножко про байт-код

#2 invokevirtual

#3 invokedynamic

#4 Лямбды

#5 Бенчи

Байт-код

– Это что ассемблер в жаве?
– Я думал это для сишников(

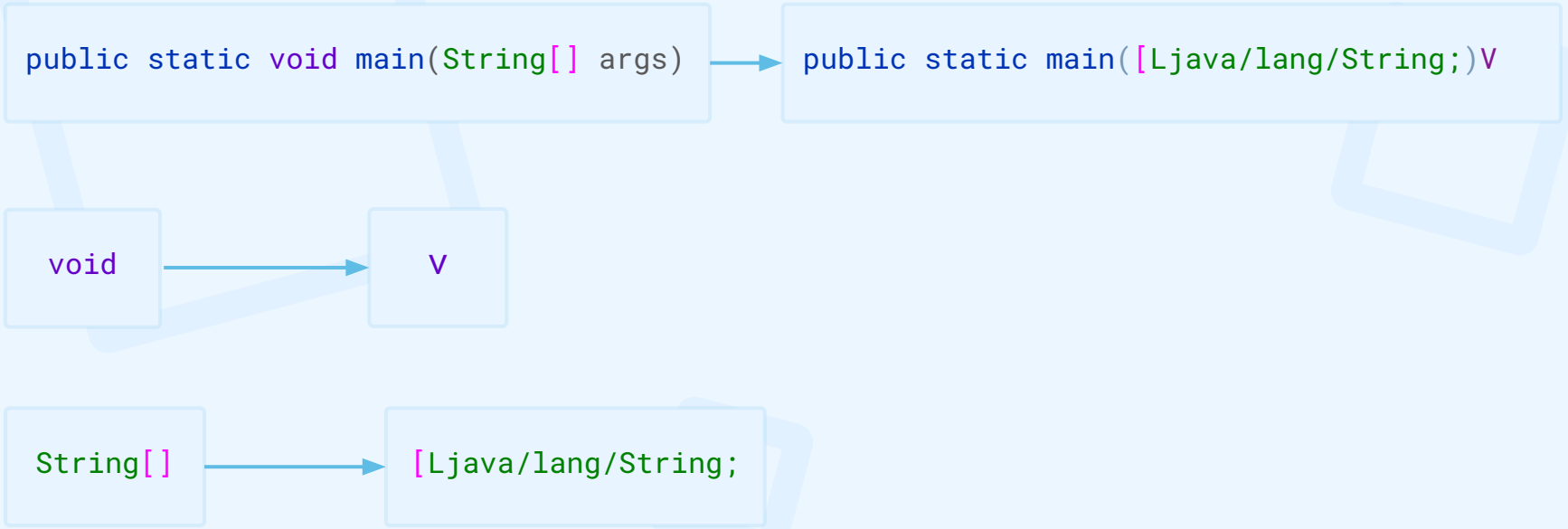
Метод

- Флаги доступа
- Имя
- Дескриптор
- Атрибуты

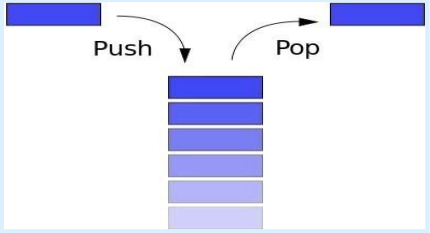
```
public static void main(String[] args) {
 System.out.println("Hello, Joker!");
}
```

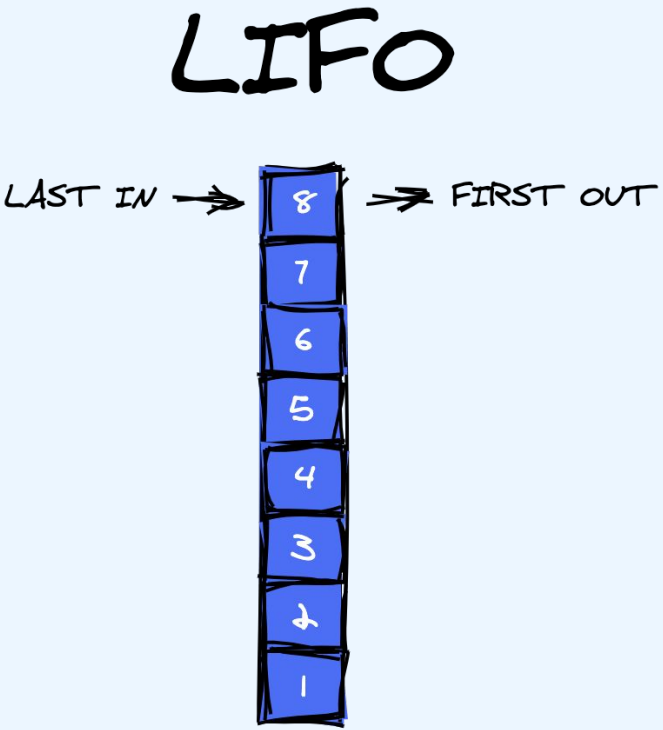
```
public static main([Ljava/lang/String;)V
L0
 LINENUMBER 21 L0
 GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
 LDC "Hello, Joker!"
 INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
L1
 LINENUMBER 22 L1
 RETURN
L2
 LOCALVARIABLE args [Ljava/lang/String; L0 L2 0
 MAXSTACK = 2
 MAXLOCALS = 1
```

Дескриптор

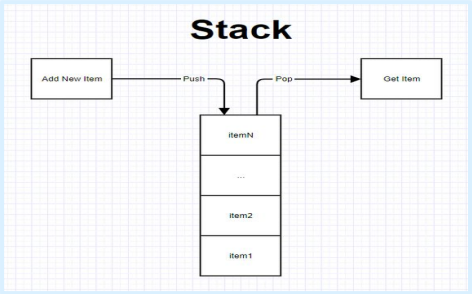

public static void main(String[] args)
public static main([Ljava/lang/String;)V
void
V
String[]
[Ljava/lang/String;

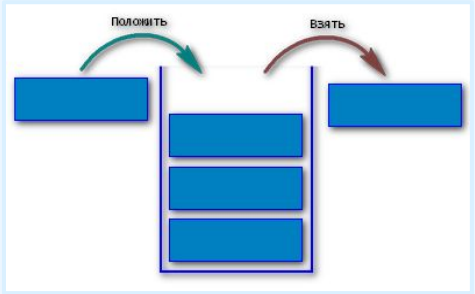
Стек

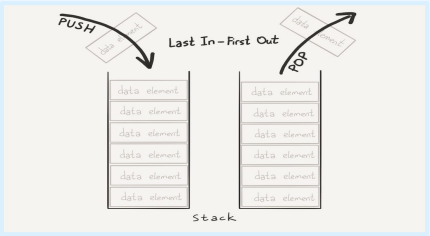

Push
Pop

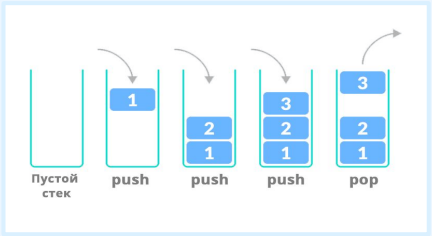

LIFO
LAST IN
FIRST OUT
8
7
6
5
4
3
2
1


вершина стека


Stack
Add New Item
Push
Pop
Get Item
itemN
...
item2
item1


Положить
Взять


PUSH
Last In - First Out
POP
data element
Stack


Пустой стек
push
push
push
pop

Опкод

```
GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
```

Мнемоника

```
GETSTATIC
```

Аргумент

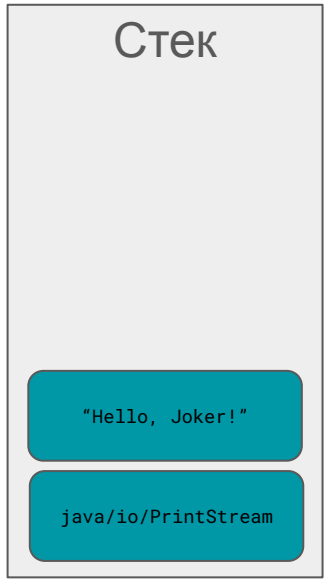
```
java/lang/System.out : Ljava/io/PrintStream;
```

Эмулируем байт-код

```
GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
LDC "Hello, Joker!"
INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
```

Console:

```
Hello, Joker!
```



invokevirtual

invokevirtual

Используется для вызова “обычных” методов

- Не final
- Не статичные
- Не конструктор
- Не super
- Не интерфейс

invokevirtual. Обычный метод

```
void run() {
    a();
}
void a() {}
```



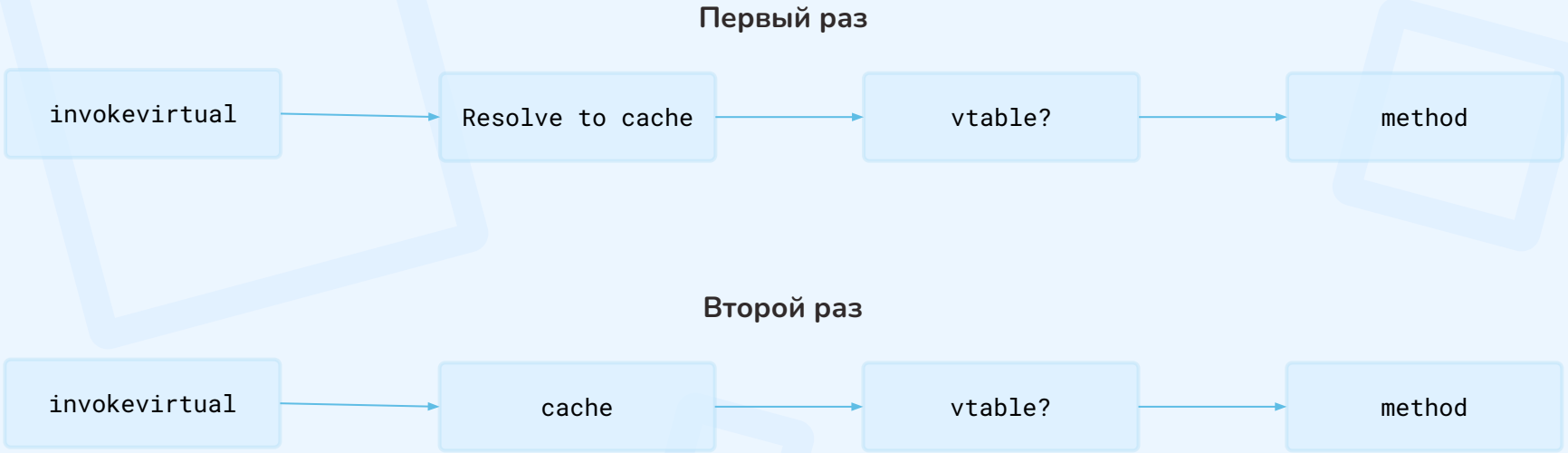
```
ALOAD 0
INVOKEVIRTUAL org/example/Main.a ()V
```

invokevirtual. Обычный метод

```
public void run() {
 b(a, b, c);
}
public void b(int a, Object b, List<Integer> c){}
```

```
ALOAD 0
ICONST_1
ACONST_NULL
ACONST_NULL
INVOKEVIRTUAL org/example/Main.b (ILjava/lang/Object;Ljava/util/List;)V
```

invokevirtual



Лямбды

Анонимный класс

Код

```
public static void main() {
    callRunnable(new Runnable()
    {
        @Override
        public void run() {
            doSmth();
        }
    });
}
```

Байт-код

```
NEW org/example/Main$1
DUP
INVOKESPECIAL org/example/Main$1.<init> ()V
INVOKESTATIC org/example/Main.callRunnable (Ljava/lang/Runnable;)V
```

Скомпилированный класс

```
class Main$1 implements Runnable {
    public void run() {
        Main.doSmth();
    }
}
```

Лямбда

Код

```
public static void run() {
    callRunnable(() -> doSmth());
}
```

Байт-код

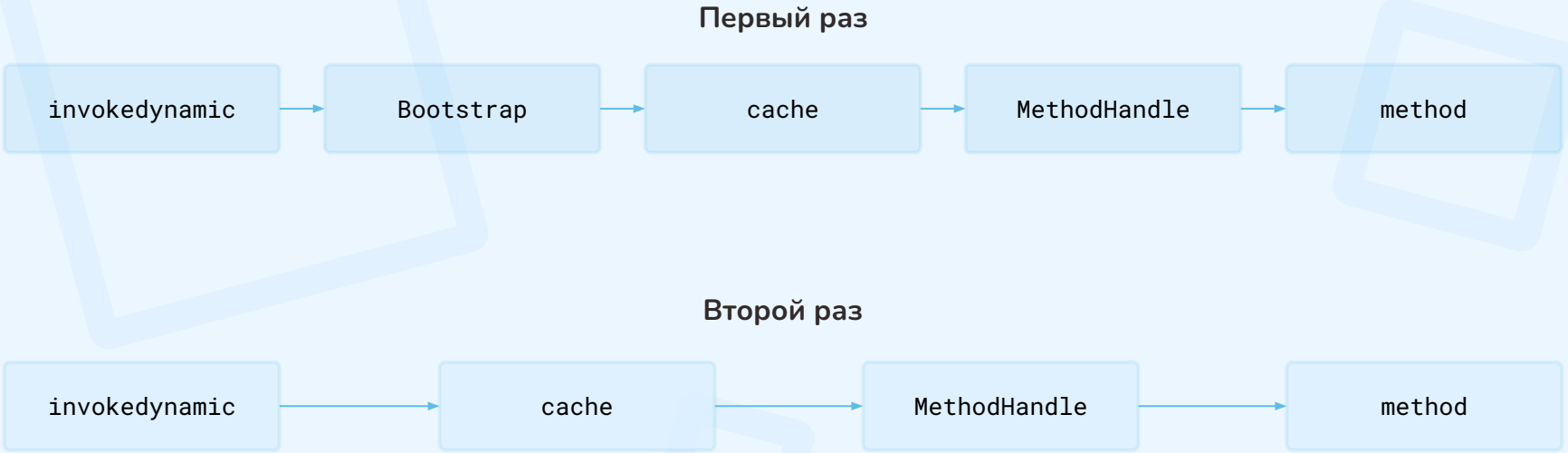
```
INVOKEDYNAMIC run()Ljava/lang/Runnable; [
  java/lang/invoke/LambdaMetafactory.metafactory(...)Ljava/lang/invoke/CallSite;

  ()V,
  org/example/Main.lambda$run$0()V,
  ()V
]
INVOKESTATIC org/example/Main.callRunnable (Ljava/lang/Runnable;)V
```

invokedynamic

- Используется для вызова метода с динамической линковкой
- Лямбды
- Конкатенация строк
- record

invokedynamic



LambdaMetafactory.metafactory

Байт-код

```
INVOKEDYNAMIC run()Ljava/lang/Runnable; [
  java/lang/invoke/LambdaMetafactory.metafactory(...)Ljava/lang/invoke/CallSite;
  ()V,
  org/example/Main.lambda$run$0()V,
  ()V
]
```

metafactory

```
public static CallSite metafactory(
        MethodHandles.Lookup caller,
        String interfaceMethodName,
        MethodType factoryType,
        MethodType interfaceMethodType,
        MethodHandle implementation,
        MethodType dynamicMethodType
)
```

metafactory. Тривиальное.

Лямбда

```
public static void run(Integer a) {
   callRunnable(() -> doSmth());
}
```

Синтетический метод

```
public static void lambda$main$0() {
   doSmth();
}
```

Сгенерированный класс

```
final class Main$$Lambda implements Runnable {
   private Main$$Lambda() {
   }
   public void run() {
       Main.lambda$main$0();
   }
}
```

metafactory. Метод референс.

Лямбда

```
public static void run() {
   callInt(Main::supplyInt);
}
```

Байт-код

```
INVOKEDYNAMIC get()Ljava/util/function/Supplier; [
java/lang/invoke/LambdaMetafactory.metafactory(...)Ljava/lang/invoke/CallSite;

    ()Ljava/lang/Object;,
    org/example/Main.supplyInt()I,
    ()Ljava/lang/Integer;
]
```

metafactory. Метод референс.

Лямбда

```
public static void main() {
   callInt(Main::supplyInt);
}
```

Сгенерированный класс

```
final class Main$$Lambda implements Supplier {
   private Main$$Lambda() {
   }
   public Object get() {
       return Main.supplyInt();
   }
}
```

```
public java.lang.Object get() {
  INVOKESTATIC org/example/Main.supplyInt()I
  INVOKESTATIC java/lang/Integer.valueOf(I)Ljava/lang/Integer;
  ARETURN
}
```

bridge методы

```
interface SupplyObj {
   Object f();
}
```

```
interface SupplyNum {
   Number f();
}
```

```
interface SupplyInt {
   Integer f();
}
```

```
interface Foo extends SupplyObj, SupplyNum, SupplyInt {}
```



```
class FooImpl implements Foo {
   @Override public Integer f() {return null;}
}
```

metafactory. Метод референс.

```
class FooImpl implements Foo {
   @Override public Integer f() {
       return null;
   }
}
```

```
class Main$FooImpl() {
 public f()Ljava/lang/Integer; {
         aconst_null
         areturn
 }

 public bridge synthetic f()Ljava/lang/Object; {
         aload 0 // reference to self
         invokevirtual org/example/Main$FooImpl.f()Ljava/lang/Integer;
         areturn
 }

 public bridge synthetic f()Ljava/lang/Number; {
         aload 0 // reference to self
         invokevirtual org/example/Main$FooImpl.f()Ljava/lang/Integer;
         areturn
 }
}
```

altMetafactory

Код

```
public static void main() {
   foo(() -> 5);
}

static void foo(Foo foo) {}
```

Байт-код

```
INVOKEDYNAMIC f()Lorg/example/Main$Foo; [
java/lang/invoke/LambdaMetafactory.altMetafactory(...)Ljava/lang/invoke/CallSite;

  ()Ljava/lang/Integer;,
  org/example/Main.lambda$main$0()Ljava/lang/Integer;,
  ()Ljava/lang/Integer;,
  4, // FLAG_BRIDGES
  2,
  ()Ljava/lang/Number;,
  ()Ljava/lang/Object;
]
```

LambdaMetafactory.altMetafactory

```
public static CallSite altMetafactory(
  MethodHandles.Lookup caller,
  String interfaceMethodName,
  MethodType factoryType,
  Object... args
)
```



```
public static CallSite altMetafactory(
  MethodHandles.Lookup caller,
  String interfaceMethodName,
  MethodType factoryType,
  MethodType interfaceMethodType,
  MethodHandle implementation,
  MethodType dynamicMethodType,
  int flags,
  int altInterfaceCount,    // IF flags has MARKERS set
  Class... altInterfaces,   // IF flags has MARKERS set
  int altMethodCount,       // IF flags has BRIDGES set
  MethodType... altMethods  // IF flags has BRIDGES set
                            )
```

LambdaMetafactory.altMetafactory

Байт код

```
INVOKEDYNAMIC f()Lorg/example/Main$Foo; [
java/lang/invoke/LambdaMetafactory.altMetafactory(...)Ljava/lang/invoke/CallSite;

  ()Ljava/lang/Integer;,
  org/example/Main.lambda$main$0()Ljava/lang/Integer;,
  ()Ljava/lang/Integer;,

  4, // FLAG_BRIDGES
  2,
  ()Ljava/lang/Number;,
  ()Ljava/lang/Object;
]
```

Псевдо сигнатура altMetafactory

```
public static CallSite altMetafactory(
  MethodHandles.Lookup caller,

  String interfaceMethodName,
  MethodType factoryType,
  MethodType interfaceMethodType,
  MethodHandle implementation,
  MethodType dynamicMethodType,

  int flags,
  int altInterfaceCount,    // IF flags has MARKERS set
  Class... altInterfaces,   // IF flags has MARKERS set
  int altMethodCount,       // IF flags has BRIDGES set
  MethodType... altMethods  // IF flags has BRIDGES set
)
```

altMetafactory. bridges

```
public static void main() {
   foo(() -> 5);
}
```



```
final class Main$$Lambda implements Main.Foo {
   private Main$$Lambda() {
   }

   public Integer f() {
      return Main.lambda$main$0();
   }

   public Number f() {
      return Main.lambda$main$0();
   }

   public Object f() {
      return Main.lambda$main$0();
   }
}
```

altMetafactory

Код

```
public static void main() {
  Foo f = (Serializable
          & Foo
          & Marker) () -> 5;
  foo(f);
}

static void foo(Foo foo) {}
interface Marker {}
```

Сгенерированный класс

```
final class Main$$Lambda implements SupplyInt,
 Foo, Marker, Serializable {

   public Integer f() {
      return Main.lambda$main$bbee6ded$1();
   }

// Bridges…

   private final Object writeReplace() {
      return new SerializedLambda(...);
   }
}
```

Почему так?

We could “just” user inner classes

- We could define that a lambda is “just” an inner class instance (where the compiler spins the inner class)
  - `p -> p.age < k` translates to

```
class Foo$1 implements Predicate<Person> {
    private final int $v0;
    Foo$1(int v0) { this.$v0 = v0; }
    public boolean test(Person p) {
        return p.age < $v0;
    }
}
```

  - Capture == invoke constructor (`new Foo$1(k)`)
  - One class per lambda expression – yuck
  - Would like to improve over inner classes
    - If we define things this way, we’re stuck with inner class behavior forever
    - Back to that “conflates binary representation with implementation” problem

ORACLE

Почему так?

- Не хотели делать лишние классы
- Хотели иметь по-настоящему анонимные классы
- Хотели иметь возможность в будущем изменять имплементацию

А бенчи будут?

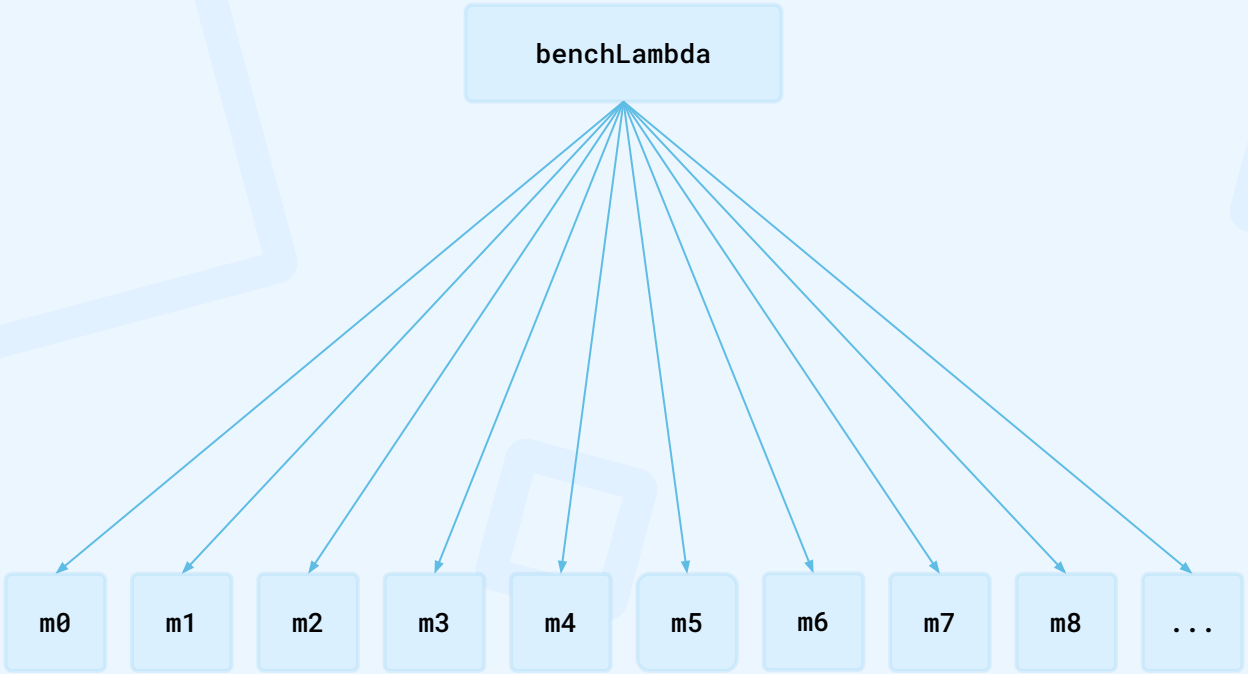
Бенчи лямбды

```
private static void benchLambda() {
   consume(() -> 0);
   consume(() -> 1);
   consume(() -> 2);
   consume(() -> 3);
   consume(() -> 4);
   consume(() -> 5);
   consume(() -> 6);
   // ...
}
```

Бенчи лямбды

```
private static void benchAnon() {
      consume(new Supplier<Integer>() { public Integer get() { return 0;}});
      consume(new Supplier<Integer>() { public Integer get() { return 1;}});
      consume(new Supplier<Integer>() { public Integer get() { return 2;}});
      consume(new Supplier<Integer>() { public Integer get() { return 3;}});
      consume(new Supplier<Integer>() { public Integer get() { return 4;}});
      consume(new Supplier<Integer>() { public Integer get() { return 5;}});
      consume(new Supplier<Integer>() { public Integer get() { return 6;}});
      // ...
}
```

Бенчи лямбды



Бенчи лямбды

```
lambdaCold elapsed 176.535398ms.
lambdaHot elapsed 103.910383ms.
lambdaVeryHot elapsed 0.492593ms.


anonCold elapsed 137.405322ms.
anonHot elapsed 92.048287ms.
anonVeryHot elapsed 0.625313ms.
```

Бенчи лямбды

```
public void consume(Blackhole bh, Lambda c) {
 bh.consume(c);
}

public interface Lambda {
 int get();
}
```

Бенчи лямбды

```
@Benchmark
public void benchLambda(Blackhole bh, MyState state) {
 consume(bh, () -> state.value);
}
```

Бенчи лямбды

```
@Benchmark
public void benchAnon(Blackhole bh, MyState state) {
 consume(bh, new Lambda() {
   @Override
   public int get() {
     return state.value;
   }
 });
}
```

Бенчи лямбды

Benchmark	Mode	Cnt	Score + Error	Units
LambdaBenches.benchAnon	thrpt	25	0.384 ± 0.026	ops/ns
LambdaBenches.benchLambda	thrpt	25	0.398 ± 0.038	ops/ns
LambdaBenches.benchAnon	avgt	25	2.315 ± 0.024	ns/op
LambdaBenches.benchLambda	avgt	25	2.338 ± 0.082	ns/op

СПАСИБО ЗА ВНИМАНИЕ!

ГОТОВ ОТВЕТИТЬ НА ВАШИ ВОПРОСЫ