WPF Compiler und Interpreter:

Java-Hardener

Christoph Jerolimov, Martrikelnr. 11084742
Sommersemester 2013

Idee

```
// Objective-C
NSArray* list = ...
if (!list.count) {
    // Runs when list is nil and empty!
}

// Java
List list = ...
if (!list.isEmpty()) {
    // Should run when list is null?
}
```

Idee

```
// Objective-C
NSArray* list = ...
if (!list.count) {
    // Runs when list is nil and empty!
}

// Java
List list = ...
if (!list.isEmpty()) {
    // Should run when list is null?
}
```

→ throws NullPointerException

NullPointerException

```
public class Demo {
   public static void main(String[] args) {
      Map<String, String> map = new LinkedHashMap<String, String>();
      map.put("key", "value");

      System.out.println("Key length: " + map.get("key").length());
   }
}
```

→ `Key length: 5`

NullPointerException

```
public class Demo {
   public static void main(String[] args) {
      Map<String, String> map = new LinkedHashMap<String, String>();
      map.put("key", "value");

      System.out.println("Key length: " + map.get("does not exist").length());
   }
}
```

NullPointerException

```
public class Demo {
   public static void main(String[] args) {
      Map<String, String> map = new LinkedHashMap<String, String>();
      map.put("key", "value");

      System.out.println("Key length: " + map.get("does not exist").length());
   }
}
```

```
Demo!Exception in thread "main" java.lang.NullPointerException
   at Demo.main(Demo.java:39)
```

NullPointerException

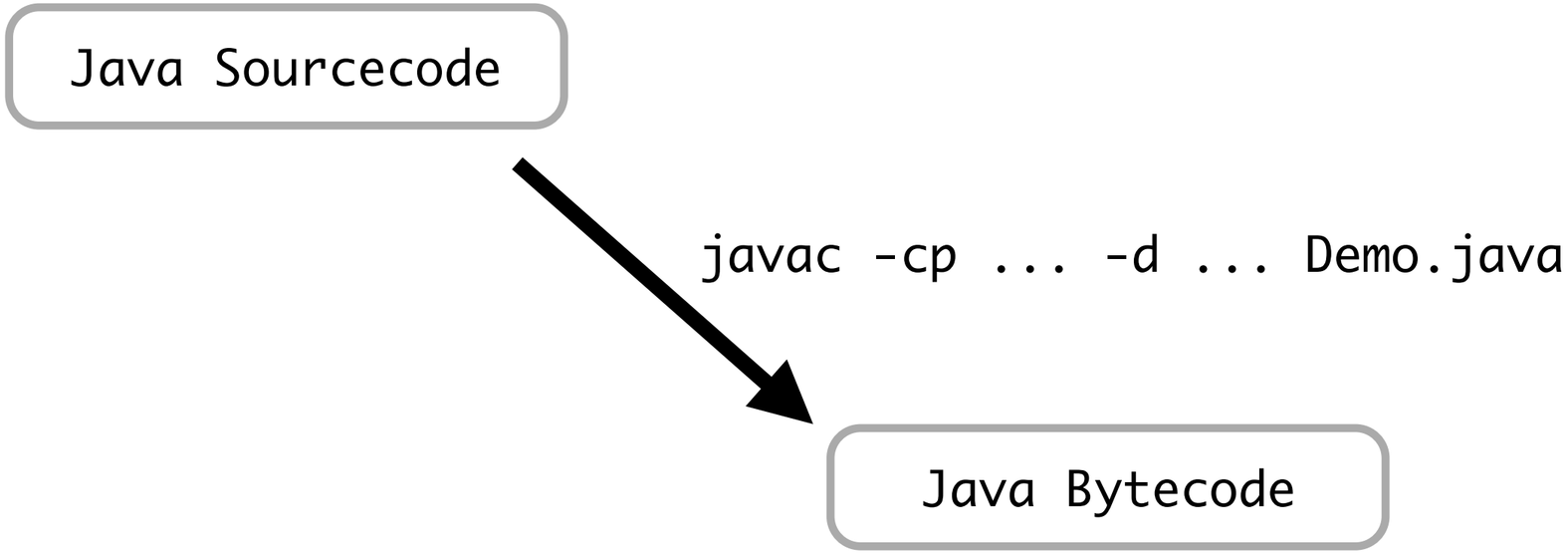
```
public class Demo {
   public static void main(String[] args) {
      Map<String, String> map = new LinkedHashMap<String, String>();
      map.put("key", "value");

      System.out.println("Key length: " + map.get("does not exist").length());
   }
}
```

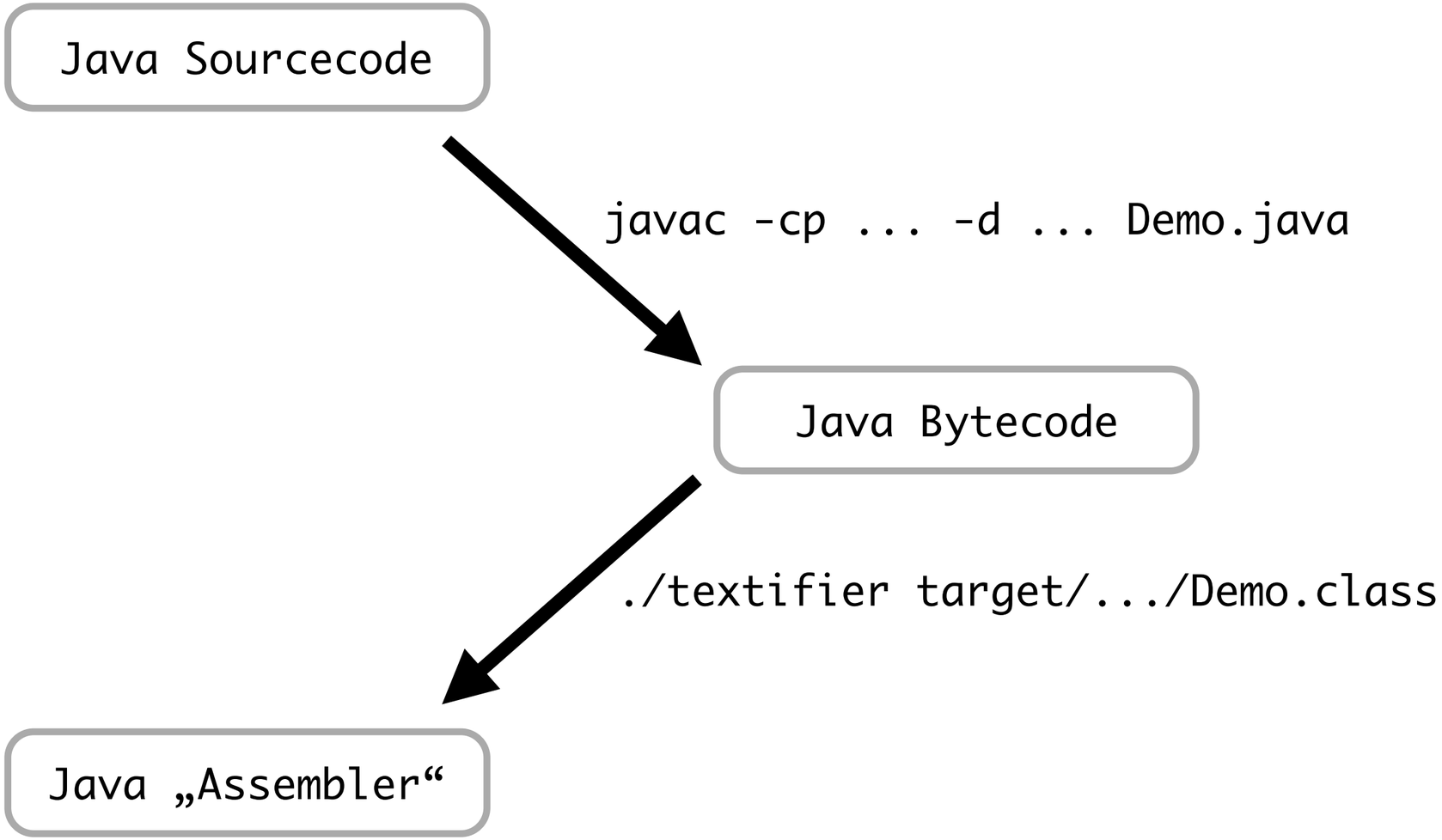
```
Demo!Exception in thread "main" java.lang.NullPointerException
   at Demo.main(Demo.java:39)
```

In Objective-C this will just outout: Key length: 0

Bytecode Analyse

Java Sourcecode


Java Sourcecode
javac -cp ... -d ... Demo.java
Java Bytecode


Java Sourcecode
javac -cp ... -d ... Demo.java
Java Bytecode
./textifier target/.../Demo.class
Java „Assembler“

Test1.java

```
public class Test1 {
   public static void main(String[] args) {
      Map map = new LinkedHashMap();
      String entry = map.get("key");
      System.out.println(entry.length());
   }
}
```

Test1.java

```java
public class Test1 {
   public static void main(String[] args) {
      Map map = new LinkedHashMap();
      String entry = map.get("key");
      System.out.println(entry.length());
   }
}
```

```
// class version 51.0 (51)
// access flags 0x21
public class de/fhkoeln/.../Test1 {

  // access flags 0x9
  public static main([LString;)V
    NEW java/util/LinkedHashMap
    DUP
    INVOKESPECIAL LinkedHashMap.<init>
    ASTORE 1
    ALOAD 1
    LDC "key"
    INVOKEINTERFACE Map.get (LObject;)
    CHECKCAST String
    ASTORE 2
    ...
   RETURN
    MAXSTACK = 2
    MAXLOCALS = 3
}
```

Test1.java

```java
public class Test1 {
   public static void main(String[] args) {
      Map map = new LinkedHashMap();
      String entry = map.get("key");
      System.out.println(entry.length());
   }
}
```

```
...
GETSTATIC System.out : LPrintStream;
ALOAD 2
INVOKEVIRTUAL String.length ()I
INVOKEVIRTUAL PrintStream.println (I)V
...
```

Test2.java

```java
public class Test2 {
   public static void main(String[] args) {
      Map map = new LinkedHashMap();
      String entry = map.get("key");
      System.out.println(
        entry != null ? entry.length() : 0
      );
   }
}
```

```
 ...
 GETSTATIC System.out : LPrintStream;
 ALOAD 2
 IFNULL L0
 ALOAD 2
 INVOKEVIRTUAL String.length ()I
 GOTO L1
L0
FRAME FULL [[LString; Map java/lang/Str
 ICONST_0
L1
FRAME FULL [[LString; Map java/lang/Str
 INVOKEVIRTUAL PrintStream.println (I)V
```

Test3.java

```java
public class Test3 {
   public static void main(String[] args) {
      Map map = new LinkedHashMap();
      String entry = map.get("key");
      int l;
      try {
         l = entry.length();
      } catch (NullPointerException e) {
         l = 0;
      }
      System.out.println(l);
   }
}
```

```
   TRYCATCHBLOCK L0 L1 L2 java/lang/NullPo
   ...
L0:
   ALOAD 2
   INVOKEVIRTUAL String.length ()I
   ISTORE 3
L1:
   GOTO L3
L2:
   FRAME FULL [[LString; Map String] [Nul
   ASTORE 4
   ICONST_0
   ISTORE 3
 L3
   FRAME APPEND [I]
   GETSTATIC System.out : LPrintStream;
   ILOAD 3
   INVOKEVIRTUAL PrintStream.println (I)V
   ...
```

Umsetzung

Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   IFNULL fallback
   INVOKE_* // Original method

fallback:
   ICONST_0
```

Stack

Theorie

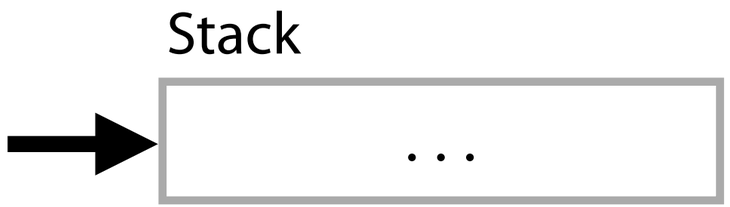
„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   IFNULL fallback
   INVOKE_* // Original method

fallback:
   ICONST_0
```

Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   IFNULL fallback
   INVOKE_* // Original method

fallback:
   ICONST_0
```

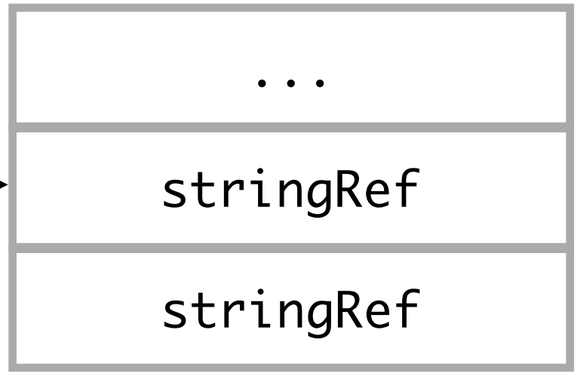


Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   DUP
   IFNULL fallback
   INVOKE_* // Original method
   GOTO behind

fallback:
   POP
   ICONST_0

behind:
```

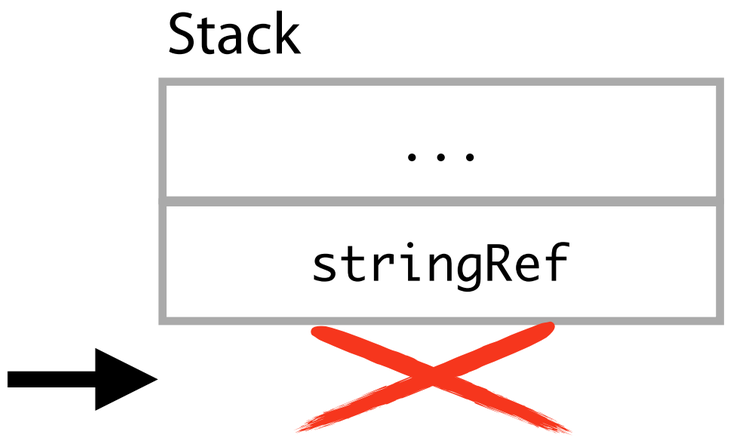
Stack



Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

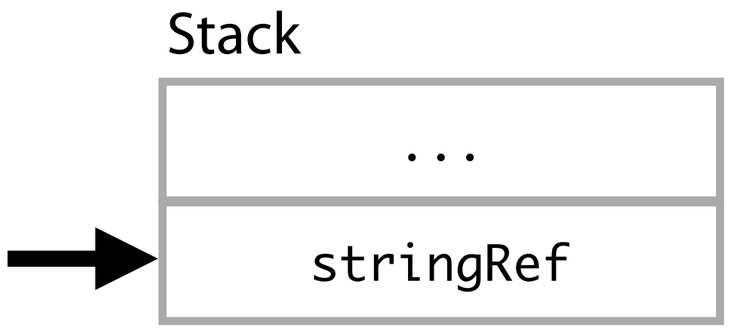
```
   DUP
   IFNULL fallback
   INVOKE_* // Original method
   GOTO behind

fallback:
   POP
   ICONST_0

behind:
```



Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   DUP
   IFNULL fallback
   INVOKE_* // Original method
   GOTO behind

fallback:
   POP
   ICONST_0

behind:
```

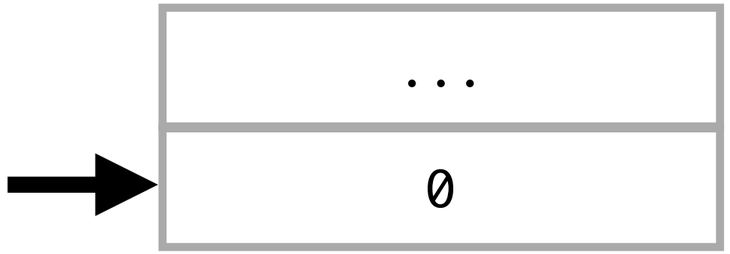


Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   DUP
   IFNULL fallback
   INVOKE_* // Original method
   GOTO behind

fallback:
   POP
   ICONST_0

behind:
```

Stack



Theorie

„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

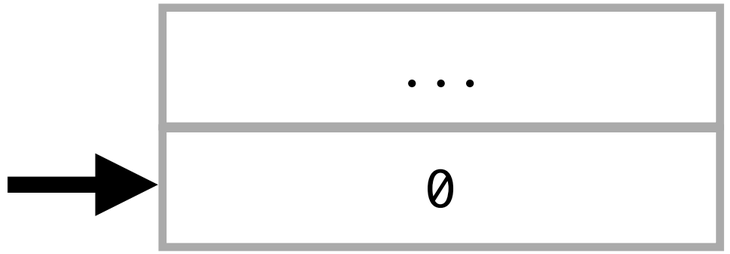
```
   DUP
   IFNULL fallback
   INVOKE_* // Original method
   GOTO behind

fallback:
   POP
   ICONST_0

behind:
```

Theorie

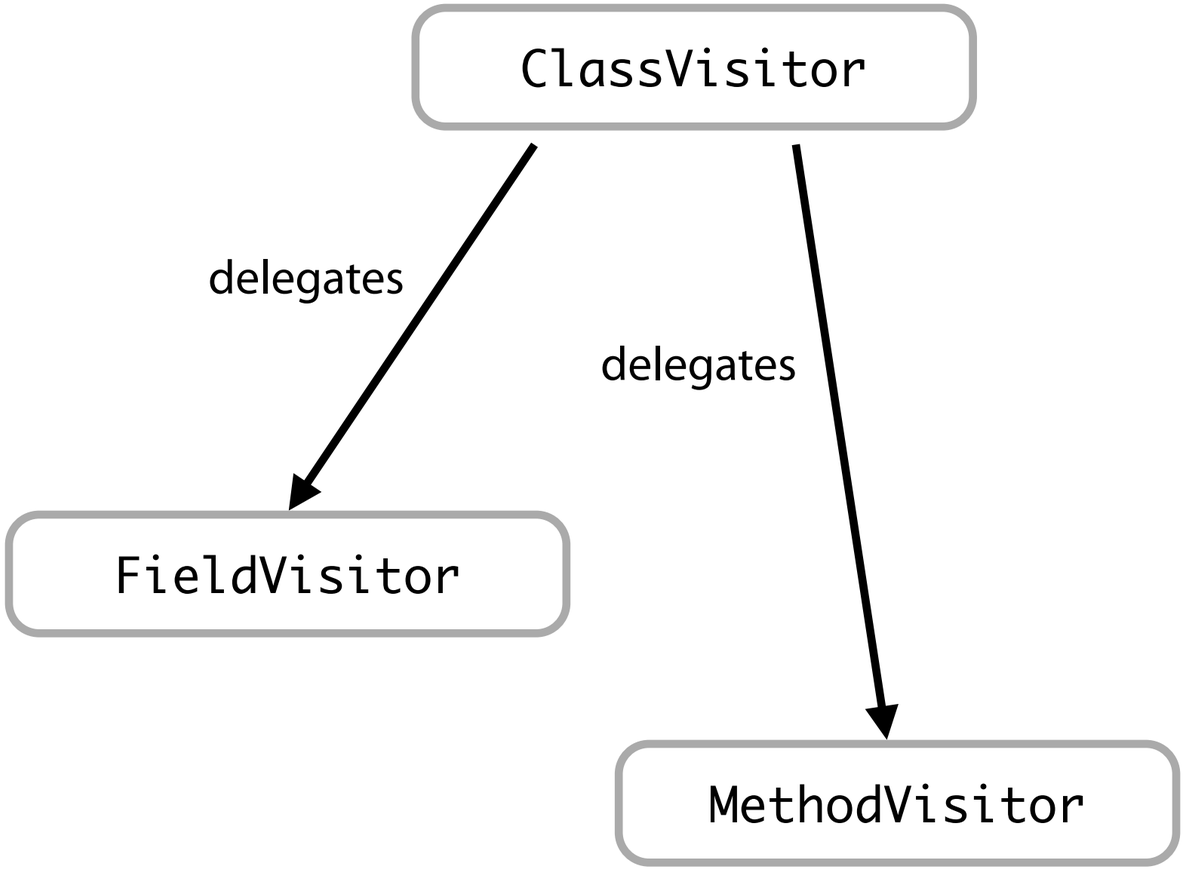
„Immer wenn ein `INVOKE_*` kommt möchten wir diesen mit IFNULL absichern!“

```
   DUP
   IFNULL fallback
   INVOKE_* // Original method
   GOTO behind

fallback:
   POP
   ICONST_0

behind:
```

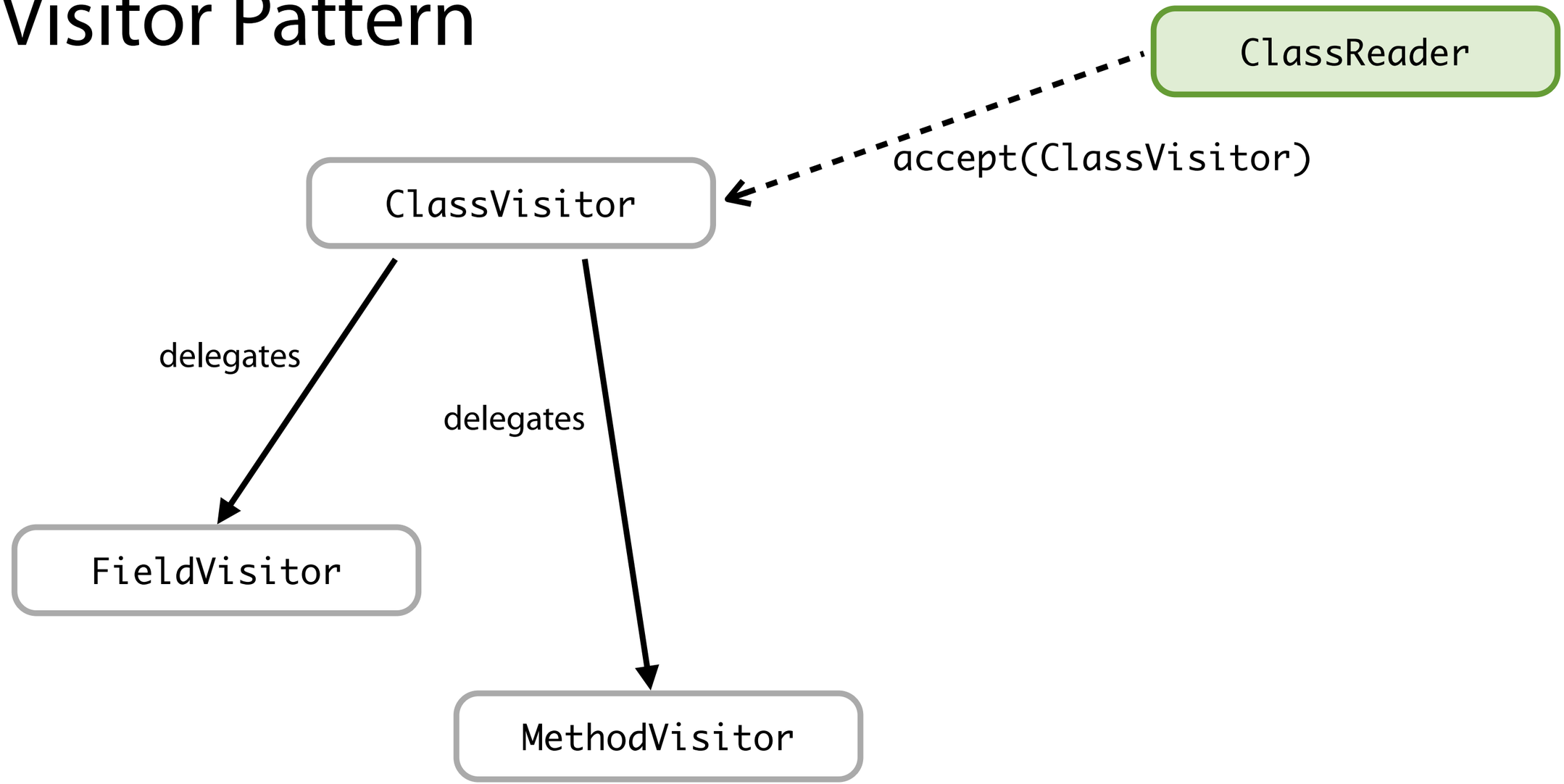
Stack



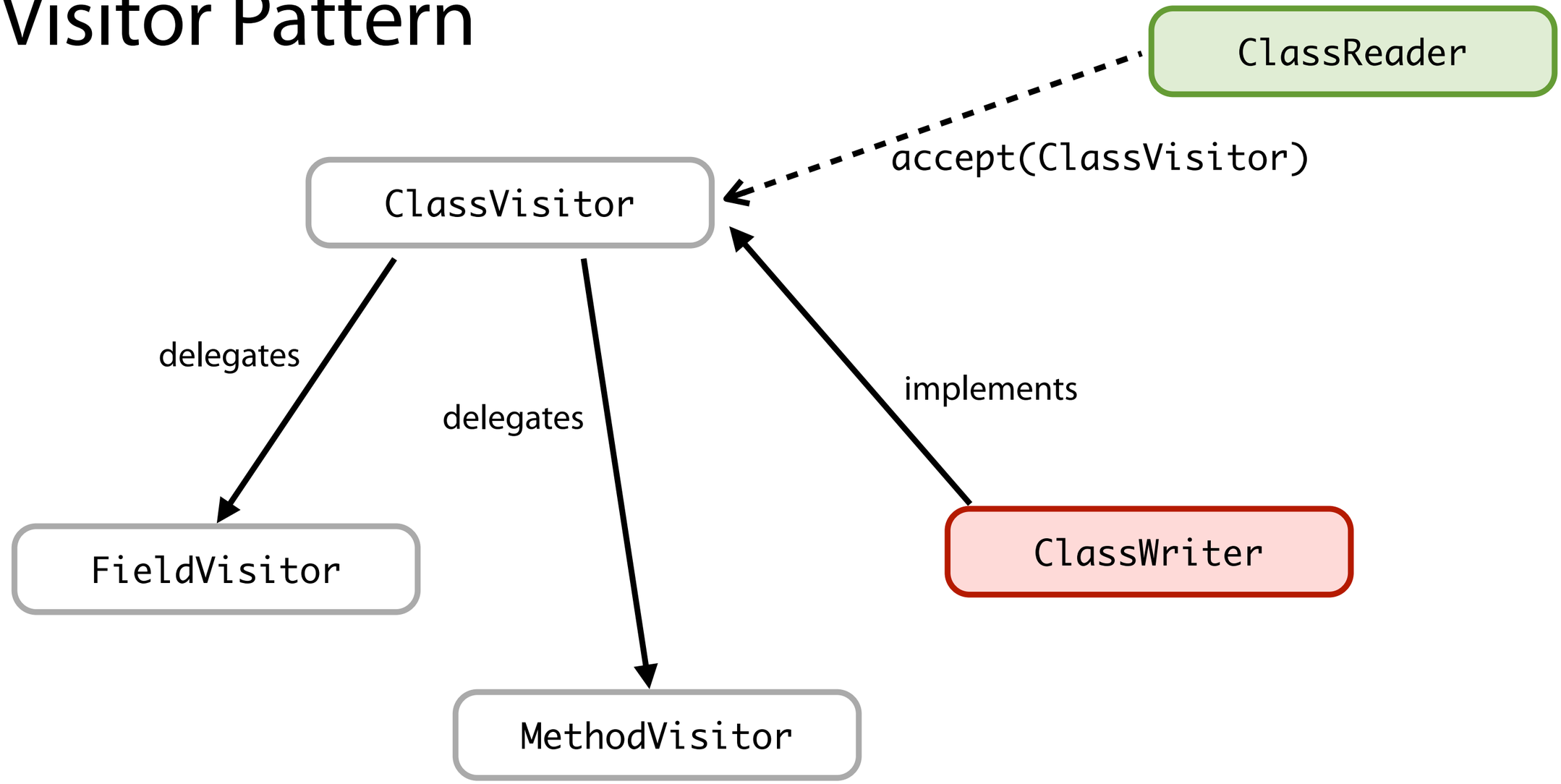
Visitor Pattern



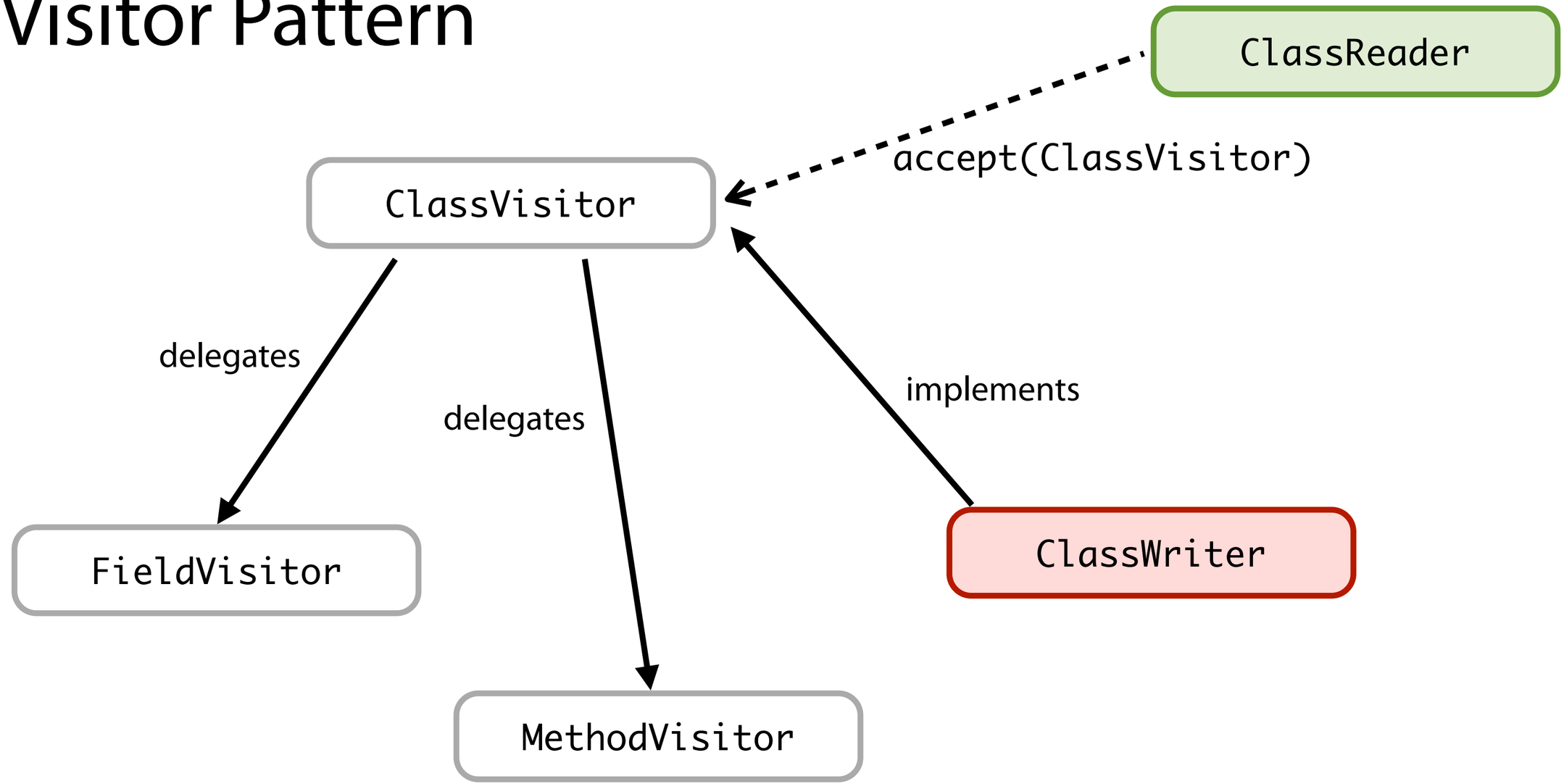
Visitor Pattern



Visitor Pattern



Visitor Pattern



Alle Visitor haben eine Referenz auf sich selbst,
hierdurch wird das Hintereinander schalten der Visitoren ermöglicht!

ClassVisitor Implementierung

```
public class CheckNullClassVisitor extends ClassVisitor {
   public CheckNullClassVisitor(ClassVisitor visitor) {
      super(Opcodes.ASM4, visitor);
   }

   @Override
   public MethodVisitor visitMethod(
         int access, String name, String desc,
         String signature, String[] exceptions) {

      MethodVisitor visitor = super.visitMethod(access, name, desc, signature, except

      return new CheckNullMethodVisitor(visitor);
   }
}
```

MethodVisitor Implementierung

```
public void visitMethodInsn(int opcode, String owner, String name, String desc) {

  if (opcode == Opcodes.INVOKEINTERFACE || opcode == Opcodes.INVOKEVIRTUAL) {

    if ((owner + "." + name).equals("java/lang/String.length")) {

      // TODO

    } else {
      super.visitMethodInsn(opcode, owner, name, desc);
    }
  } else {
    super.visitMethodInsn(opcode, owner, name, desc);
  }
}
```

MethodVisitor Implementierung

```
Label fallback = new Label();
Label behind = new Label();

// We surround the original call with an IFNULL check:
super.visitInsn(Opcodes.DUP); // Duplicate stack pointer
super.visitJumpInsn(Opcodes.IFNULL, fallback); // Skip method call if reference is n
super.visitMethodInsn(opcode, owner, name, desc); // Original method call
super.visitJumpInsn(Opcodes.GOTO, behind); // Jump over the reference is null path

// But if not we need add a default value to the stack:
super.visitLabel(fallback); // Reference is null path
super.visitInsn(Opcodes.POP); // Pop the dup value from stack
super.visitInsn(Opcodes.ICONST_0); // Push int 0 to the stack

super.visitLabel(behind); // Label to jump of the reference is null patha
```

Probleme

Rückgabetyp

```
Label fallback = new Label();
Label behind = new Label();

// We surround the original call with an IFNULL check:
super.visitInsn(Opcodes.DUP); // Duplicate stack pointer
super.visitJumpInsn(Opcodes.IFNULL, fallback); // Skip method call if reference is n
super.visitMethodInsn(opcode, owner, name, desc); // Original method call
super.visitJumpInsn(Opcodes.GOTO, behind); // Jump over the reference is null path

// But if not we need add a default value to the stack:
super.visitLabel(fallback); // Reference is null path
super.visitInsn(Opcodes.POP); // Pop the dup value from stack
super.visitInsn(Opcodes.ICONST_0); // Push int 0 to the stack

super.visitLabel(behind); // Label to jump of the reference is null patha
```

Stack-Reihenfolge bei Methoden mit Argumenten

```
Label fallback = new Label();
Label behind = new Label();

// We surround the original call with an IFNULL check:
super.visitInsn(Opcodes.DUP); // Duplicate stack pointer
super.visitJumpInsn(Opcodes.IFNULL, fallback); // Skip method call if reference is n
super.visitMethodInsn(opcode, owner, name, desc); // Original method call
super.visitJumpInsn(Opcodes.GOTO, behind); // Jump over the reference is null path

// But if not we need add a default value to the stack:
super.visitLabel(fallback); // Reference is null path
super.visitInsn(Opcodes.POP); // Pop the dup value from stack
super.visitInsn(Opcodes.ICONST_0); // Push int 0 to the stack

super.visitLabel(behind); // Label to jump of the reference is null patha
```

Stack-Reihenfolge bei Methoden mit Argumenten

```
Label fallback = new Label();
Label behind = new Label();

// We surround the original call with an IFNULL check:
super.visitInsn(Opcodes.DUP); // Duplicate stack pointer
super.visitJumpInsn(Opcodes.IFNULL, fallback); // Skip method call if reference is n
super.visitMethodInsn(opcode, owner, name, desc); // Original method call
super.visitJumpInsn(Opcodes.GOTO, behind); // Jump over the reference is null path

// But if not we need add a default value to the stack:
super.visitLabel(fallback); // Reference is null path
super.visitInsn(Opcodes.POP); // Pop the dup value from stack
super.visitInsn(Opcodes.ICONST_0); // Push int 0 to the stack

super.visitLabel(behind); // Label to jump of the reference is null patha
```

→ Funktioniert aktuell nur mit maximal einem Argument! (DUP2 und POP2)

Demo

Vielen Dank