Java Langage & JRE Internal Basics

This document:

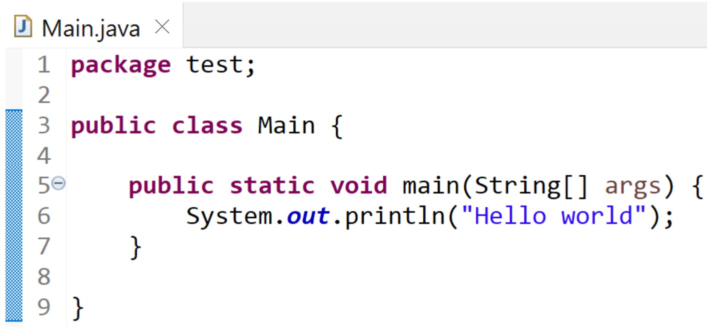
https://github.com/Arnaud-Nauwynck/presentations/blob/main/java/Java-Langage-JRE-Internal-Basics.pdf

Outline

- Overview compile – runtime chain
- Compiler basics: grammar, parser to AST
  - Declaration-Statement-Expression
  - Bytecode, stack
- Langage Class Symbol resolution
  - Class.forName() / ClassLoader
  - First reference, Hot swap code
- Method Symbol resolution + call
  - Invokestatic, invokespecial
  - Invokevirtual
  - Invokeinterface
  - invokedynamic

Compile – Runtime Chain

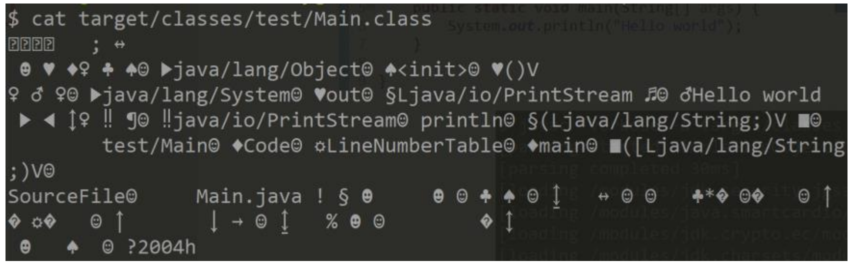
UTF-8 files
src/main/java/*.java



```java
package test;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

javac

Bytecode (binary) files
target/classes/*.class



```
$ cat target/classes/test/Main.class
????    ; ↔
 ☻ ♥ ♦♀ ♣ ♠☺ ►java/lang/Object☺ ♠<init>☺ ♥()V
♀ ♂ ♀☺ ►java/lang/System☺ ♥out☺ §Ljava/io/PrintStream ♫☺ ♂Hello world
 ► ◄ ↕♀ ‼ ¶☺ ‼java/io/PrintStream☺ println☺ §(Ljava/lang/String;)V ■☺
          test/Main☺ ♦Code☺ ☼LineNumberTable☺ ♦main☺ ■([Ljava/lang/String
;)V☺
SourceFile☺     Main.java ! § ☻     ☻ ☺ ♣ ♠ ☺ ↨   ↔ ☺ ☺   ♣*� ☺�   ☺ ↑
� ☼�    ☺ ↑        ↓ → ☺ ↨   % ☻ ☺         � ↕
 ☻    ♠   ☺ ?2004h
```

jar

target/*.jar

(jar = zip file
of *.class)

*.jar or .class
In CLASSPATH

+ main FQN

java

JRE

Symbols

C0
bytecode
interpreter

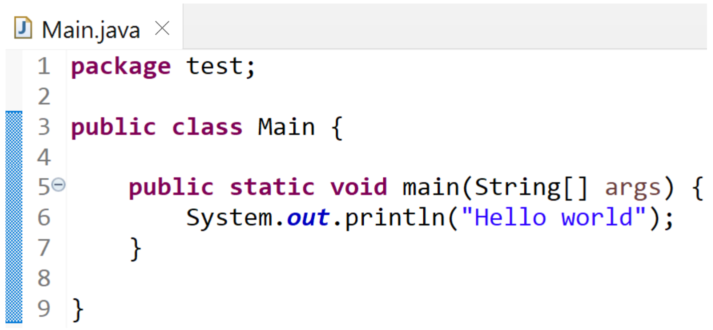
.. bytecode
link resolved
On first use

C1
assembly
langage

C2
assembly
... optimized

Compile (+Extension) – Runtime(+JVM Agent) Chain

UTF-8 files
src/main/java/*.java



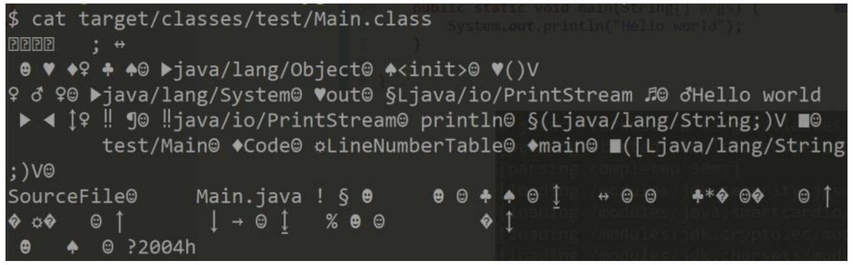
Main.java

```java
package test;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

javac

Compiler Extensions
(ex: Lombok)

Bytecode (binary) files
target/classes/*.class



```
$ cat target/classes/test/Main.class
????    ; ↔
 ☻ ♥ ♦♀ ♣ ♠☺ ►java/lang/Object☺ ♠<init>☺ ♥()V
♀ ♂ ♀☺ ►java/lang/System☺ ♥out☺ §Ljava/io/PrintStream ♫☺ ♂Hello world
 ► ◄ ↕♀ ‼ ¶☺ ‼java/io/PrintStream☺ println☺ §(Ljava/lang/String;)V ■☺
       test/Main☺ ♦Code☺ ☼LineNumberTable☺ ♦main☺ ■([Ljava/lang/String
;)V☺
SourceFile☺     Main.java ! § ☻      ☻ ☺ ♣ ♠ ☺ ↨   ↔ ☺ ☺   ♣*� ☺�   ☺ ↑
� ☼�    ☺ ↑        ↓ → ☺ ↨   % ☻ ☺        � ↕
 ☻    ♠  ☺ ?2004h
```

jar

target/*.jar

(jar = zip file
of *.class)

*.jar or .class
In CLASSPATH

+ main FQN

java

javaagent Extensions
(ex: glowroot profiler)

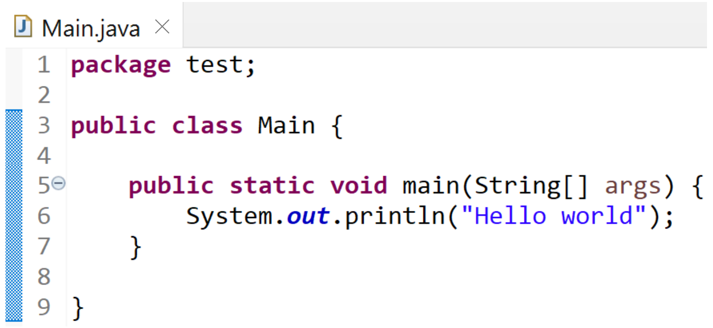
JRE

Symbols

$$\begin{tikzcd}
\text{C0 bytecode interpreter} \arrow[r] & \text{.. bytecode link resolved On first use} \arrow[r] & \text{C1 assembly langage} \arrow[r] & \text{C2 assembly ... optimized}
\end{tikzcd}$$

Compile steps

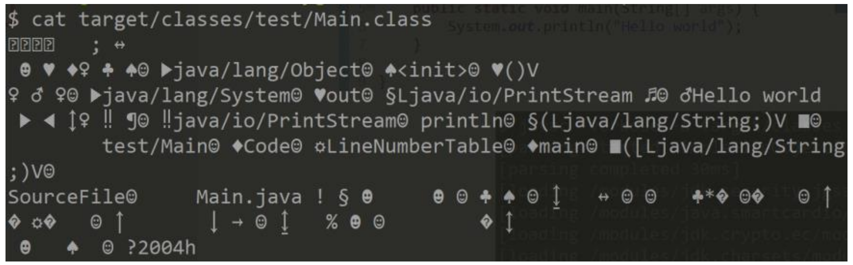
UTF-8 files
src/main/java/*.java



```java
package test;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

javac

Bytecode (binary) files
target/classes/*.class

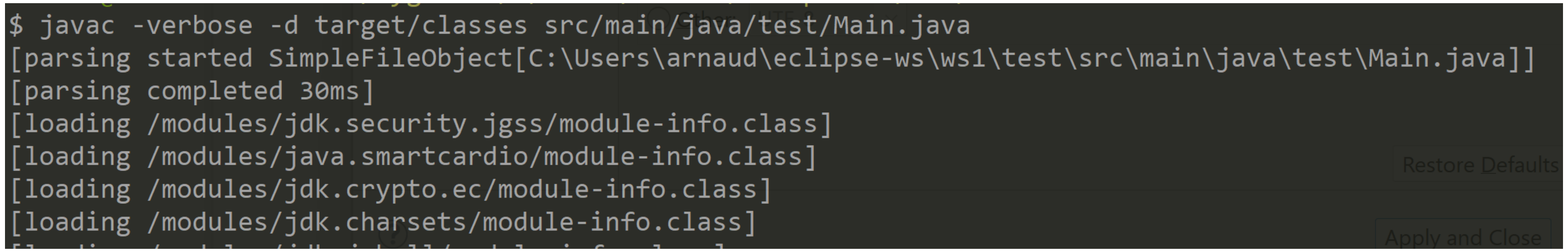


```
$ cat target/classes/test/Main.class
????    ; ↔
 ☻ ♥ ♦♀ ♣ ♠☺ ►java/lang/Object☺ ♠<init>☺ ♥()V
♀ ♂ ♀☺ ►java/lang/System☺ ♥out☺ §Ljava/io/PrintStream ♬☺ ♂Hello world
 ► ◄ ↕♀ ‼ ¶☺ ‼java/io/PrintStream☺ println☺ §(Ljava/lang/String;)V ■☺
        test/Main☺ ♦Code☺ ☼LineNumberTable☺ ♦main☺ ■([Ljava/lang/String
;)V☺
SourceFile☺     Main.java ! § ☻     ☻ ☺ ♣ ♠ ☺ ↕   ↔ ☺ ☺   ♣*� ☺�   ☺ ↑
� ☼�    ☺ ↑         ↓ → ☺ ↕   % ☻ ☺        � ↕
 ☻    ♠   ☺ ?2004h
```

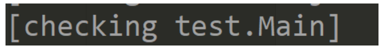
jar

Zip file
target/*.jar

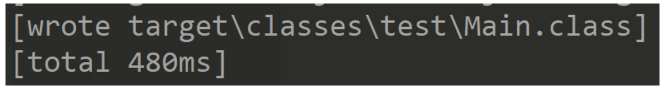
mvn package



```
$ javac -verbose -d target/classes src/main/java/test/Main.java
[parsing started SimpleFileObject[C:\Users\arnaud\eclipse-ws\ws1\test\src\main\java\test\Main.java]]
[parsing completed 30ms]
[loading /modules/jdk.security.jgss/module-info.class]
[loading /modules/java.smartcardio/module-info.class]
[loading /modules/jdk.crypto.ec/module-info.class]
[loading /modules/jdk.charsets/module-info.class]
```

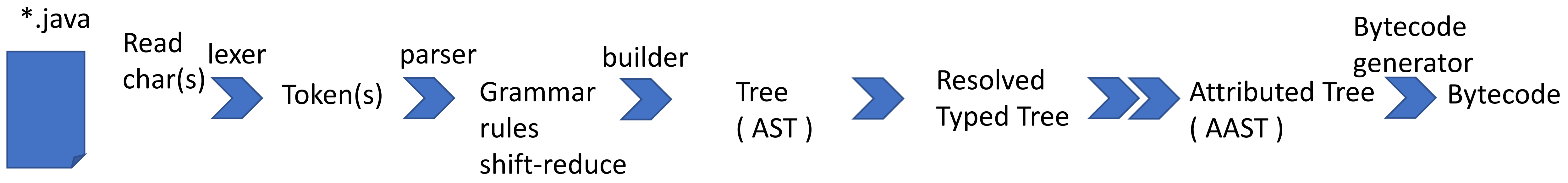


```
[checking test.Main]
```



```
[wrote target\classes\test\Main.class]
[total 480ms]
```

Javac .. steps



CST = **Concrete** Syntaxic Tree

contain « ; » and parenthesis « () »

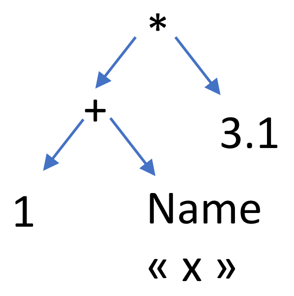
AST = **Abstract Syntaxic Tree**

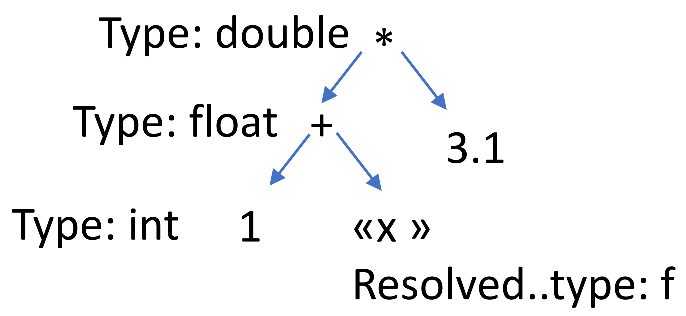
AAST =
Attributed
Abstract Syntaxic Tree

Stack
Operator

Example:

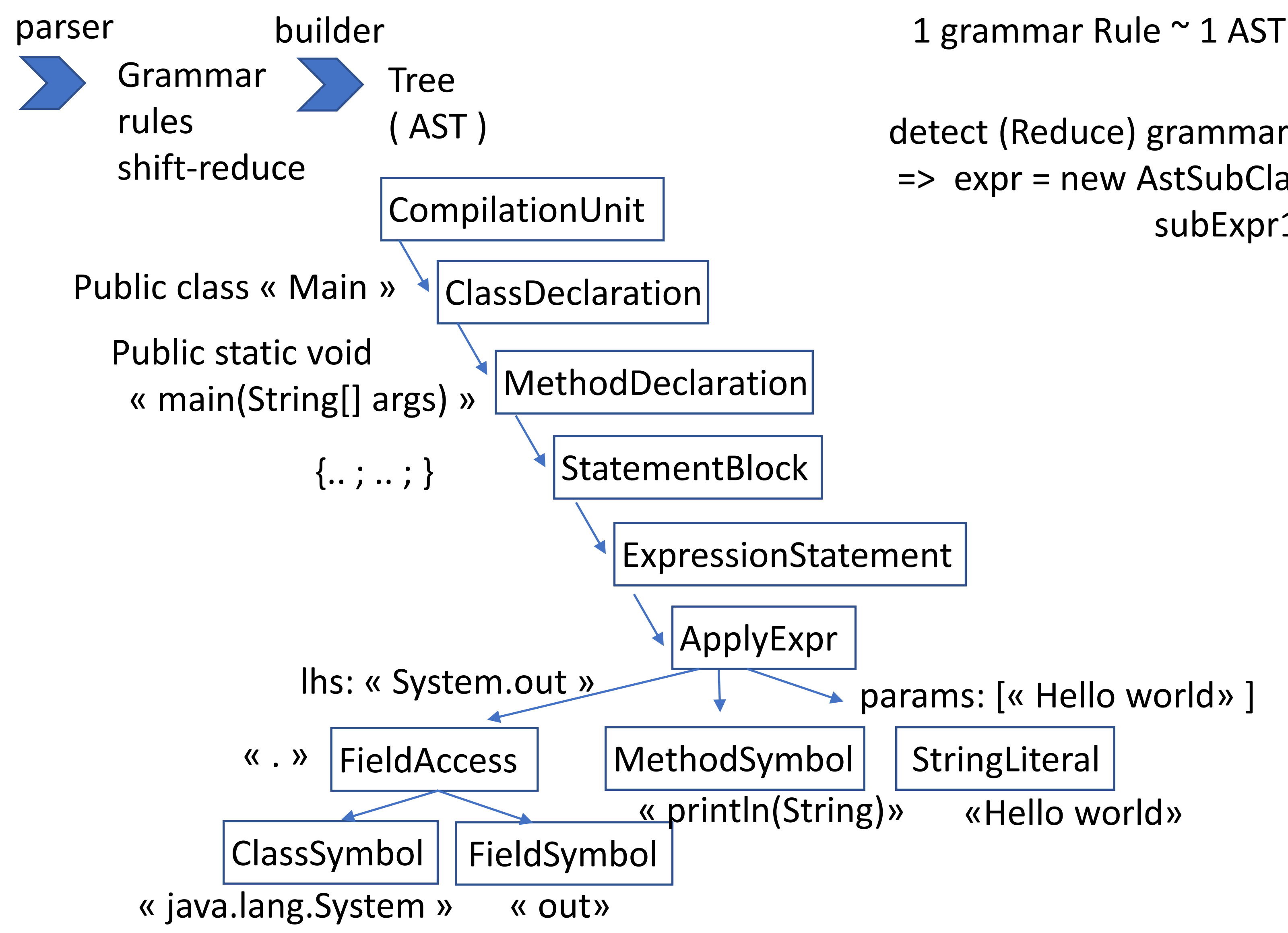
« (1 + x) * 3.1 »





push 1
push x
fadd
push 3.1
dmult

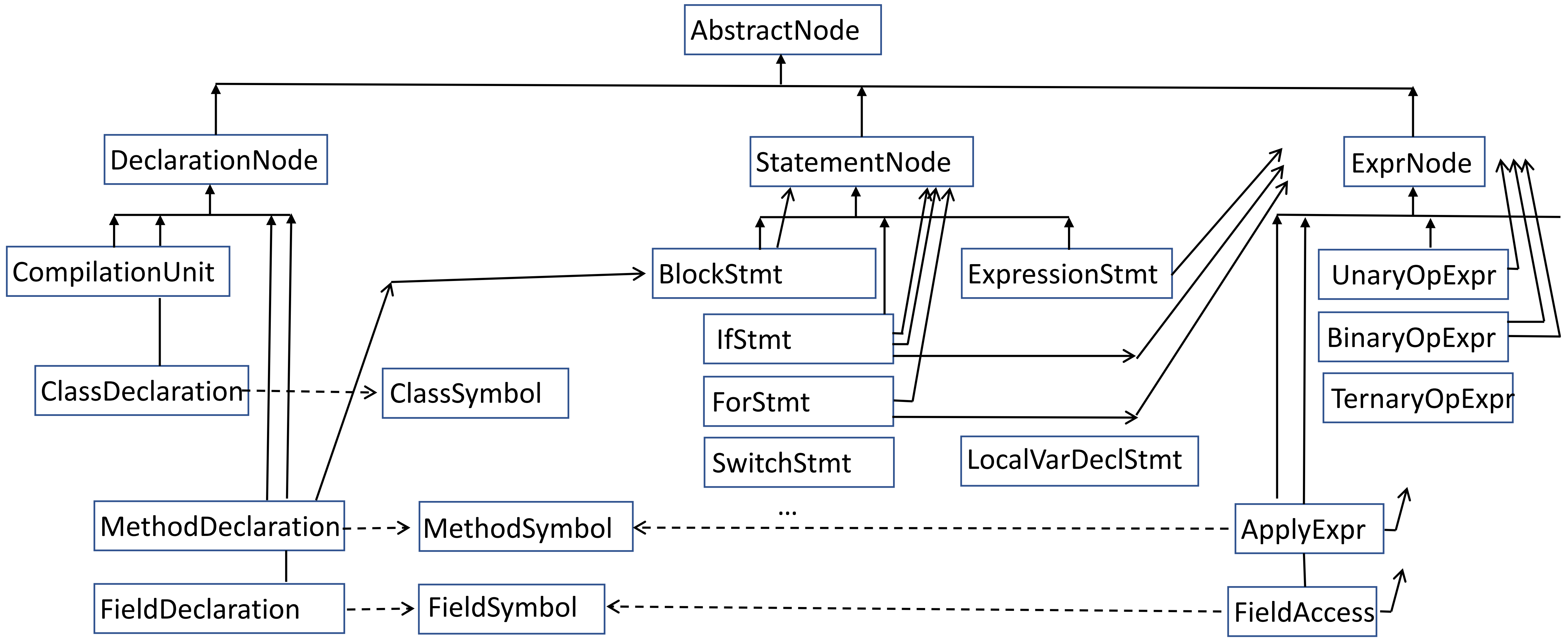
Javac .. steps



1 grammar Rule ~ 1 AST sub-class

detect (Reduce) grammar rule
=> expr = new AstSubClass(
subExpr1, ..subExprN)

Abstract Syntaxic Tree



Declaration

Something that can be prefixed with access modifier
« public|protected|private »   « static » « final »

A declaration produces a « symbol », that can be imported / used

Statement

Something that can be suffixed by « ; »
or wrapped in « { ; ; } »

A statement has no type (or « void »)

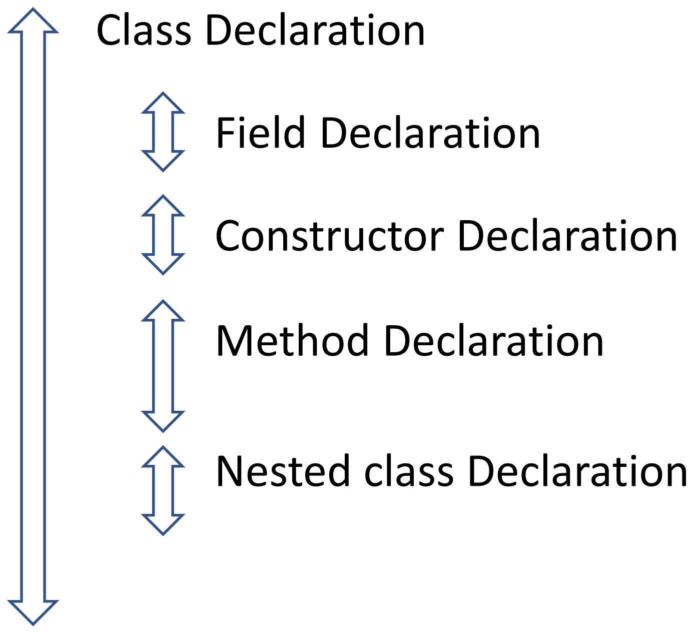
Expression

Something that can be wrapped by « (..) »

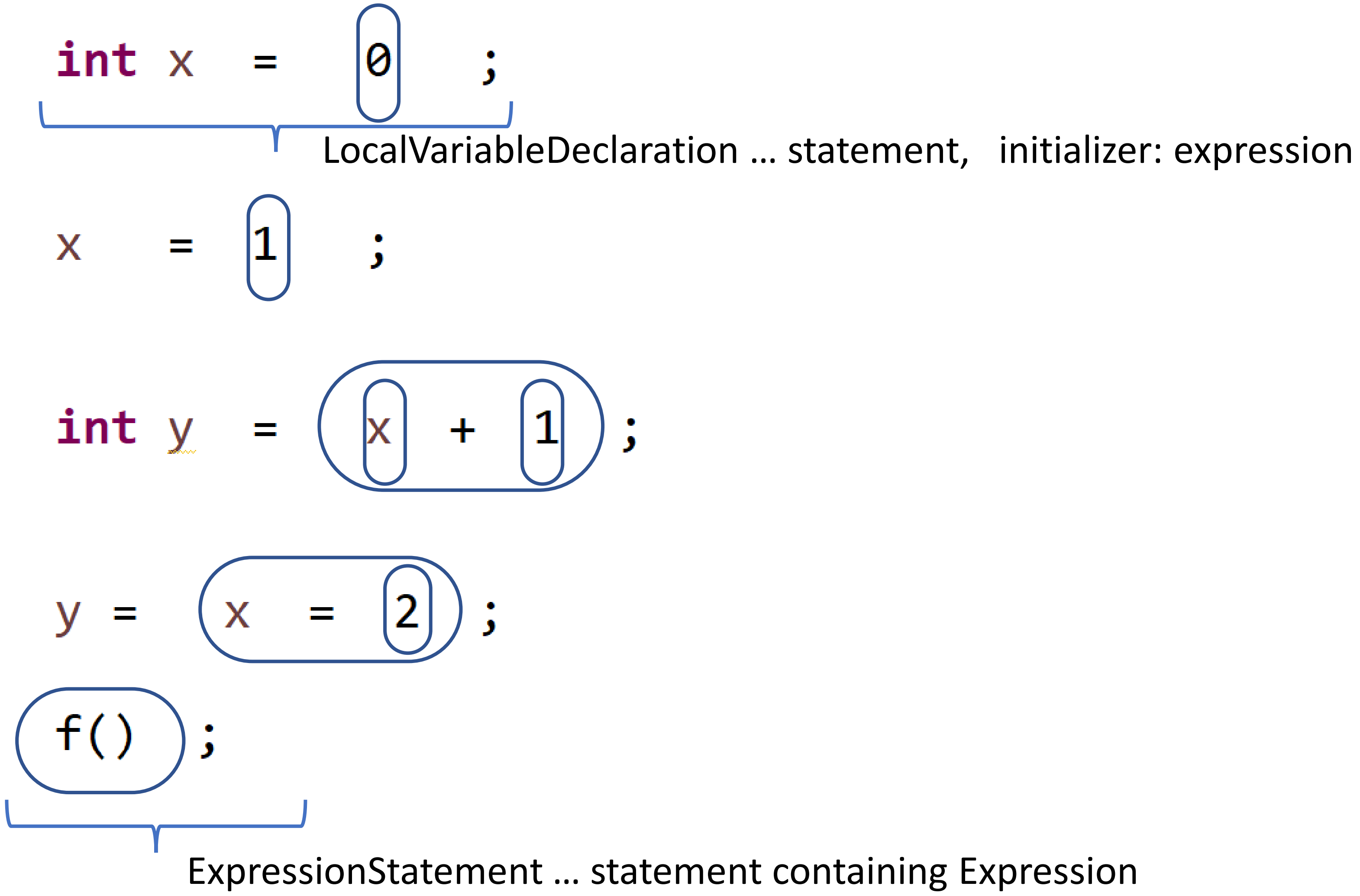
A statement has a type amoung
- Primitive type
- Pointer to class/enum/interface/..
- Array

Declaration examples

```java
public class Foo {

    private int field1;

    public Foo() {
    }

    public int getField1() {
        return field1;
    }

    public static class NestedBar {
    }

}
```



Statement / Expression Examples



```
int x = 0 ;
```

LocalVariableDeclaration … statement,   initializer: expression

```
x = 1 ;

int y = x + 1 ;

y = x = 2 ;
f() ;
```

ExpressionStatement … statement containing Expression

Symbol

Java (Source Code File)

Declaration

produce

.jar / .class library
Contains Symbol Tables
By [index] internally in bytecode
By [name] externally

Symbol

Compile/link

At Compile-Time
Import
+ resolve (ambiguous names)
+ type-check
... use in AST

load

At Runtime
Resolve by unique symbol name
+ redo type-check
... use in bytecode

Explicit Reflection Loading / Implicit Link for bytecode

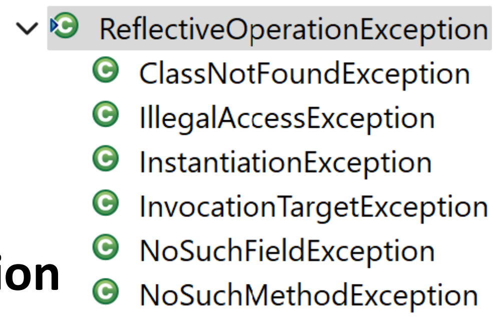
Symbol

load

At Runtime
Resolve by unique symbol name
+ redo type-check

Explicit from reflect code:
Class.forName()
/ ClassLoader.load() / .define()
/ Class.getMethod() / .getField()

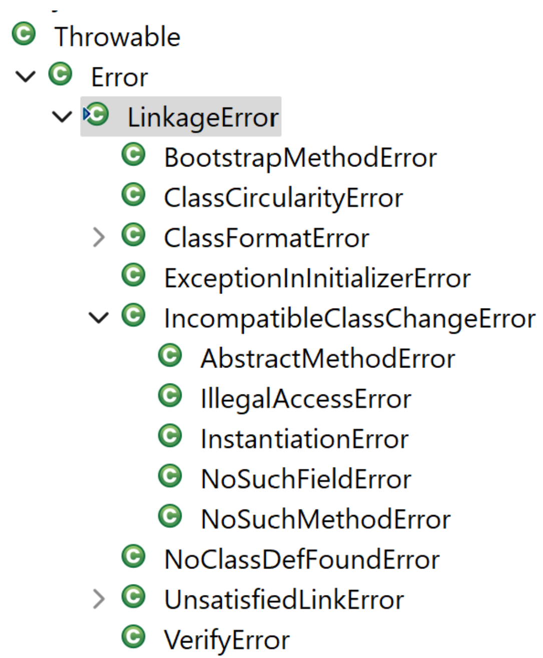
May throw ReflectiveOperation**Exception**



- ReflectiveOperationException
  - ClassNotFoundException
  - IllegalAccessException
  - InstantiationException
  - InvocationTargetException
  - NoSuchFieldException
  - NoSuchMethodException

Implicit Link For bytecode
Resolved at first usage

May throw Linkage**Error**



- Throwable
  - Error
    - LinkageError
      - BootstrapMethodError
      - ClassCircularityError
      - ClassFormatError
      - ExceptionInInitializerError
      - IncompatibleClassChangeError
        - AbstractMethodError
        - IllegalAccessError
        - InstantiationError
        - NoSuchFieldError
        - NoSuchMethodError
      - NoClassDefFoundError
      - UnsatisfiedLinkError
      - VerifyError

Type Descriptor & Symbol Name Mangling

Primitive Type => V, Z(boolean), B(byte), I(int),J(long), F(float),D(double),..

Array Type => « [» typeName

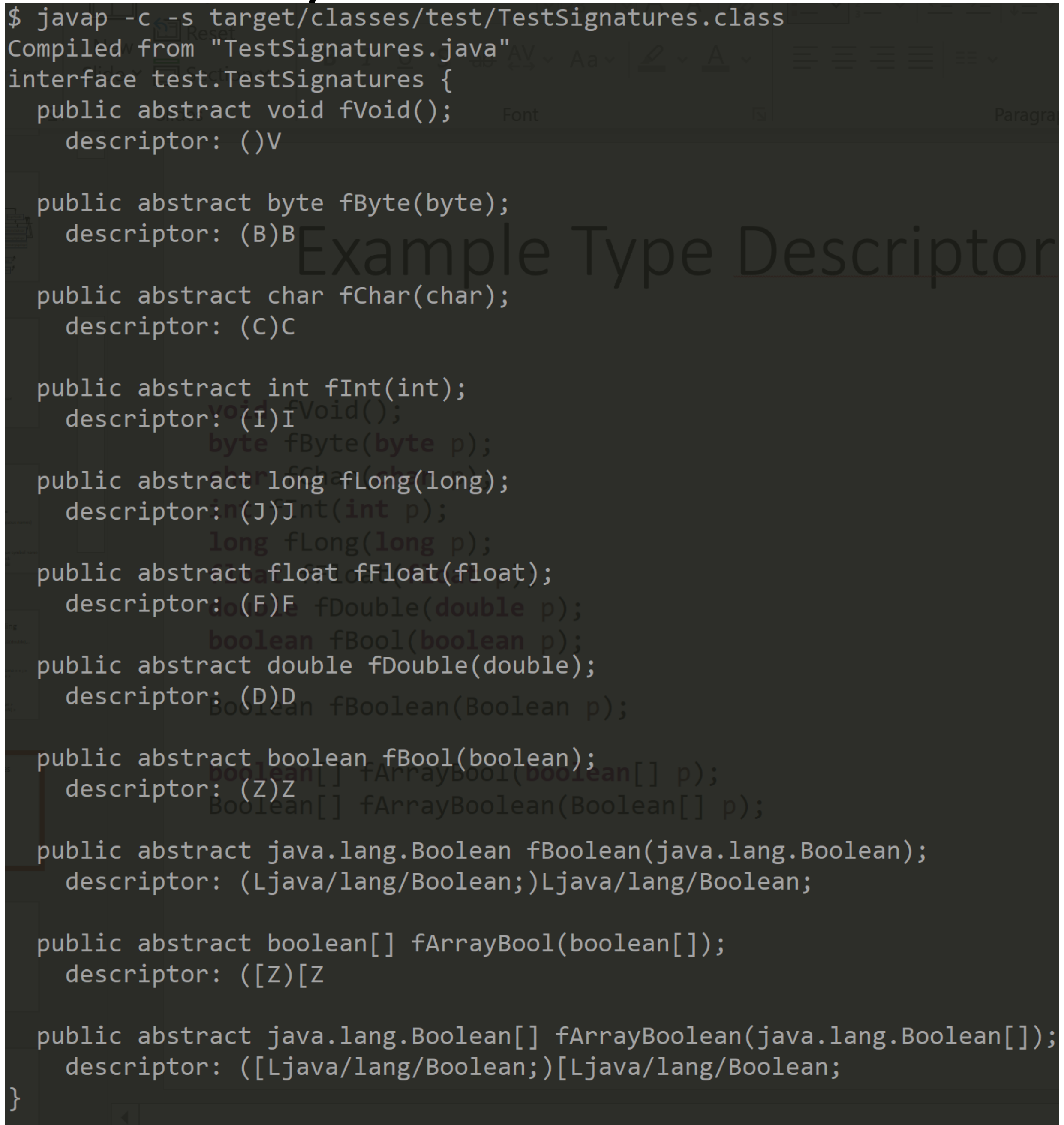
Class => type descriptor: « L » « pack »/ « subpack » /« ClassName » « ; »
symbol name : « pack » . « subpack » . « ClassName »
FQN (Fully Qualified Name)

Field => FQN « $ » « fieldName »

Method => type descriptor: « (type1 type2 ..typeN) returnType »
symbol name : «methodName(type1 type2 ..typeN) »

Example Type Descriptor & Symbol Names

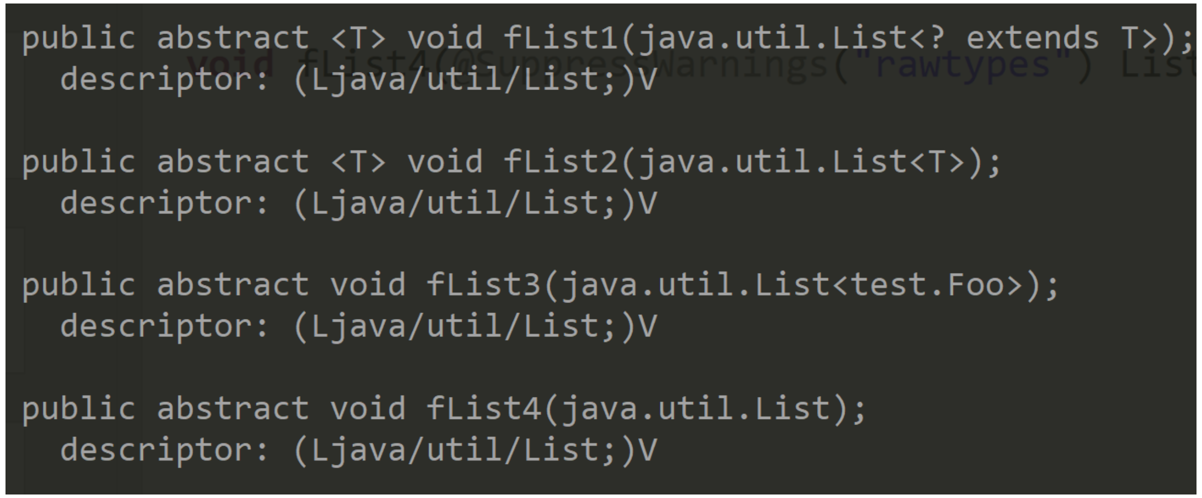
```
void fVoid();
byte fByte(byte p);
char fChar(char p);
int fInt(int p);
long fLong(long p);
float fFloat(float p);
double fDouble(double p);
boolean fBool(boolean p);

Boolean fBoolean(Boolean p);

boolean[] fArrayBool(boolean[] p);
Boolean[] fArrayBoolean(Boolean[] p);
```



```
$ javap -c -s target/classes/test/TestSignatures.class
Compiled from "TestSignatures.java"
interface test.TestSignatures {
  public abstract void fVoid();
    descriptor: ()V

  public abstract byte fByte(byte);
    descriptor: (B)B

  public abstract char fChar(char);
    descriptor: (C)C

  public abstract int fInt(int);
    descriptor: (I)I

  public abstract long fLong(long);
    descriptor: (J)J

  public abstract float fFloat(float);
    descriptor: (F)F

  public abstract double fDouble(double);
    descriptor: (D)D

  public abstract boolean fBool(boolean);
    descriptor: (Z)Z

  public abstract java.lang.Boolean fBoolean(java.lang.Boolean);
    descriptor: (Ljava/lang/Boolean;)Ljava/lang/Boolean;

  public abstract boolean[] fArrayBool(boolean[]);
    descriptor: ([Z)[Z

  public abstract java.lang.Boolean[] fArrayBoolean(java.lang.Boolean[]);
    descriptor: ([Ljava/lang/Boolean;)[Ljava/lang/Boolean;
}
```

Notice: Generic Types => same as <Object> exact Type Erased in Descriptor « Type Erasure »

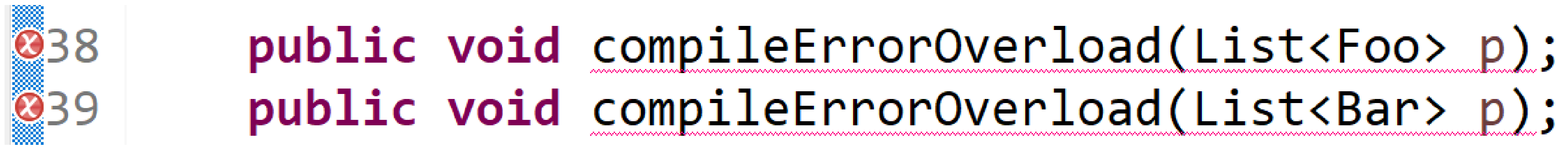
```java
<T> void fList1(List<? extends T> p1);

<T> void fList2(List<T> p1);

void fList3(List<Foo> p1);

void fList4(@SuppressWarnings("rawtypes") List p1);
```



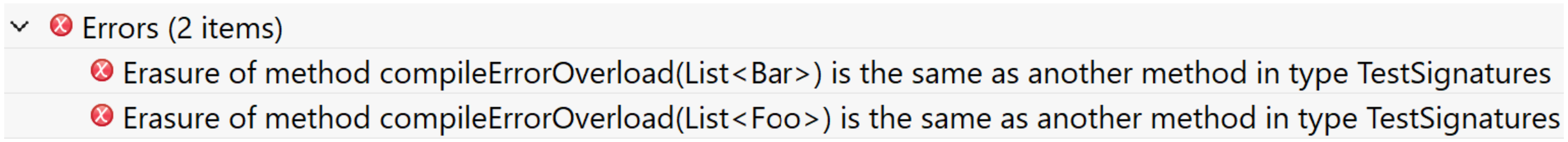
```
public abstract <T> void fList1(java.util.List<? extends T>);
  descriptor: (Ljava/util/List;)V

public abstract <T> void fList2(java.util.List<T>);
  descriptor: (Ljava/util/List;)V

public abstract void fList3(java.util.List<test.Foo>);
  descriptor: (Ljava/util/List;)V

public abstract void fList4(java.util.List);
  descriptor: (Ljava/util/List;)V
```

Overload method signatures

Can not have 2 methods overload differing only by template type



```
    public void compileErrorOverload(List<Foo> p);
    public void compileErrorOverload(List<Bar> p);
```

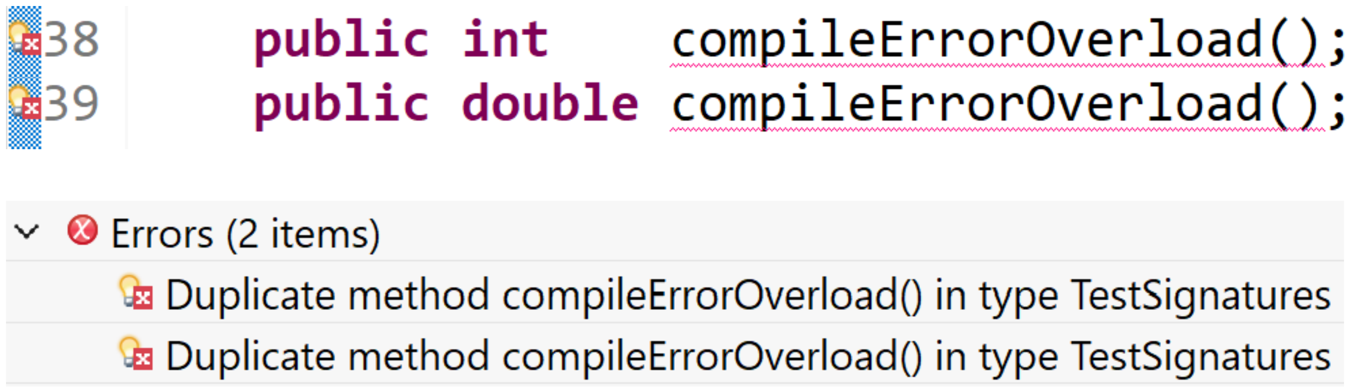


```
Errors (2 items)
    Erasure of method compileErrorOverload(List<Bar>) is the same as another method in type TestSignatures
    Erasure of method compileErrorOverload(List<Foo>) is the same as another method in type TestSignatures
```

(not an error in other langages like C++)

Notice 2 : return type not in symbol name

Can not have 2 methods overload
differing only by return type or template type



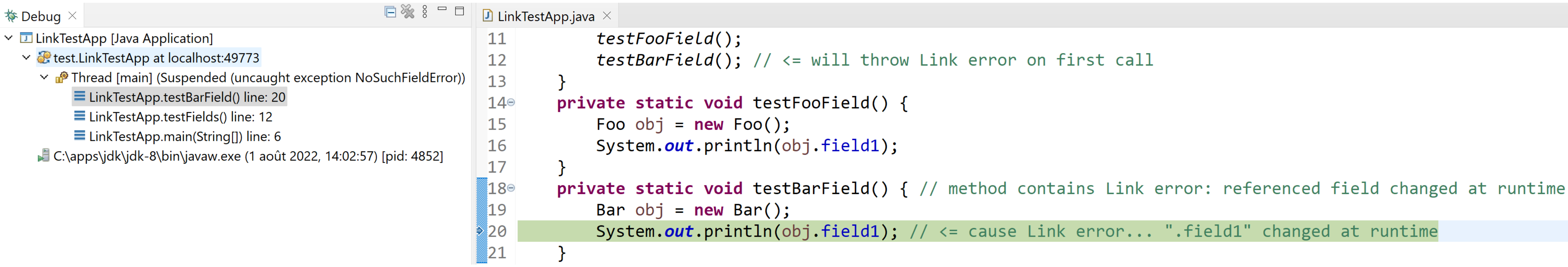
```
public int    compileErrorOverload();
public double compileErrorOverload();
```

```
Errors (2 items)
    Duplicate method compileErrorOverload() in type TestSignatures
    Duplicate method compileErrorOverload() in type TestSignatures
```

Compiled OK … BUT change in CLASSPATH => LinkError

```java
public class Bar {

    public int field1;
```

```java
// previously compiled with..
/*
public int field1;
*/
// changed at runtime with..
public long field1;
```



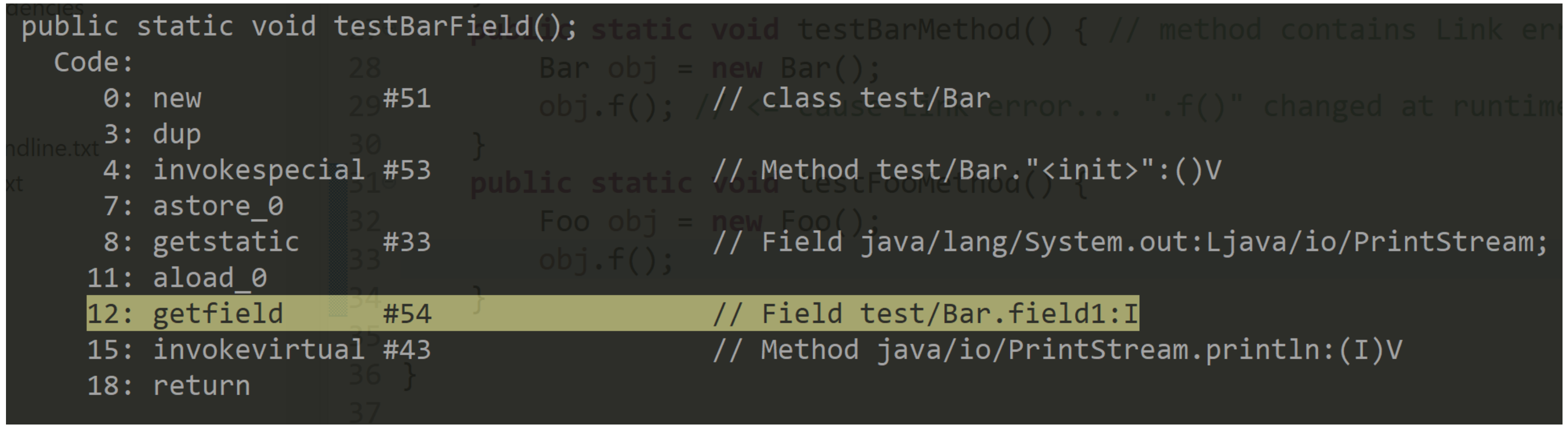
```
Debug
LinkTestApp [Java Application]
  test.LinkTestApp at localhost:49773
    Thread [main] (Suspended (uncaught exception NoSuchFieldError))
      LinkTestApp.testBarField() line: 20
      LinkTestApp.testFields() line: 12
      LinkTestApp.main(String[]) line: 6
  C:\apps\jdk\jdk-8\bin\javaw.exe (1 août 2022, 14:02:57) [pid: 4852]
```

```java
LinkTestApp.java
11          testFooField();
12          testBarField(); // <= will throw Link error on first call
13      }
14      private static void testFooField() {
15          Foo obj = new Foo();
16          System.out.println(obj.field1);
17      }
18      private static void testBarField() { // method contains Link error: referenced field changed at runtime
19          Bar obj = new Bar();
20          System.out.println(obj.field1); // <= cause Link error... ".field1" changed at runtime
21      }
```

Program start OK
Error method partially execute .. OK !
but fail on first bytecode Link error

```
0
Exception in thread "main" java.lang.NoSuchFieldError: field1
	at test.LinkTestApp.testBarField(LinkTestApp.java:20)
	at test.LinkTestApp.testFields(LinkTestApp.java:12)
	at test.LinkTestApp.main(LinkTestApp.java:6)
```

Internally … « getfield » => « _fast_*getfield »



```
public static void testBarField();
  Code:
     0: new           #51                 // class test/Bar
     3: dup
     4: invokespecial #53                 // Method test/Bar."<init>":()V
     7: astore_0
     8: getstatic     #33                 // Field java/lang/System.out:Ljava/io/PrintStream;
    11: aload_0
    12: getfield      #54                 // Field test/Bar.field1:I
    15: invokevirtual #43                 // Method java/io/PrintStream.println:(I)V
    18: return
```

On First use….

Resolve Bytecode « getfield #fieldIdx»

Load + type-check

Class c = .. Class.forName(« test.Bar »)
f = c.getField(« field1 »);
assert f.getType().equals(int.class);

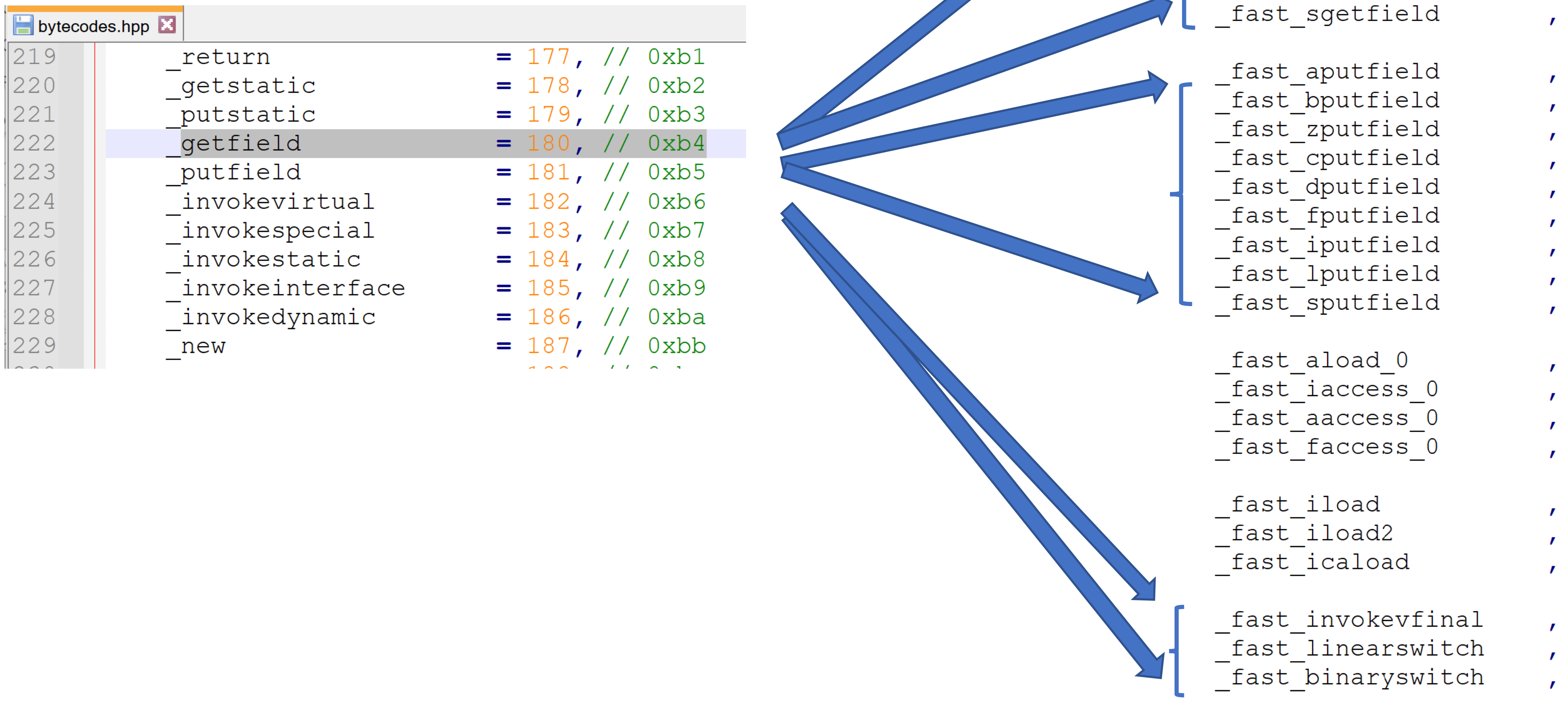
Bytecode instruction
« getfield #idx»
HOT REPLACED BY internal
« **_fast_getfield offset**»

Code replaced (?) or re-executed
with « throw new …Error() »

«fast_getfield» on next uses

Cf OpenJdk ..
src/hotspot/share/interpreter/bytecodes.hpp

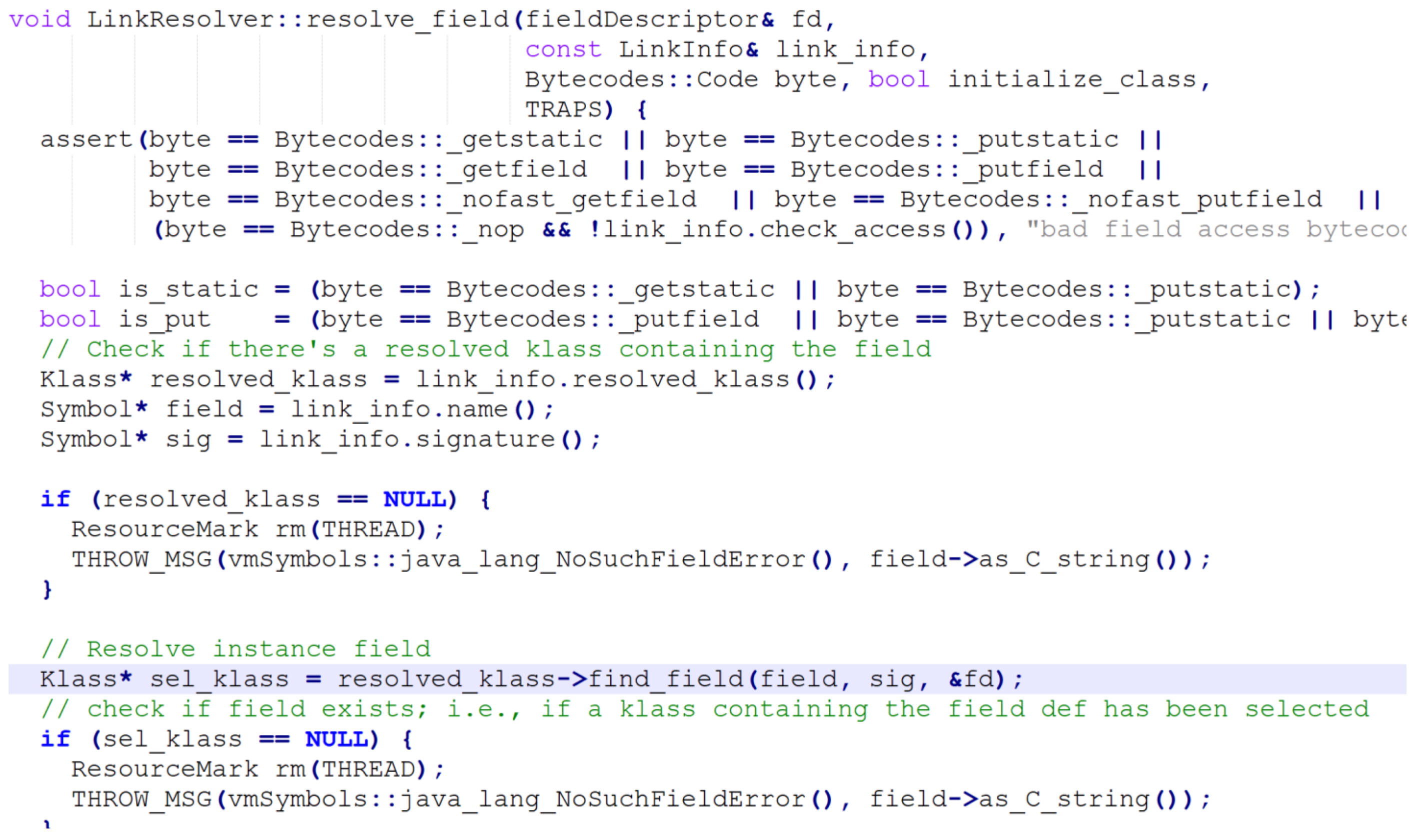
bytecodes + Internal Reserved fast_* bytecodes



```
    _return              = 177, // 0xb1
    _getstatic           = 178, // 0xb2
    _putstatic           = 179, // 0xb3
    _getfield            = 180, // 0xb4
    _putfield            = 181, // 0xb5
    _invokevirtual       = 182, // 0xb6
    _invokespecial       = 183, // 0xb7
    _invokestatic        = 184, // 0xb8
    _invokeinterface     = 185, // 0xb9
    _invokedynamic       = 186, // 0xba
    _new                 = 187, // 0xbb
```

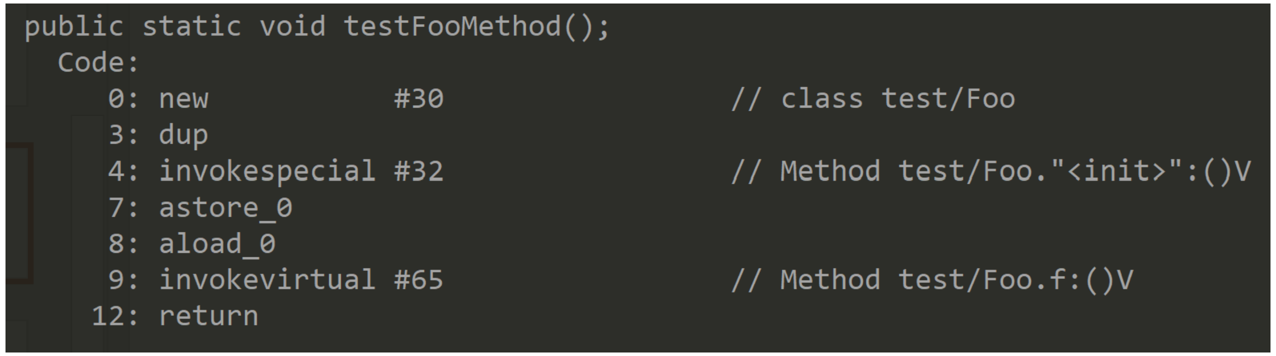
```
// JVM bytecodes
_fast_agetfield        =
_fast_bgetfield        ,
_fast_cgetfield        ,
_fast_dgetfield        ,
_fast_fgetfield        ,
_fast_igetfield        ,
_fast_lgetfield        ,
_fast_sgetfield        ,

_fast_aputfield        ,
_fast_bputfield        ,
_fast_zputfield        ,
_fast_cputfield        ,
_fast_dputfield        ,
_fast_fputfield        ,
_fast_iputfield        ,
_fast_lputfield        ,
_fast_sputfield        ,

_fast_aload_0          ,
_fast_iaccess_0        ,
_fast_aaccess_0        ,
_fast_faccess_0        ,

_fast_iload            ,
_fast_iload2           ,
_fast_icaload          ,

_fast_invokevfinal     ,
_fast_linearswitch     ,
_fast_binaryswitch     ,
```

linkResolver.cpp + cpCache.cpp + rewriter.cpp ..



```
void LinkResolver::resolve_field(fieldDescriptor& fd,
                                 const LinkInfo& link_info,
                                 Bytecodes::Code byte, bool initialize_class,
                                 TRAPS) {
  assert(byte == Bytecodes::_getstatic || byte == Bytecodes::_putstatic ||
         byte == Bytecodes::_getfield  || byte == Bytecodes::_putfield  ||
         byte == Bytecodes::_nofast_getfield  || byte == Bytecodes::_nofast_putfield  ||
         (byte == Bytecodes::_nop && !link_info.check_access()), "bad field access byteco

  bool is_static = (byte == Bytecodes::_getstatic || byte == Bytecodes::_putstatic);
  bool is_put    = (byte == Bytecodes::_putfield  || byte == Bytecodes::_putstatic || byte
  // Check if there's a resolved klass containing the field
  Klass* resolved_klass = link_info.resolved_klass();
  Symbol* field = link_info.name();
  Symbol* sig = link_info.signature();

  if (resolved_klass == NULL) {
    ResourceMark rm(THREAD);
    THROW_MSG(vmSymbols::java_lang_NoSuchFieldError(), field->as_C_string());
  }

  // Resolve instance field
  Klass* sel_klass = resolved_klass->find_field(field, sig, &fd);
  // check if field exists; i.e., if a klass containing the field def has been selected
  if (sel_klass == NULL) {
    ResourceMark rm(THREAD);
    THROW_MSG(vmSymbols::java_lang_NoSuchFieldError(), field->as_C_string());
  }
```

Idem « invoke* » => « fast_invoke* »



```
public static void testFooMethod();
  Code:
     0: new           #30                 // class test/Foo
     3: dup
     4: invokespecial #32                 // Method test/Foo."<init>":()V
     7: astore_0
     8: aload_0
     9: invokevirtual #65                 // Method test/Foo.f:()V
    12: return
```

On First use….

Resolve Bytecode « invoke* #methIdx»

Load + type-check

Class c = .. Class.forName(« test.Bar »)
m = c.getMethod(« field1(type1..typeN »);
assert m.getType().equals(…);

Bytecode instruction
« invoke* #idx»
HOT REPLACED BY internal
« **_fast_invoke* offset**»

Code replaced (?) or re-executed
with « throw new …Error() »

«fast_invoke*» on next uses

Different « invoke* » : {static|special|virtual|interface|dynamic}

static => … to call fixed (known) function, and update stack

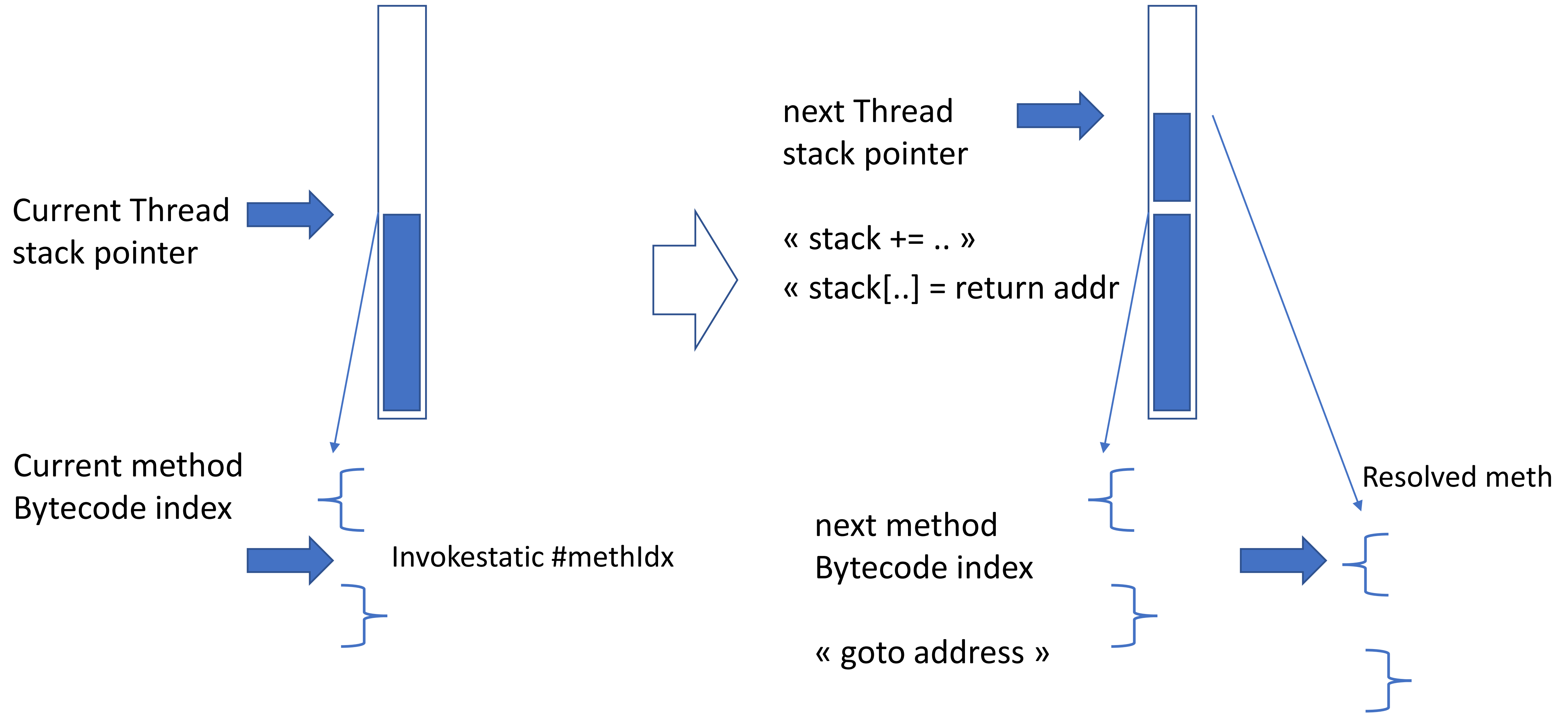
special => … idem … call fixed (known) function, and update stack
after « new »: « <init> » method, or « super() » method

virtual => … need array access to object class « virtual table », to determine exact method

interface => … need lookup interfaces table … then virtual table, to determine method

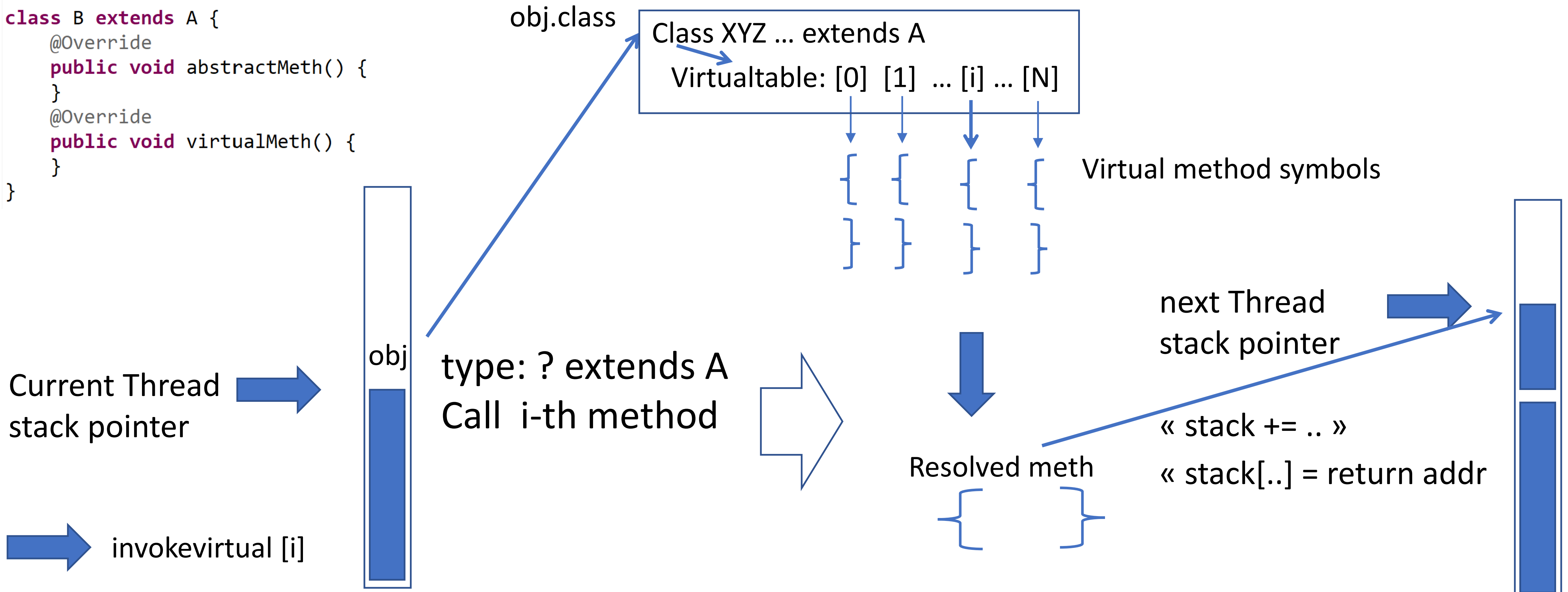
dynamic => … internal for JRE, allowing type evolution

invokestatic


Current Thread stack pointer
Current method Bytecode index
Invokestatic #methIdx
next Thread stack pointer
« stack += .. »
« stack[..] = return addr
next method Bytecode index
« goto address »
Resolved meth

invokevirtual

```java
public abstract class A {
    public abstract void abstractMeth();
    public void virtualMeth() {
        // maybe overriden
    }
}

class B extends A {
    @Override
    public void abstractMeth() {
    }
    @Override
    public void virtualMeth() {
    }
}
```



invokevirtual .. slower than invokestatic

Invokestatic …. O(1 call)

Invokevirtual …. O(**1 array access** + **push 1 extra param** « this » + 1 call)

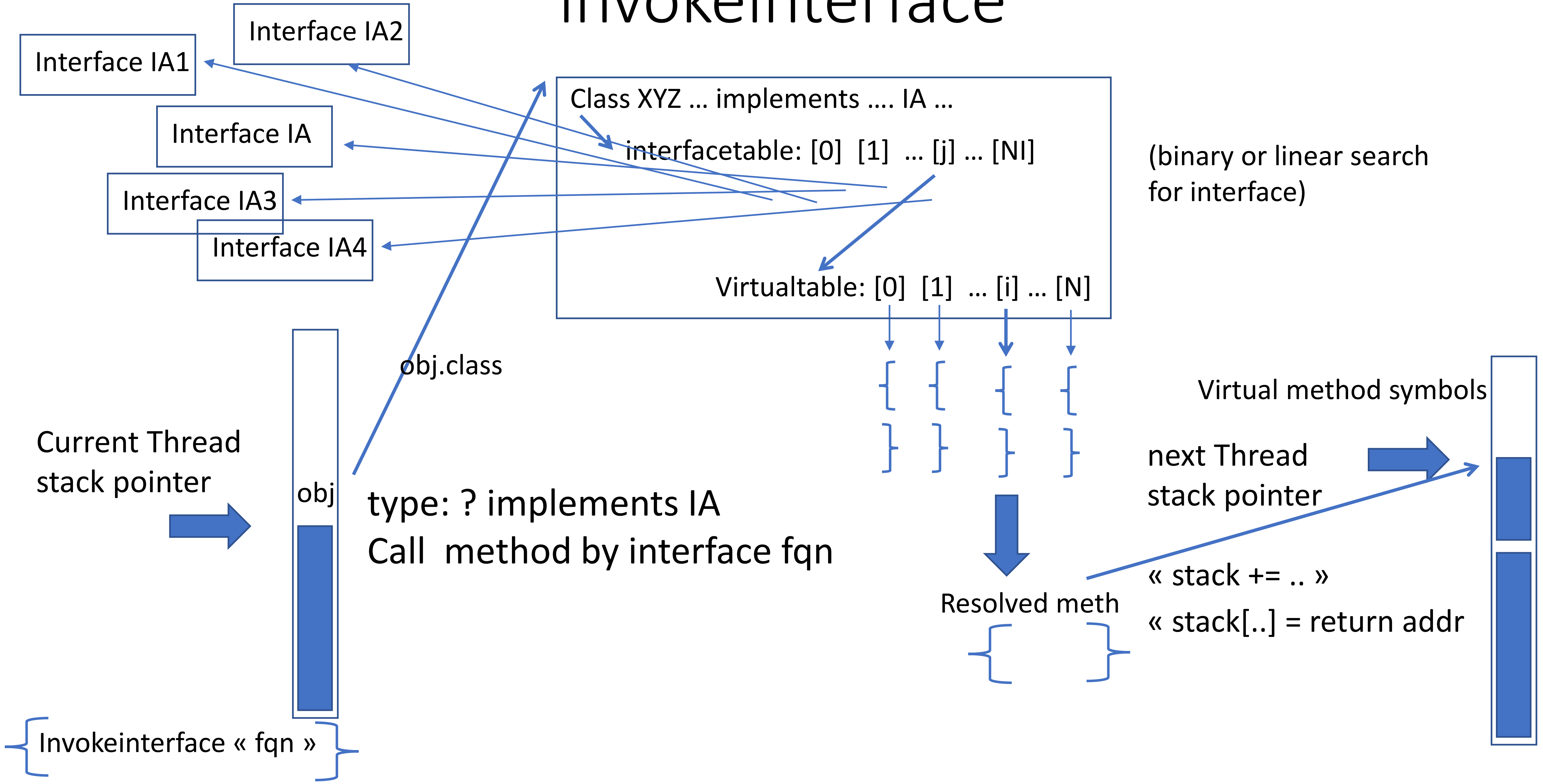
invokeinterface

```
public interface IA {
    public void meth();
    public default void defaultMeth() {
        // maybe overriden
    }
}

class B extends Foo implements IA1, IA2, IA3,   IA,   IA4, IA5 {
//                                              /\
//                                          j-th position
    @Override
    public void meth() {
    }
    @Override
    public void defaultMeth() {
    }
}
```

```
IA anyObj = new B();
anyObj.meth();  // <= invokeinterface
```

Performance Problem :
  no relation between interface « IA » and extends class (here: « Foo »)
.. can not use virtualtable directly

invokeinterface


Interface IA2
Interface IA1
Interface IA
Interface IA3
Interface IA4
Class XYZ … implements …. IA …
interfacetable: [0] [1] … [j] … [NI]
Virtualtable: [0] [1] … [i] … [N]
(binary or linear search for interface)
obj.class
Virtual method symbols
Current Thread stack pointer
obj
next Thread stack pointer
type: ? implements IA
Call method by interface fqn
« stack += .. »
Resolved meth
« stack[..] = return addr
Invokeinterface « fqn »

invokeinterface .. slower than invokevirtual

Invokestatic …. O(1 call)

Invokevirtual …. O(1 array access + push 1 extra param « this » + 1 call)

Invokeinterface …. O(1 pointer access
+ J linear OR log(J) binary search interface tables
+ 1 virtual table array access
+ push 1 extra param « this » + 1 call)

Remark: abstract class vs interface

For performance (with millions of calls…):

1/ Prefer declare virtual method in abstract class
Rather than method in interface

2/ do not use too many interfaces

3/ sort « implements » interfaces by most frequent usage first
(? .. If linear search)

Example: cf in JDK …
public abstract class InputStream { public abstract int read(); }
public abstract class OutputStream { public abstract void write(int data); }

Questions ?