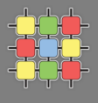
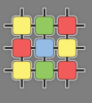


Architecture of software systems

Course 3: Virtual machines, byte-code, de-compiler, code protection, class loaders, reflection

David Šišlák
david.sislak@fel.cvut.cz

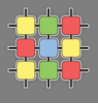
Introduction - Programming languages



- from **source-execution flow** perspective:
  - scripting – no pre-compilation (e.g. Perl)
  - with compilation
    - into native-code (e.g. C)
    - into byte-code (e.g. Java, .NET)
- from **execution** perspective:
  - machine-code execution
  - interpreted (script or byte-code)
  - hybrid – both interpreted and machine-code execution

- BUT
  - ahead-of-time (AOT) compiler – byte-code into native-code compiler
    for JAVA, .NET and others

Introduction - Programming languages



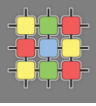
- Interpreted/byte-code vs. machine-code
  - (+) platform independence – architecture (RISC/CISC, bits), OS
  - (+) reflection – observe, modify own structure at run-time
  - (+) dynamic typing – at run-time (JAVA has no dynamic typing)
  - (+) small size
  - (+) dynamic scoping – (Perl)
  - (-) slower execution – interpreted mode, JIT latencies

```
var x := 5;
var y := "37";
var z := x + y;
```

```
$x = 0;
sub f { return $x; }
sub g { my $x = 1; return f(); }
print g()."\n";
```

```
$x = 0;
sub f { return $x; }
sub g { local $x = 1; return f(); }
print g()."\n";
```

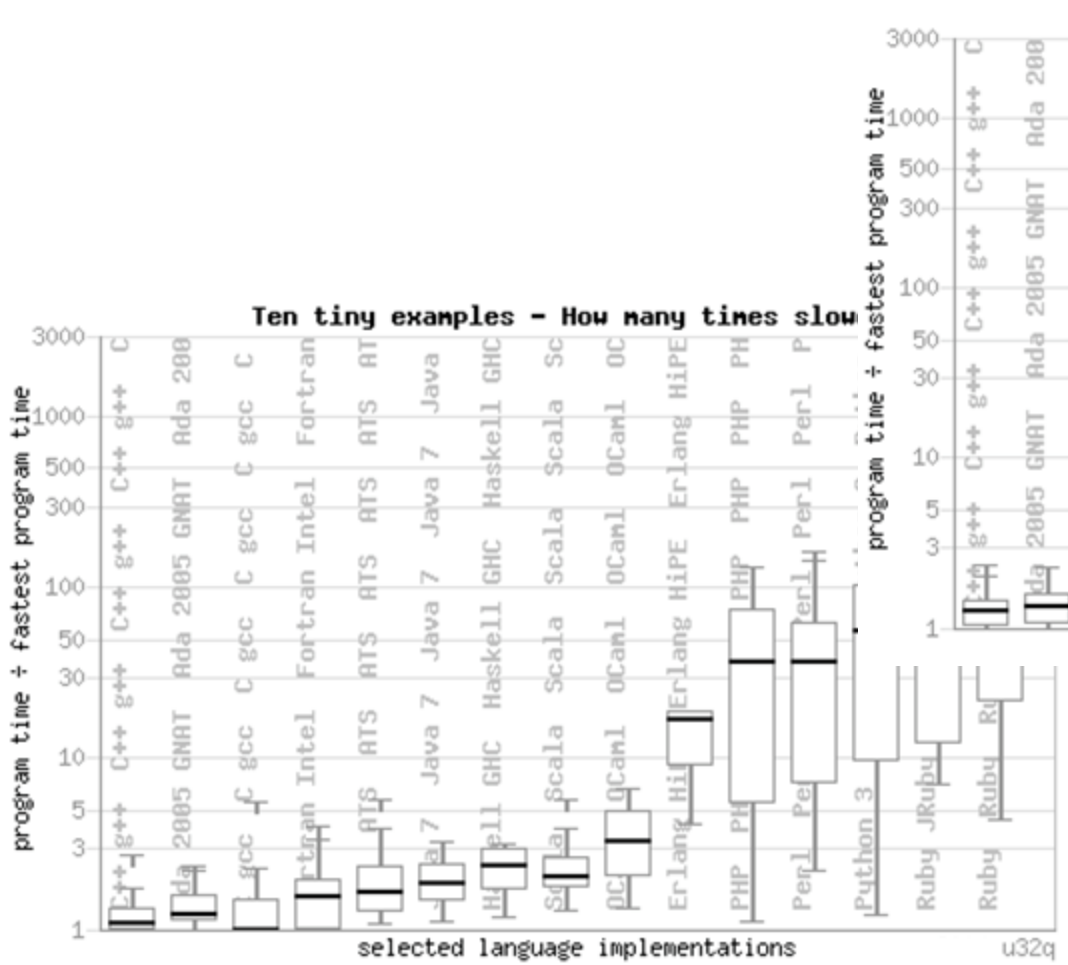
Introduction - Programming languages

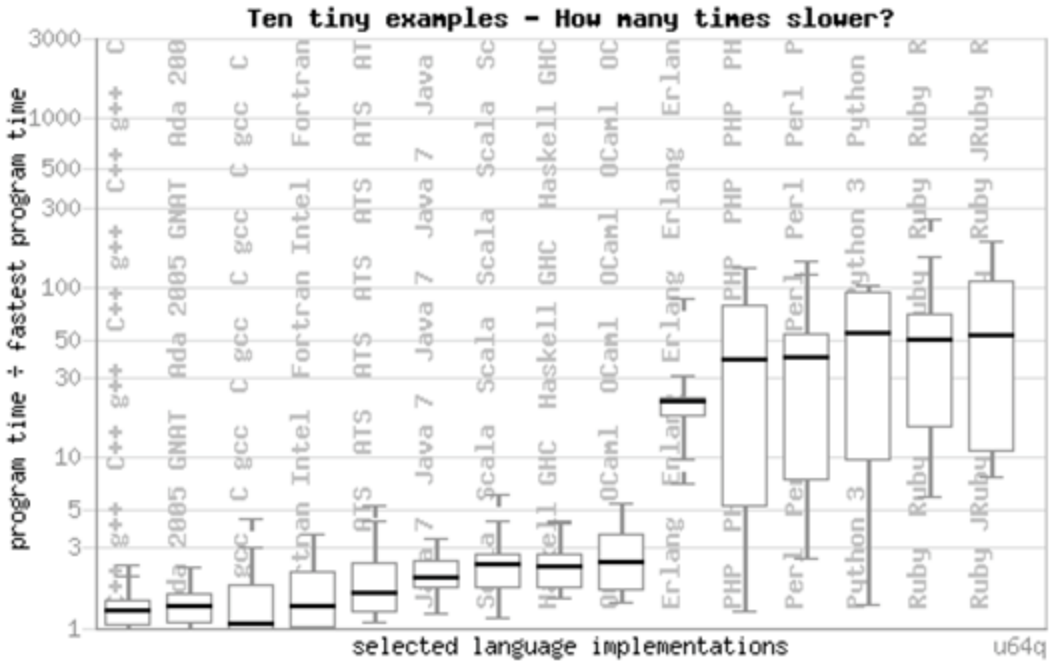


» The Computer Language Benchmarks Game

(source: http://shootout.alioth.debian.org/)

» various langs, 13 different domains (binary-trees, etc.)





Introduction - JAVA

- first release 1995 by Sun Microsystems (later Oracle)
- many different implementations (GNU, IBM (J9), etc.)

- standard source-execution model:
  - code: source (.java)
  - compile: byte-code (.class)
  - run-time: Java Virtual Machine (JVM) in Java Runtime Environment (JRE)
    - interpreter: byte-code
    - Just-in-Time (JIT) compiler: native-code

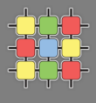
Java virtual machine and byte-code

- Why understand byte-code?
  - helps the same way as assembler helps C programmer
  - tuning your programs
- **stack-oriented** - stack machine model for passing parameters

$$(2 + 3) \times 11 + 1$$

Input	2	3	add	11	mul	1	add
Stack	2	3 2	5	11 5	55	1 55	56

Java virtual machine and byte-code

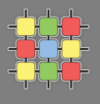


- **opcode** (1 byte + various parameters):
  - load and store (aload_0, istore, aconst_null, …)
  - arithmetic and logic (ladd, fcmpl, …)
  - type conversion (i2b, d2i, …)
  - object manipulation (new, putfield, getfield, …)
  - stack management (swap, dup2, …)
  - control transfer (ifeq, goto, …)
  - method invocation (invokespecial, areturn, …)
  - exceptions and monitor concurrency (athrow, monitorenter, …)
- prefix/suffix – i, l, s, b, c, f, d and a (reference)
- variables as registers – e.g. istore_1 (variable 0 is **this** for instance method)

```
add eax, edx
mov ecx, eax
```

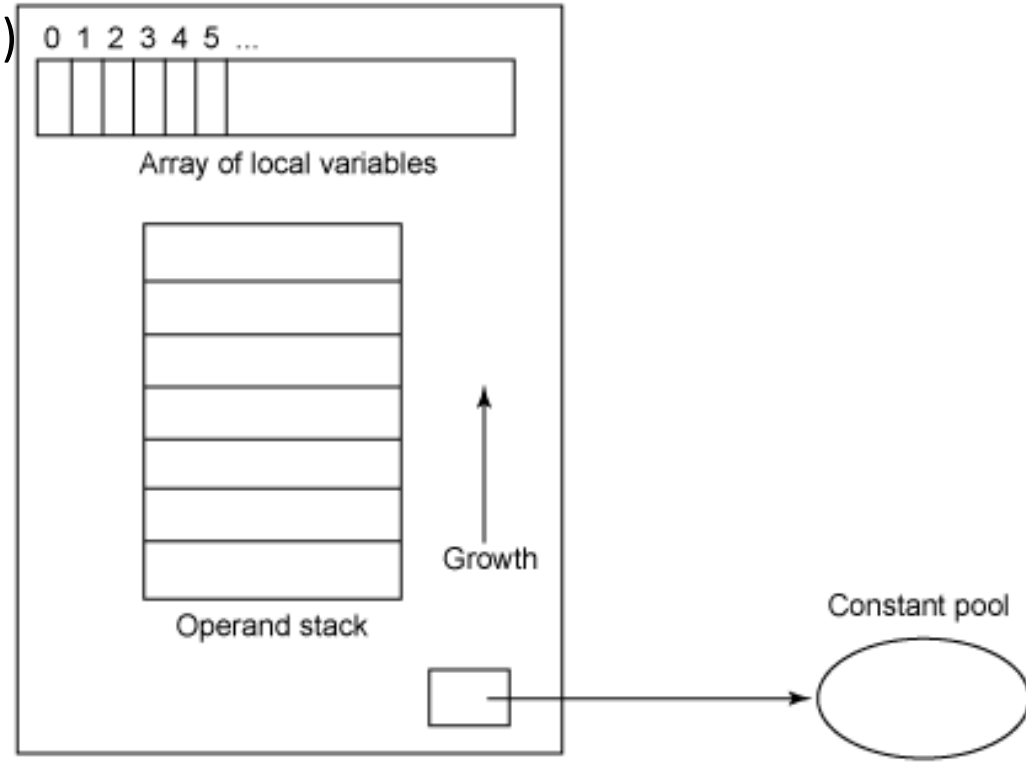
VS.

```
0 iload_1
1 iload_2
2 iadd
3 istore_3
```

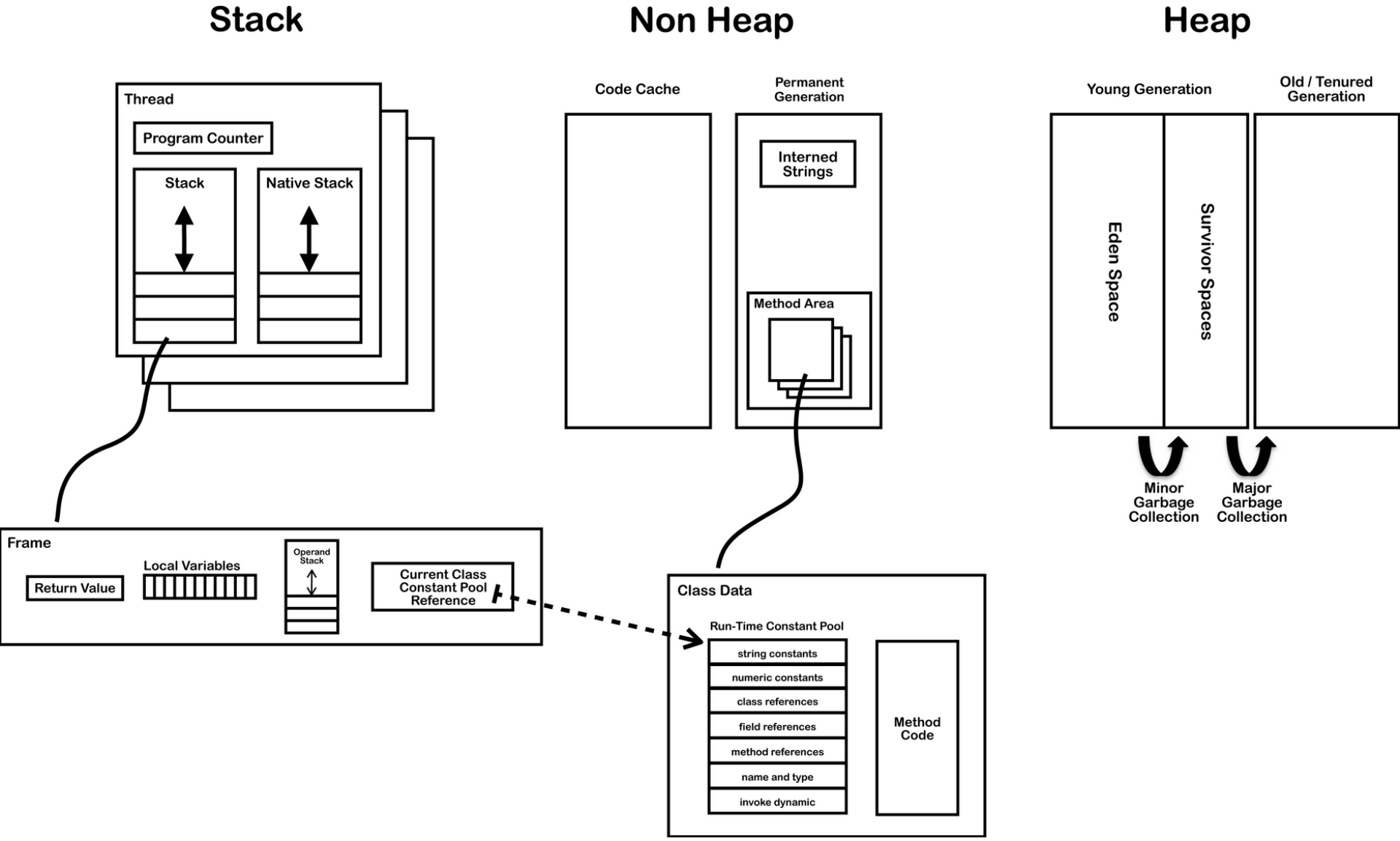


- **frame**
  - each thread has stack with frames (outside of heap, fixed length)

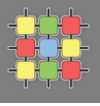
    StackOverflowError vs. OutOfMemoryError
  - frame is **created** each time method is invoked (after it is **destroyed**)
  - frame **size** determined at compile-time (in class file)
  - variables (long, double in two)
    - {this} – *instance call only!*
    - {method parameters}
    - {local variables}
  - operand stack (any type)
    - LIFO
  - reference to run-time constant pool (class def)
  - method + class is associated



Java virtual machine – memory organisation

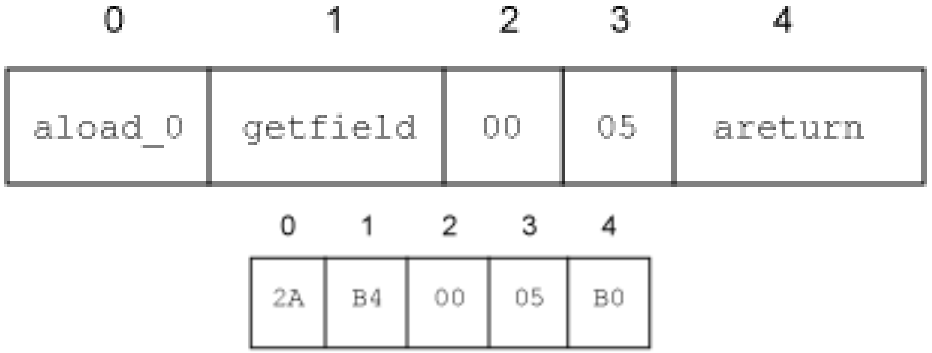


Java virtual machine and byte-code

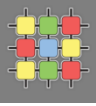


```
public String employeeName()
{
return name;
}

Method java.lang.String employeeName()
0 aload_0
1 getfield #5 <Field java.lang.String name>
4 areturn
```

- getfield
  - takes 1 ref from stack
  - build an index into runtime pool of class instance by reference **this**
- areturn
  - takes 1 ref from stack
  - push onto the stack of calling method



Java virtual machine and byte-code



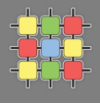
```
public Employee(String strName, int num)
{
name = strName;
idNumber = num;
storeData(strName, num);
}

Method Employee(java.lang.String,int)
0 aload_0
1 invokespecial #3 <Method java.lang.Object()>
4 aload_0
5 aload_1
6 putfield #5 <Field java.lang.String name>
9 aload_0
10 iload_2
11 putfield #4 <Field int idNumber>
14 aload_0
15 aload_1
16 iload_2
17 invokespecial #6 <Method void storeData(java.lang.String, int)>
20 return
```

EXAMPLE 1

(see different size and speed issue)

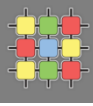
- javac -g xxx (see class file structure)
- *java xxx*
- javap -c -l -s xxx

Source code - Test.java



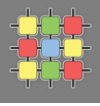
```
public final class Test {

    private int intArr[];


    private Test() {
        intArr = new int[10];
    }

    public synchronized int top1() {
        return intArr[0];
    }

    public int top2() {
        synchronized (this) {
            return intArr[0];
        }
    }

    public static void main(String[] args) {
        final Test test = new Test();
        test.top1();
        test.top2();
    }
}
```

Byte code – raw form - Test.class – “javac –g Test.java”



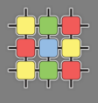
```
00000000h: CA FE BA BE 00 00 00 33 00 26 0A 00 07 00 1F 09 ; Êþº¾...3.&......
00000010h: 00 03 00 20 07 00 21 0A 00 03 00 1F 0A 00 03 00 ; ... ..!.........
00000020h: 22 0A 00 03 00 23 07 00 24 01 00 06 69 6E 74 41 ; "....#..$...intA
00000030h: 72 72 01 00 02 5B 49 01 00 06 3C 69 6E 69 74 3E ; rr...[I...<init>
00000040h: 01 00 03 28 29 56 01 00 04 43 6F 64 65 01 00 0F ; ...()V...Code...
00000050h: 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C 65 01 ; LineNumberTable.
00000060h: 00 12 4C 6F 63 61 6C 56 61 72 69 61 62 6C 65 54 ; ..LocalVariableT
00000070h: 61 62 6C 65 01 00 04 74 68 69 73 01 00 06 4C 54 ; able...this...LT
00000080h: 65 73 74 3B 01 00 04 74 6F 70 31 01 00 03 28 29 ; est;...top1...()
00000090h: 49 01 00 04 74 6F 70 32 01 00 0D 53 74 61 63 6B ; I...top2...Stack
000000a0h: 4D 61 70 54 61 62 6C 65 07 00 21 07 00 24 07 00 ; MapTable..!..$..
000000b0h: 25 01 00 04 6D 61 69 6E 01 00 16 28 5B 4C 6A 61 ; %...main...([Lja
000000c0h: 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 29 ; va/lang/String;)
000000d0h: 56 01 00 04 61 72 67 73 01 00 13 5B 4C 6A 61 76 ; V...args...[Ljav
000000e0h: 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 01 00 ; a/lang/String;..
000000f0h: 04 74 65 73 74 01 00 0A 53 6F 75 72 63 65 46 69 ; .test...SourceFi
00000100h: 6C 65 01 00 09 54 65 73 74 2E 6A 61 76 61 0C 00 ; le...Test.java..
00000110h: 0A 00 0B 0C 00 08 00 09 01 00 04 54 65 73 74 0C ; ...........Test.
00000120h: 00 11 00 12 0C 00 13 00 12 01 00 10 6A 61 76 61 ; ............java
00000130h: 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01 00 13 6A ; /lang/Object...j
00000140h: 61 76 61 2F 6C 61 6E 67 2F 54 68 72 6F 77 61 62 ; ava/lang/Throwab
00000150h: 6C 65 00 31 00 03 00 07 00 00 00 01 00 02 00 08 ; le.1............
00000160h: 00 09 00 00 00 04 00 02 00 0A 00 0B 00 01 00 0C ; ................
00000170h: 00 00 00 3F 00 02 00 01 00 00 00 0D 2A B7 00 01 ; ...?........*·..
00000180h: 2A 10 0A BC 0A B5 00 02 B1 00 00 00 02 00 0D 00 ; *..¼.µ..±.......
00000190h: 00 00 0E 00 03 00 00 00 06 00 04 00 07 00 0C 00 ; ................
000001a0h: 08 00 0E 00 00 00 0C 00 01 00 00 00 0D 00 0F 00 ; ................
000001b0h: 10 00 00 00 21 00 11 00 12 00 01 00 0C 00 00 00 ; ....!...........
000001c0h: 31 00 02 00 01 00 00 00 07 2A B4 00 02 03 2E AC ; 1........*´....¬
000001d0h: 00 00 00 02 00 0D 00 00 00 06 00 01 00 00 00 0B ; ................
000001e0h: 00 0E 00 00 00 0C 00 01 00 00 00 07 00 0F 00 10 ; ................
000001f0h: 00 00 00 01 00 13 00 12 00 01 00 0C 00 00 00 6C ; ...............l
00000200h: 00 02 00 03 00 00 00 12 2A 59 4C C2 2A B4 00 02 ; ........*YLÂ*´..
00000210h: 03 2E 2B C3 AC 4D 2B C3 2C BF 00 02 00 04 00 0C ; ..+Ã¬M+Ã,¿......
00000220h: 00 0D 00 00 00 0D 00 10 00 0D 00 00 00 03 00 0D ; ................
00000230h: 00 00 00 0E 00 03 00 00 00 0F 00 04 00 10 00 0D ; ................
00000240h: 00 11 00 0E 00 00 00 0C 00 01 00 00 00 12 00 0F ; ................
00000250h: 00 10 00 00 00 14 00 00 00 12 00 01 FF 00 0D 00 ; ............ÿ...
00000260h: 02 07 00 15 07 00 16 00 01 07 00 17 00 09 00 18 ; ................
00000270h: 00 19 00 01 00 0C 00 00 00 53 00 02 00 02 00 00 ; .........S......
```

Byte code – raw form



```
ClassFile {
    u4              magic;
    u2              minor_version;
    u2              major_version;
    u2              constant_pool_count;
    cp_info         contant_pool[constant_pool_count - 1];
    u2              access_flags;
    u2              this_class;
    u2              super_class;
    u2              interfaces_count;
    u2              interfaces[interfaces_count];
    u2              fields_count;
    field_info      fields[fields_count];
    u2              methods_count;
    method_info     methods[methods_count];
    u2              attributes_count;
    attribute_info  attributes[attributes_count];
}
```

Byte code – opcode form – “javap -c -l -s Test”



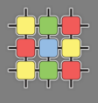
```
public final class Test {
  public static void main(java.lang.String[]);
    Signature: ([Ljava/lang/String;)V
    Code:
       0: new           #3                  // class Test
       3: dup
       4: invokespecial #4                  // Method "<init>":()V
       7: astore_1
       8: aload_1
       9: invokevirtual #5                  // Method top1:()I
      12: pop
      13: aload_1
      14: invokevirtual #6                  // Method top2:()I
      17: pop
      18: return
    LineNumberTable:
      line 22: 0
      line 23: 8
      line 24: 13
      line 25: 18
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0      19     0  args   [Ljava/lang/String;
          8      11     1  test   LTest;
```

final Test test = new Test();

test.top1();

test.top2();

Byte code – opcode form – “javap -c -l -s Test”



```
public synchronized int top1();
  Signature: ()I
  Code:
     0: aload_0
     1: getfield      #2                  // Field intArr:[I
     4: iconst_0
     5: iaload
     6: ireturn
  LineNumberTable:
    line 11: 0
  LocalVariableTable:
    Start  Length  Slot  Name   Signature
        0       7     0  this   LTest;
```

```
public int top2();
  Signature: ()I
  Code:
     0: aload_0                                    synchronized (this) {
     1: dup
     2: astore_1
     3: monitorenter                                                    ()V
     4: aload_0                                    return intArr[0];
     5: getfield      #2                                  // Field intArr:[I
     8: iconst_0
     9: iaload
    10: aload_1
    11: monitorexit
    12: ireturn
    13: astore_2
    14: aload_1
    15: monitorexit
    16: aload_2
    17: athrow
  Exception table:
     from    to  target type
         4    12    13   any
        13    16    13   any
  LineNumberTable:
    line 15: 0
    line 16: 4
    line 17: 13
  LocalVariableTable:
    Start  Length  Slot  Name   Signature
        0      18     0  this   LTest;
```

Java virtual machine and byte-code

- Summary of Example1
  - byte-code is not better than your source code
    - invariants in loop are not removed
  - no optimizations like
    - loop unrolling
    - algebraic simplification
    - strength reduction

- debug information (affect the size of byte-code)
  - -g
  - -g:none

Java virtual machine and byte-code

- method area shared among all threads
  - class definitions
    - run-time constant pool
    - field and method data
    - byte-code for methods and constructors
    - initialization methods (**<clinit>**, **<init>**)

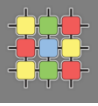
- native method stacks
  - implementation of native methods

EXAMPLE 2

(more complex example + decompiler)

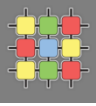
- e.g. jad
- javap -c -l -s -private Test
- jad -p Test.class
- try: -g:none

Source code – Test.java



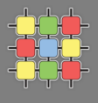
```
public final class Test {
   private static int j;

   private static final void doSomething() {
     outer:
        for (int i = 2; i < 1000; i++) {
          for (j = 2; j < i; j++) {
            if (i % j == 0)
               continue outer;
          }
          System.out.println (i);
        }
   }


   public static void main(String[] args) {
     doSomething();
   }
}
```

Byte code – raw form – Test.class – “javac –g:none Test.java ”



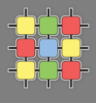
```
00000000h: CA FE BA BE 00 00 00 33 00 20 0A 00 07 00 11 09 ; Êþº¾...3. ......
00000010h: 00 06 00 12 09 00 13 00 14 0A 00 15 00 16 0A 00 ; ................
00000020h: 06 00 17 07 00 18 07 00 19 01 00 01 6A 01 00 01 ; ............j...
00000030h: 49 01 00 06 3C 69 6E 69 74 3E 01 00 03 28 29 56 ; I...<init>...()V
00000040h: 01 00 04 43 6F 64 65 01 00 0B 64 6F 53 6F 6D 65 ; ...Code...doSome
00000050h: 74 68 69 6E 67 01 00 0D 53 74 61 63 6B 4D 61 70 ; thing...StackMap
00000060h: 54 61 62 6C 65 01 00 04 6D 61 69 6E 01 00 16 28 ; Table...main...(
00000070h: 5B 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 ; [Ljava/lang/Stri
00000080h: 6E 67 3B 29 56 0C 00 0A 00 0B 0C 00 08 00 09 07 ; ng;)V...........
00000090h: 00 1A 0C 00 1B 00 1C 07 00 1D 0C 00 1E 00 1F 0C ; ................
000000a0h: 00 0D 00 0B 01 00 04 54 65 73 74 01 00 10 6A 61 ; .......Test...ja
000000b0h: 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01 00 ; va/lang/Object..
000000c0h: 10 6A 61 76 61 2F 6C 61 6E 67 2F 53 79 73 74 65 ; .java/lang/Syste
000000d0h: 6D 01 00 03 6F 75 74 01 00 15 4C 6A 61 76 61 2F ; m...out...Ljava/
000000e0h: 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D 3B 01 ; io/PrintStream;.
000000f0h: 00 13 6A 61 76 61 2F 69 6F 2F 50 72 69 6E 74 53 ; ..java/io/PrintS
00000100h: 74 72 65 61 6D 01 00 07 70 72 69 6E 74 6C 6E 01 ; tream...println.
00000110h: 00 04 28 49 29 56 00 31 00 06 00 07 00 00 00 01 ; ..(I)V.1........
00000120h: 00 0A 00 08 00 09 00 00 00 03 00 01 00 0A 00 0B ; ................
00000130h: 00 01 00 0C 00 00 00 11 00 01 00 01 00 00 00 05 ; ................
00000140h: 2A B7 00 01 B1 00 00 00 00 00 1A 00 0D 00 0B 00 ; *·..±...........
00000150h: 01 00 0C 00 00 00 57 00 02 00 01 00 00 00 38 05 ; ......W.......8.
00000160h: 3B 1A 11 03 E8 A2 00 31 05 B3 00 02 B2 00 02 1A ; ;...è¢.1.³..²...
00000170h: A2 00 19 1A B2 00 02 70 9A 00 06 A7 00 15 B2 00 ; ¢...²..pš..§..².
00000180h: 02 04 60 B3 00 02 A7 FF E6 B2 00 03 1A B6 00 04 ; ..`³..§ÿæ²...¶..
00000190h: 84 00 01 A7 FF CE B1 00 00 00 01 00 0E 00 00 00 ; „..§ÿÎ±.........
000001a0h: 0D 00 06 FC 00 02 01 0A 11 0A 06 FA 00 05 00 09 ; ...ü.......ú....
000001b0h: 00 0F 00 10 00 01 00 0C 00 00 00 10 00 00 00 01 ; ................
000001c0h: 00 00 00 04 B8 00 05 B1 00 00 00 00 00 00       ; ....¸..±......
```

Byte code – opcode form – “javap -c -l -s -private Test”



```
public final class Test {
  private static int j;
    Signature: I

  public Test();
    Signature: ()V
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Signature: ([Ljava/lang/String;)V
    Code:
       0: invokestatic  #5                  // Method doSomething:()V
       3: return
```

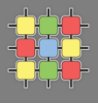
Byte code – opcode form – “javap -c -l -s -private Test”



```
private static final void doSomething();
  Signature: ()V
  Code:
     0: iconst_2
     1: istore_0
     2: iload_0
     3: sipush        1000
     6: if_icmpge     55
     9: iconst_2
    10: putstatic     #2                  // Field j:I
    13: getstatic     #2
    16: iload_0
    17: if_icmpge     42
    20: iload_0
    21: getstatic     #2                  // Field j:I
    24: irem
    25: ifne          31
    28: goto          49
    31: getstatic     #2                  // Field j:I
    34: iconst_1
    35: iadd
    36: putstatic     #2                  // Field j:I
    39: goto          13
    42: getstatic     #3
        // Field java/lang/System.out:Ljava/io/PrintStream;
    45: iload_0
    46: invokevirtual #4                  // Method java/io/PrintStream.println:(I)V
    49: iinc          0, 1
    52: goto          2
    55: return
```

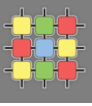
```
for (int i = 2; i < 1000; i++) {
for (j = 2; j < i; j++) {
if (i % j == 0)
continue outer;
System.out.println (i);
```

Decompiled source code – "jad -p Test.class"



```
// Decompiled by Jad v1.5.8g.
import java.io.PrintStream;

public final class Test
{
    public Test()
    {
    }

    private static final void doSomething()
    {
label0:
        for(int i = 2; i < 1000; i++)
        {
            for(j = 2; j < i; j++)
                if(i % j == 0)
                    continue label0;
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        doSomething();
    }

    private static int j;
}
```

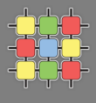
Protection techniques (not only JAVA)



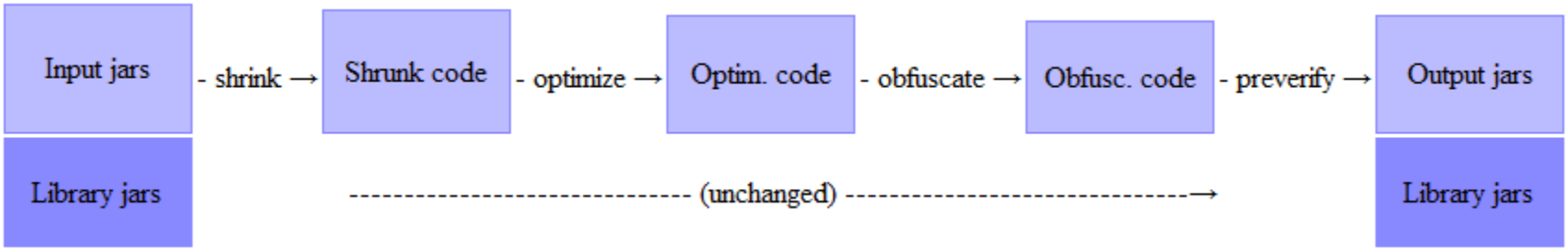
- Protect your code
  - no physical access to the program – e.g. client-server model
  - encryption of code - if not in hardware, can be intercepted
  - native codes instead of byte-code (can be disassembed anyway)
  - code obfuscation (harder for disassembling)

- **Obfuscator** properties
  - the same functional result
  - difficult to understand
  - obfuscations
    - layout – identifier names, class definitions
    - data – local <-> global, encoding, aggregation (e.g. arrays), ordering
    - control – aggregation, ordering, control flow
    - preventative transformations – extra instructions, mix commands

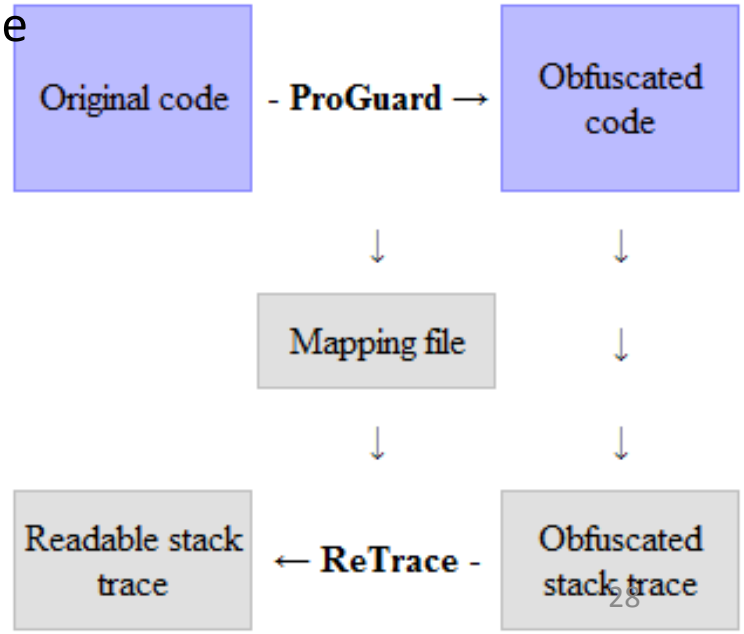
JAVA obfuscators

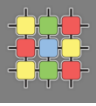


- various open-source obfuscators available
- **changes in byte-code**



- ProGuard
  - define entry points
  - shrinker- compact code, remove dead code
  - optimizer - faster code
    - private, static, final
    - inline
  - obfuscator
    - renaming, layout, etc.
    - automatically detect reflection
  - preverifier – check for Java loading



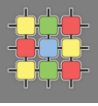


EXAMPLE 2 cont.

(obfuscator + decompiler)

- jar -ce Test Test.class >Test.jar (see jar structure)
- *java –jar Test.jar*
- java –jar proguard.jar @applications.pro (see .pro file)
- Extract Test.class from obfuscated jar
- jad –p Test.class

JAVA archive – "jar -ce Test Test.class >Test.jar"



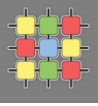
```
Test.jar
|   Test.class
|
\---META-INF
        MANIFEST.MF
```

MANIFEST.MF:

```
Manifest-Version: 1.0
Created-By: 1.7.0_06 (Oracle Corporation)
Main-Class: Test
```

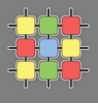
```
java -jar Test.jar

2
3
5
7
11
13
17
19
23
29
```

Proguard configuration file – applications.pro



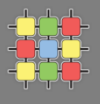
```
-injars  Test.jar
-outjars Test_obfuscated.jar

-libraryjars <java.home>/lib/rt.jar

-keepclasseswithmembers public class * {
    public static void main(java.lang.String[]);
}

-allowaccessmodification
-mergeinterfacesaggressively
-optimizationpasses 9
```

Decompiled obfuscated class



```
// Decompiled by Jad v1.5.8g
import java.io.PrintStream;

public final class Test
{
    public Test()
    {
    }

    public static void main(String args[])
    {
label0:
        for(args = 2; args < 1000; args++)
        {
            for(a = 2; a < args; a++)
                if(args % a == 0)
                    continue label0;
            System.out.println(args);
        }
    }

    private static int a;
}
```

Java virtual machine - JVM



- used to implement also other languages than JAVA
  - Erlang -> Erjang
  - JavaScript -> Rhino
  - Python -> Jython
  - Ruby -> Jruby
  - Scala, Clojure – functional programming
  - others
- java virtual machine VS. java processor

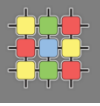
- byte-code is **verified** before executed:
  - branches (jump) are always to valid locations – only within method
  - any instruction operates on a fixed stack location (helps JIT for registers)
  - data is always initialized and references are always type-safe
  - access to private, package is controlled

- **heap** – area for dynamic memory allocation (all objects)

Java virtual machine

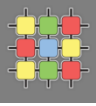
- **Just-in-time** (JIT)
  - converts byte-code into native code in run-time
  - different version for client, server run (-client, -server)
  - client
    - window-based optimization over small set of instructions
    - simplified inlining
  - server – high-end fully optimizing compiler
    - dead code elimination, loop invariant hoisting, common sub-expression elimination, constant propagation
    - full inlining, full deoptimization

Java virtual machine



- **adaptive optimization**
  - balance trade-off between JIT and interpreting instructions
  - monitors frequently executed parts “hot spots” including data on caller-callee relationship for virtual method invocation
  - makes dynamic recompilation based on current execution profile
  - inline expansion to remove context switching
  - optimize branches
  - can make risky assumption (e.g. skip code) ->
    - unwind to valid state
    - deoptimize previously JITed code even if code is already executed

Java virtual machine



- key startup parameters related with JIT and optimization
  - server (defualt for 64-bit edition)
  - XX:+CITime (prints time spent in JIT compiler)
  - XX:+PrintCompilation
  - XX:CompileThreshold=XXXX (number of invocation, branches)
  - XX:+PrintClassLoading