Effective Software

Lecture 3: Virtual machine, byte-code, (de-)compilers, disassembler, profiling

David Šišlák
david.sislak@fel.cvut.cz
Introduction – Virtual Machine

» Virtual machine execution model
  • source code
  • compiled VM byte-code
  • hybrid run-time environment (platform dependent VM impl.)
    – interpreted byte-code
    – complied assembly-code (native CPU code)
    – automated platform capability optimizations (e.g. use of SIMD)

» byte-code vs. assembly-code
  • (+) platform independence (portable) – architecture (RISC/CISC, bits), OS
  • (+) reflection – observe, modify own structure at run-time
  • (+) small size
  • (-) slower execution – interpreted mode, compilation latencies
JAVA Versions

» first release 1996 by Sun Microsystems (later Oracle)
» many different implementations (GNU, IBM, etc.)

» language changes and improvements
  • 1.4 (2002) – assert, NIO
  • 5.0 (2004) – generics, annotations, auto-boxing, enum, concurrency
      utilsi varargs, foreach, profiling API
  • 6 (2005) – basic java script support, performance and GC
      improvements (G1, compressed pointers), compiler API
  • 7 (2011) – invokedynamic, switch strings, auto-closeable, GPU pipeline
      API
  • 8 (2014) – lambda, streams, improved java script support (base on
      invokedynamic), removed permgen (metaspace/native mem. is used)
  • 9 (2017 ?) – Ahead-of-Time Compilation (non-tiered vs. tiered AOT)
      non-tiered AOT provide predictable performance
Execution Time Comparison

» The Computer Language Benchmarks Game

(http://benchmarksgame.alioth.debian.org/)

• 10 different algs. (e.g. DNA manipulation)
JAVA Virtual Machine – Memory Layout
JAVA Virtual Machine – Method Area

» 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
  » implementation of native methods
frame

- each thread has stack with frames (outside of heap, fixed length)
  - StackOverflowError vs. OutOfMemoryError
- frame is **created** each time method is invoked (**destroyed** after return)
- 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
» **stack-oriented** - stack machine model for passing parameters and output

\[(2 + 3) \times 11 + 1\]
JAVA Virtual Machine – Opcodes

- **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, ...) – frame manipulation
  - 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)

```
mov  %rax,%r8
shl  $0x5,%eax
sub  %r8d,%eax
add  %ecx,%eax
inc  %edx
```

*VS.*

```
iconst_0
istore_3
iload_3
bipush 100
```
JAVA Virtual Machine

» used to implement also other languages than JAVA
  » Erlang -> Erjang
  » JavaScript -> Rhino
  » Python -> Jython
  » Ruby -> Jruby
  » Scala, Clojure – functional programming
  » others

» byte-code is **verified** before executed:
  » branches (jumps) are always to valid locations – only within method
  » any instruction operates on a fixed stack location (helps JIT for registers mapping)
  » data is always initialized and references are always type-safe
  » access to private, package is controlled
public class Employee<Type> {
    private Type data;
    public int id;

    public Employee(Type data, int id) {
        update(data, id);
    }

    private void update(Type data, int id) {
        this.data = data;
        this.id = id;
    }

    public Type employeeData() {
        return data;
    }
}
JAVA Virtual Machine – Class File Structure

ClassFile {
    u4 magic;
    u2 minor_version;
    u2 major_version;
    u2 constant_pool_count;
    cp_info constant_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];
}
JAVA Virtual Machine – Example 1 – Class File

```java
0000000  ca fe bo be 00 00 00 34 00 20 0a 00 06 00 19 0a |
0000010  00 05 00 1a 09 00 05 00 1b 09 00 05 00 1c 07 00 |
0000020  1d 87 00 1e 01 00 04 64 61 74 61 01 00 12 4c 6a |
0000030  61 76 61 2f 6c 61 6e 67 2f 62 6f 62 6a 65 63 74 3b |
0000040  01 00 09 53 69 67 6e 61 74 75 72 6f 01 00 00 54 |
0000050  54 79 70 65 3b 01 00 02 69 64 01 00 01 49 01 00 |
0000060  06 3c 66 6e 69 74 3e 01 00 16 28 4c 6a 61 76 61 |
0000070  02 6c 61 6e 67 2f 62 6f 62 6a 65 63 74 3b 01 00 29 |
0000080  01 00 04 43 6f 64 65 01 00 0f 4c 69 66 65 4e 75 |
0000090  7b 62 65 75 54 68 69 6e 67 7b 66 6f 62 6a 65 63 |
00000a0  74 01 00 00 6e 75 68 6f 20 61 70 61 74 69 6f 6e |
00000b0  00 0c 65 6d 70 6c 69 67 68 65 63 74 01 00 00 65 |
00000c0  14 28 29 4c 6a 61 76 61 2f 62 6f 62 6a 65 63 74 |
00000d0  6a 65 63 74 3b 01 00 08 28 29 54 72 61 76 61 74 |
00000e0  01 00 2b 3c 54 72 61 76 61 74 65 61 73 74 01 00 |
00000f0  61 6e 67 2f 4f 62 6a 65 63 74 3b 01 00 00 6e 75 |
0000100  68 6f 20 61 70 61 74 69 6f 6e 00 0c 00 00 0c |
0000110  0c 00 12 00 0e 00 00 0c |
0000120  00 11 65 6d 70 6c 69 67 68 65 63 74 01 00 00 65 |
0000130  00 0e 00 00 00 02 00 00 02 00 07 08 00 00 00 09 |
0000140  05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000150  00 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00001a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00001b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00001c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00001d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00001e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00001f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
0000250  17 00 00 00 00 02 00 18 |
```

6th March 2017
ESW – Lecture 3
13
javap – JAVA disassembler included in JDK

```
public class comemployee.Employee extends java.lang.Object extends java.lang.Object

minor version: 0
major version: 52
flags: ACC_PUBLIC, ACC_SUPER

Constant pool:
    #1 = Methodref    #6.#25  // java/lang/Object."<init>"();V
    #2 = Methodref    #5.#26  // employee/Employee.<init>(Ljava/lang/Object;I)V
    #3 = Fieldref     #5.#27  // employee/Employee.data:Ljava/lang/Object;
    #4 = Fieldref     #5.#28  // employee/Employee.id:I
    #5 = Class        #29     // employee/Employee
    #6 = Class        #30     // java/lang/Object

    #7 = Utf8         data
    #8 = Utf8         Ljava/lang/Object;
    #9 = Utf8         Signature
    #10 = Utf8        TType;
    #11 = Utf8        id
    #12 = Utf8        I
    #13 = Utf8        <init>
    #14 = Utf8        (Ljava/lang/Object;I)V
    #15 = Utf8        Code
    #16 = Utf8        LineNumberTable
    #17 = Utf8        (TType;I)V
    #18 = Utf8        update
    #19 = Utf8        employeeData
    #20 = Utf8        ()Ljava/lang/Object;
    #21 = Utf8        ()TType;
    #22 = Utf8        <Type:Ljava/lang/Object;>Ljava/lang/Object;
    #23 = Utf8        SourceFile
    #24 = Utf8        Employee.java
    #25 = NameAndType #13:#31  // "<init>"();V
    #26 = NameAndType #18:#14  // update:(Ljava/lang/Object;I)V
    #27 = NameAndType #7:#8    // data:Ljava/lang/Object;
    #28 = NameAndType #11:#12  // id:I
    #29 = Utf8        employee/Employee
    #30 = Utf8        java/lang/Object
    #31 = Utf8        ()V
```

Signature: #22  // <Type:Ljava/lang/Object;>Ljava/lang/Object;

6th March 2017
JAVA Virtual Machine – Example 1 – Disassembled Fields

{
    private Type data;
    descriptor: Ljava/lang/Object;
    flags: ACC_PRIVATE
    Signature: #10 // TType;

    public int id;
    descriptor: I
    flags: ACC_PUBLIC
JAVA Virtual Machine – Example 1 – Disassembled Method

```java
public Type employeeData()
{
    return data;
}
```

```
stack=1, locals=1, args_size=1
  0: aload_0
  1: getfield #3 // Field data:Ljava/lang/Object;
  4: areturn
LineNumberTable:
  line 17: 0
Signature: #21 // ()TType;
```  

» **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
public employee.Employee(Type, int);
descraper: (Ljava/lang/Object;I)V
flags: ACC_PUBLIC
Code:
  stack=3, locals=3, args_size=3
  0: aload_0
  1: invokespecial #1              // Method java/lang/Object."<init">:()V
  4: aload_0
  5: aload_1
  6: iload_2
  7: invokespecial #2              // Method update:(Ljava/lang/Object;I)V
  10: return
LineNumberTable:
  line 7: 0
  line 8: 4
  line 9: 10
Signature: #17                   // (Ljava/lang/Object;)V

private void update(Type, int);
descraper: (Ljava/lang/Object;I)V
flags: ACC_PRIVATE
Code:
  stack=2, locals=3, args_size=3
  0: aload_0
  1: aload_1
  2: putfield  #3                // Field data:Ljava/lang/Object;
  5: aload_0
  6: iload_2
  7: putfield  #4                // Field id:I
  10: return
LineNumberTable:
  line 12: 0
  line 13: 5
  line 14: 10
Signature: #17                   // (Ljava/lang/Object;)V

class Employee {
  public Employee(Type data, int id) {
    update(data, id);
  }
  private void update(Type data, int id) {
    this.data = data;
    this.id = id;
  }
}
**procyon** – open-source JAVA decompiler, support JAVA 8

```java
package employee;

public class Employee<Type> {
    private Type data;
    public int id;

    public Employee(final Type type, final int n) {
        this.update(type, n);
    }

    private void update(final Type data, final int id) {
        this.data = data;
        this.id = id;
    }

    public Type employeeData() {
        return this.data;
    }
}
```

```java
public class Employee<Type> {
    private Type data;
    public int id;

    public Employee(Type data, int id) {
        update(data, id);
    }

    private void update(Type data, int id) {
        this.data = data;
        this.id = id;
    }

    public Type employeeData() {
        return data;
    }
}
```
private static Integer daysInMonth(int month, int year) {
    int retVal;
    switch (month) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            retVal = 31;
            break;
        case 2:
            retVal = (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)) ? 29 : 28;
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            retVal = 30;
            break;
        default:
            throw new IllegalArgumentException("Unknown month: " + month);
    }
    return new Integer(retVal);
}

private static int compute() {
    int month = 4;
    int year = 2000;
    int o = 0;
    for (int i = 0; i < 1_000_000; i++) {
        o += daysInMonth(month, year);
    }
    return o;
}
private static java.lang.Integer daysInMonth(int, int);
descriptor: ([II]Ljava/lang/Integer;
flags: ACC_PRIVATE, ACC_STATIC
Code:
    stack=4, locals=3, args_size=2
    0: iload_0
    1: tableswitch { // 1 to 12
        0: 64
        1: 70
        2: 64
        3: 4
        4: 102
        5: 64
        6: 102
        7: 64
        8: 64
        9: 102
       10: 64
       11: 102
       12: 64
    default: 108
} 64: bipush 31
66: istore_2
67: goto
70: iload_1
71: iconst_4
72: irem
73: ifne
76: iload_1
77: bipush 100
79: irem
80: ifne
83: iload_1
84: sipush 400
87: irem
88: ifne
91: bipush 29
93: goto
96: bipush 28
98: istore_2
99: goto
102: bipush 30
104: istore_2
105: goto
135
int ret_val;
switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        retval=31;
        break;
    case 2:
        retval = (year % 4 == 0 &&
            (year % 100 != 0 ||
            year % 400 == 0)) ?
            29 : 28;
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        retval = 30;
        break;
    default:
        throw new IllegalArgumentException(
            "Unknown month: " + month);
    }
    return new Integer(retval);
JAVA Virtual Machine – Example 2 – Switch Bytecode

```
108: new #2
111: dup
112: new #3
115: dup
116: invokespecial #4
119: ldc #5
121: invokevirtual #6
124: iload_0
125: invokevirtual #7
128: invokevirtual #8
131: invokespecial #9
134: athrow
135: new #10
138: dup
139: iload_2
140: invokespecial #11
143: areturn

LineNumberTable:
line 7: 0
line 16: 64
line 17: 67
line 19: 70
line 20: 99
line 25: 102
line 26: 105
line 28: 108
line 30: 135

StackMapTable: number_of_entries = 8
frame_type = 251 /* same_frame_extended */
  offset_delta = 64
frame_type = 5 /* same */
frame_type = 20 /* same */
frame_type = 4 /* same */
frame_type = 65 /* same_locals_1_stack_item */
  stack = [ int ]
frame_type = 3 /* same */
frame_type = 5 /* same */
frame_type = 252 /* append */
  offset_delta = 26
locals = [ int ]
```

```java
// class java/lang/IllegalArgumentException

// Method java/lang/StringBuilder."<init>":(?)V
// String Unknown month:
// Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
// Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
// Method java/lang/Integer."<init>":(?I)V

default:
  throw new IllegalArgumentException("
  Unknown month: " + month);
}
return new Integer(retVal);
```
private static int compute()
{
    int month = 4;
    int year = 2000;
    int o=0;
    for (int i=0; i<1_000_000; i++) {
        o+=daysInMonth(month, year);
    }
    return o;
}

private static int compute();
derector: ()I
flags: ACC_PRIVATE, ACC_STATIC
Code:
  stack=3, locals=4, args_size=0
  0: invokevirtual #12          // int 1000000
  13: if_icmpge  33
  16: iload_2
  17: iload_0
  18: iload_1
  19: invokestatic #13         // Method daysInMonth:(II)Ljava/lang/Integer;
  22: invokevirtual #14        // Method java/lang/Integer.intValue():I
  25: iadd
  26: istore_2
  27: inc 3, 1
  30: goto 10
  33: iload_2
  34: ireturn
LineNumberTable:
    line 34: 0
    line 35: 2
    line 36: 6
    line 37: 8
    line 38: 16
    line 37: 27
    line 40: 33
StackTrace:
   number_of_entries = 2
   frame_type = 255 /* full_frame */
   offset_delta = 10
   locals = [ int, int, int, int ]
   stack = []
   frame_type = 250 /* chop */
   offset_delta = 22
source code compilation (Source->Bytecode)
  » bytecode is not better than your source code
    » invariants in loop are not removed
  » no optimizations like
    » loop unrolling
    » algebraic simplification
    » strength reduction

optional external obfuscator bytecode optimizations - ProGuard
  • shrinker – compact code, remove dead code
  • optimizer
    – modify access pattern (private, static, final)
    – inline bytecode
  • obfuscator – renaming, layout changes
  • preverifier – ensure class loading
» **Just-in-time (JIT)**
  » converts bytecode into assembly code in run-time
  » look OpenJDK sources for very detailed information

» **adaptive optimization** (tiered compilation)
  » balance trade-off between JIT and interpreting instructions
  » monitors frequently executed parts “hot spots” including data on caller-callee relationship for virtual method invocation
  » triggers dynamic re-compilation 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
> Just-in-time (JIT) – usually asynchronous (3 C1, 7 C2 threads for 32 cores)
>   » **C1** (client) – much faster than C2
>     » simplified inlining, using CPU registry
>     » window-based optimization over small set of instructions
>     » intrinsic functions with vector operations (Math, `arraycopy`, …)
>   » **C2** (server, d64) – high-end fully optimizing compiler
>     » dead code elimination, loop unrolling, loop invariant hoisting, common sub-expression elimination, constant propagation
>     » full inlining, full deoptimization (back to level 0)
>     » escape analysis, null check elimination,
>     » pattern-based loop vectorization and super word packing (**SIMD**)
>   » **tiered compilation** – hybrid adapting (since JVM 7, default in JVM8)
>   » on-stack replacement (OSR) – optimization during execution
>     » start at bytecode jump targets (goto, if_)

```java
CompLevel_none = 0, // Interpreter
CompLevel_simple = 1, // C1
CompLevel_limited_profile = 2, // C1, invocation & backedge counters
CompLevel_full_profile = 3, // C1, invocation & backedge counters + mdo
CompLevel_full_optimization = 4, // C2
```
Assembly Code

» reasons to study assembly code (both Java and C/C++)
  • educational reasons
    – predict efficient coding techniques
  • debugging and verification
    – how well the code looks like
  • optimize code
    – for speed
      • avoid poorly compiled patterns
      • data fits into cache
      • predictable branches or no branches
      • use vector programming if possible (SIMD)
        » 256bit registers with AVX2 since Intel Sandy Bridge
        » 512bit AVX-512 since Intel Knight Landing (Xeon Phi)
    – for size
      • primarily code cache efficiency
JAVA Virtual Machine – Example 2 – Tiered Compilation

» -XX:+PrintCompilation (-XX:+PrintInlining)

```
67    1    3    java.lang.String::hashCode (55 bytes)
68    2    3    java.lang.String::charAt (29 bytes)
69    3    3    java.lang.String::length (6 bytes)
74    4    3    java.lang.String::indexOf (70 bytes)
74    5    n    0    java.lang.System::arraycopy (native) (static)
74    6    3    java.lang.String::equals (81 bytes)
75    8    3    java.lang.Object::<init> (1 bytes)
75    9    3    java.lang.Math::min (11 bytes)
75    7    3    java.lang.AbstractStringBuilder::ensureCapacityInternal (16 bytes)
75    10   3    java.lang.AbstractStringBuilder::append (50 bytes)
76    11   3    java.lang.String::getChars (62 bytes)
81    12   1    java.lang.ref.Reference::get (5 bytes)
81    13   3    java.lang.StringBuilder::append (8 bytes)
82    14   3    java.lang.String::indexOf (7 bytes)
83    16   3    java.lang.Number::<init> (5 bytes)
83    19   1    java.lang.Object::<init> (1 bytes)
84    8    3    java.lang.Object::<init> (1 bytes)  made not entrant
84    18   3    SwitchTest::daysInMonth (144 bytes)
84    17   3    java.lang.Integer::<init> (10 bytes)
84    15   1    java.lang.Integer::intValue (5 bytes)
84    20   4    SwitchTest::daysInMonth (144 bytes)
86    18   3    SwitchTest::daysInMonth (144 bytes)  made not entrant
88    21   %   3    SwitchTest::compute @ 10 (35 bytes)
88    22   3    SwitchTest::compute (35 bytes)
89    23   %   4    SwitchTest::compute @ 10 (35 bytes)
91    21   %   3    SwitchTest::compute @ -2 (35 bytes)  made not entrant
91    23   %   4    SwitchTest::compute @ -2 (35 bytes)  made not entrant
92    24   %   4    SwitchTest::compute @ 10 (35 bytes)
94    25   4    SwitchTest::compute (35 bytes)
95    22   3    SwitchTest::compute (35 bytes)  made not entrant
```
JVM – Example 2 – daysInMonth Assembly Code – Tier 3

-XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly

all examples are in JVM 8 64-bit Server, Intel Haswell CPU, AT&T syntax

tier 3 - C1 with invocation & backedge counters + MethodDataOop counter

because: count="256" iicount="256" hot_count="256"

stack initialization, invocation counter in MDO (0xDC) + trigger C2

0x1ff8 >> 3 = 1024 invocations trigger tier 4 (C2)
JVM – Example 2 – daysInMonth Assembly Code – Tier 3

```
0x00000000108a953c4: cmp $0x1,%esi
0x00000000108a953c7: je 0x00000000108a95597
0x00000000108a953cd: cmp $0x2,%esi
0x00000000108a953d0: je 0x00000000108a95435
0x00000000108a953d6: cmp $0x3,%esi
0x00000000108a953d9: je 0x00000000108a95597
0x00000000108a953df: cmp $0x4,%esi
0x00000000108a953e2: je 0x00000000108a9557d
0x00000000108a953e8: cmp $0x5,%esi
0x00000000108a953eb: je 0x00000000108a95597
0x00000000108a953f1: cmp $0x6,%esi
0x00000000108a953f4: je 0x00000000108a9557d
0x00000000108a953fa: cmp $0x7,%esi
0x00000000108a953fd: je 0x00000000108a95597
0x00000000108a95403: cmp $0x8,%esi
0x00000000108a95406: je 0x00000000108a95597
0x00000000108a9540c: cmp $0x9,%esi
0x00000000108a9540f: je 0x00000000108a9557d
0x00000000108a95415: cmp $0xa,%esi
0x00000000108a95418: je 0x00000000108a95597
0x00000000108a9541e: cmp $0xb,%esi
0x00000000108a95421: je 0x00000000108a9557d
0x00000000108a95427: cmp $0xc,%esi
0x00000000108a9542a: je 0x00000000108a95597
0x00000000108a95430: jmpq 0x00000000108a956d0 ;*tableswitch
       ; - SwitchTest::daysInMonth@1 (line 7)
```

ESI is month input

default jump
target for month=4, **backedge counter** tracking in MDO (0x290):

```
0x0000000108a9557d: movabs $0x12169db40,%rdx
0x0000000108a95587: incl %rdx
0x0000000108a9558d: mov $0x1e,%ebx
0x0000000108a95592: jmpq 0x0000000108a9559ac ;*goto
                  ; - SwitchTest::daysInMonth@105 (line 26)
```

EBX=30 is retVal

jump target, **inlined TLAB allocation** of Integer object:

```
0x0000000108a955ac: movabs $0x7c0011320,%rdx
0x0000000108a955b6: mov 0x60(%r15),%rax
0x0000000108a955ba: lea 0x10(%rax),%rdi
0x0000000108a955be: cmp 0x70(%r15),%rdi
0x0000000108a955c2: ja 0x0000000108a955bc
0x0000000108a955c8: mov %rdi,0x60(%r15)
0x0000000108a955cc: mov 0x8(%rdx),%rcx
0x0000000108a955d3: mov %rcx,(%rax)
0x0000000108a955d6: mov %rdx,%rcx
0x0000000108a955d9: shr $0x3,%rcx
0x0000000108a955dd: mov %ecx,0x8(%rax)
0x0000000108a955e0: xor %rcx,%rcx
0x0000000108a955e3: mov %ecx,0xcc(%rax)
0x0000000108a955e6: xor %rcx,%rcx
                  ;*new ; - SwitchTest::daysInMonth@135 (line 30)
```

0x10 Integer instance size

no space in TLAB -> new TLAB + external allocation

with header init returns after the inlined allocation

object initialization, header filed with prototype mark

---

**RAX Integer instance address**

Object structure (64-bit JVM):
- header 12 or 16 Bytes
- object data super class first, type grouped

<table>
<thead>
<tr>
<th>8B - mark word</th>
</tr>
</thead>
<tbody>
<tr>
<td>4B / 8B – Klass ref.</td>
</tr>
<tr>
<td>… object data</td>
</tr>
</tbody>
</table>

Array object structure (64-bit JVM):
- header 16 or 20 Bytes
- sequence of array values

<table>
<thead>
<tr>
<th>8B - mark word</th>
</tr>
</thead>
<tbody>
<tr>
<td>4B / 8B – Klass ref.</td>
</tr>
<tr>
<td>4B – array length</td>
</tr>
<tr>
<td>sequence of values</td>
</tr>
</tbody>
</table>
inlined Integer constructor with supers, invocation counts in MDOs (0xDC)

Integer::<init>, Number::<init>, Object::<init>

- currently in tier 3 (C1 counters in MDO)

```
0x00000000018a955e9: mov %rax,%rdx
0x00000000018a955ec: movabs $0x12169db40,%rsi
0x00000000018a955f6: addq $0x1,0x358(%rsi)
0x00000000018a955fe: movabs $0x1214df850,%rdx
0x00000000018a95608: mov %rdc(%rdx),%esi
0x00000000018a9560e: add $0x8,%esi
0x00000000018a95611: mov %esi,0xrdc(%rdx)
0x00000000018a95617: movabs $0x121341738,%rdx
0x00000000018a95621: and $0x7ffff8,%esi
0x00000000018a95627: cmp $0x0,%esi
0x00000000018a9562c: je 0x00000000018a959c9
0x00000000018a95630: mov %rax,%rdx
0x00000000018a95633: movabs $0x1214df850,%rsi
0x00000000018a9563d: addq $0x1,0x108(%rsi)
0x00000000018a95645: movabs $0x1214df720,%rdx
0x00000000018a9564f: mov %rdc(%rdx),%esi
0x00000000018a95655: add $0x8,%esi
0x00000000018a95658: mov %esi,0xrdc(%rdx)
0x00000000018a9565e: movabs $0x12133a9d8,%rdx
0x00000000018a95668: and $0x7ffff8,%esi
0x00000000018a9566e: cmp $0x0,%esi
0x00000000018a95671: je 0x00000000018a959e0
0x00000000018a95677: mov %rax,%rdx
0x00000000018a9567a: movabs $0x1214df720,%rsi
0x00000000018a95684: addq $0x1,0x108(%rsi)
0x00000000018a9568c: movabs $0x12140dd8,%rdx
0x00000000018a95696: mov %rdc(%rdx),%esi
0x00000000018a9569c: add $0x8,%esi
0x00000000018a9569f: mov %esi,0xrdc(%rdx)
0x00000000018a956a5: movabs $0x12129d480,%rdx
0x00000000018a956af: and $0x7ffff8,%esi
0x00000000018a956b5: cmp $0x0,%esi
0x00000000018a956b8: je 0x00000000018a959f7
0x00000000018a956be: mov %ebx,0xc(%rax)
```

```
0x00000000018a955e9: mov %rax,%rdx
0x00000000018a955ec: movabs $0x12169db40,%rsi
0x00000000018a955f6: addq $0x1,0x358(%rsi)
0x00000000018a955fe: movabs $0x1214df850,%rdx
0x00000000018a95608: mov %rdc(%rdx),%esi
0x00000000018a9560e: add $0x8,%esi
0x00000000018a95611: mov %esi,0xrdc(%rdx)
0x00000000018a95617: movabs $0x121341738,%rdx
0x00000000018a95621: and $0x7ffff8,%esi
0x00000000018a95627: cmp $0x0,%esi
0x00000000018a9562c: je 0x00000000018a959c9
0x00000000018a95630: mov %rax,%rdx
0x00000000018a95633: movabs $0x1214df850,%rsi
0x00000000018a9563d: addq $0x1,0x108(%rsi)
0x00000000018a95645: movabs $0x1214df720,%rdx
0x00000000018a9564f: mov %rdc(%rdx),%esi
0x00000000018a95655: add $0x8,%esi
0x00000000018a95658: mov %esi,0xrdc(%rdx)
0x00000000018a9565e: movabs $0x12133a9d8,%rdx
0x00000000018a95668: and $0x7ffff8,%esi
0x00000000018a9566e: cmp $0x0,%esi
0x00000000018a95671: je 0x00000000018a959e0
0x00000000018a95677: mov %rax,%rdx
0x00000000018a9567a: movabs $0x1214df720,%rsi
0x00000000018a95684: addq $0x1,0x108(%rsi)
0x00000000018a9568c: movabs $0x12140dd8,%rdx
0x00000000018a95696: mov %rdc(%rdx),%esi
0x00000000018a9569c: add $0x8,%esi
0x00000000018a9569f: mov %esi,0xrdc(%rdx)
0x00000000018a956a5: movabs $0x12129d480,%rdx
0x00000000018a956af: and $0x7ffff8,%esi
0x00000000018a956b5: cmp $0x0,%esi
0x00000000018a956b8: je 0x00000000018a959f7
0x00000000018a956be: mov %ebx,0xc(%rax)
```

RAX.value = EBX (retVal)
final cleanup and return, RAX contains return value (pointer to Integer instance)

```assembly
data $0x80,%rsp
pop %rdi
lea $0x214c5cf(%rip)
lea $0x000000100949100
```

- **safepoint** – Oops in perfectly described state by OopMap (GCmaps)
  - threads are suspended, Oop safely manipulated externally and resumed after
  - in interpreted mode – between any 2 byte codes
  - in C1/C2 compiled – end of all methods (not in-lined), non-counted loop back edge, during JVM run-time call
  - parked, blocked on IO, monitor or lock
  - while running JNI (do not need thread suspension)
  - **global safepoint (all threads)** – stop the world
    - GC, print threads, thread dumps, heap dump, get all stack trace
    - enableBiasedLocking, RevokeBias
    - class redefinition (e.g. instrumentation), debug
  - **local safepoint (just executing thread)**
    - de-optimization, enable/revoke bias locking, OSR
JVM – Time To Safe Point (TTSP)

-XX:+PrintSafepointStatistics -XX:+PrintGCApplicationStoppedTime -XX:PrintSafepointStatisticsCount=1

GetStackTrace overheads:

![Diagram showing JVM states and stack traces]

- **GREEN**: Executing Java
- **YELLOW**: Descheduled Java
- **RED**: Native
JVM – Example 2 – daysInMonth Assembly Code – Tier 4

**tier 4 – C2 – no profile counters**

because: `count="5376"` `iicount="5376"` `hot_count="5376"`

stack initialization, **use lookup table jump** for table switch

---

```
[Entry Point]
[Verified Entry Point]
#
# method: 'daysInMonth' '()Ljava/lang/Integer;' in 'SwitchTest'
# params: rsi = int
# params: rdx = int
# [sp+0x20] (sp of caller)
0x0000000108a97020: mov %eax, -0x14000(%rsp) ; [no_reloc]
0x0000000108a97027: push %ebp
0x0000000108a97028: sub $0x10, %rsp
  *synchronization entry
  ; SwitchTest::daysInMonth@-1 (line 7)
0x0000000108a9702c: mov $esi, %r11d
0x0000000108a9702f: dec %r11d
0x0000000108a97032: cmp $0xc, %r11d
0x0000000108a97036: jae 0x0000000108a9704a
0x0000000108a97038: movsilq $esi, %r10
0x0000000108a9703b: movabs $0x108a96fco, %r11
  ; {section_word}
0x0000000108a97045: jmpq -0x8(%r11, %r10, 8) ; *tablesSwitch
  ; SwitchTest::daysInMonth@1 (line 7)
```

---

```
0x0000000108a96fc0 (offset: 0): 0x08d97083 0x0000000108a97083
0x0000000108a96fc4 (offset: 4): 0x00000001
0x0000000108a96fc8 (offset: 8): 0x08d9706c 0x0000000108a9706c
0x0000000108a96f00 (offset: 12): 0x00000001
0x0000000108a96f08 (offset: 16): 0x08d97083 0x0000000108a97083
0x0000000108a96f0c (offset: 20): 0x00000001
0x0000000108a96f08 (offset: 24): 0x08d9708a 0x0000000108a9708a
0x0000000108a96f08 (offset: 28): 0x00000001
0x0000000108a96f00 (offset: 32): 0x08d97083 0x0000000108a97083
0x0000000108a96f04 (offset: 36): 0x00000001
0x0000000108a96f08 (offset: 40): 0x08d9708a 0x0000000108a9708a
0x0000000108a96f08 (offset: 44): 0x00000001
0x0000000108a96f00 (offset: 48): 0x08d97083 0x0000000108a97083
0x0000000108a96f04 (offset: 52): 0x00000001
0x0000000108a96f08 (offset: 56): 0x08d97083 0x0000000108a97083
0x0000000108a96f08 (offset: 60): 0x00000001
0x0000000108a96f00 (offset: 64): 0x08d9708a 0x0000000108a9708a
0x0000000108a96f04 (offset: 68): 0x00000001
0x0000000108a96f00 (offset: 72): 0x08d97083 0x0000000108a97083
0x0000000108a96f04 (offset: 76): 0x00000001
0x0000000108a96f00 (offset: 80): 0x08d9708a 0x0000000108a9708a
0x0000000108a96f04 (offset: 84): 0x00000001
0x0000000108a96f00 (offset: 88): 0x08d97083 0x0000000108a97083
0x0000000108a96f04 (offset: 92): 0x00000001
```
target for month=4

Integer.<init>, Number.<init>, Object.<init> - iicount="5376" -> Inline (hot) optimized branching, inlined TLAB allocation, inlined constructors, no nulling, caching optimization

```assembly
0x00000000108a9708a: mov $0x1e,%ebp ;*goto EBP=30 is retVal
0x00000000108a9708f: mov 0x60(%r15),%rax
0x00000000108a97093: mov %rax,%r10
0x00000000108a97096: add $0x10,%r10
0x00000000108a9709a: cmp 0x70(%r15),%r10
0x00000000108a9709e: jae 0x00000000108a97124
0x00000000108a970a4: mov %r10,0x60(%r15)
0x00000000108a970a8: prefetchnta 0xc0(%r10)
0x00000000108a970b0: mov $0xf8002264,%r10d ; {metadata('java/lang/Integer')}
0x00000000108a970b6: shl $0x3,%r10
0x00000000108a970ba: mov 0x8(%r10),%r10
0x00000000108a970c1: mov %r10,(%rax)
0x00000000108a970c4: movl $0xf8002264,0x8(%rax) ;*new compressed OOP to Integer class
0x00000000108a970cb: mov %ebp,0xc(%rax) ;*synchronization entry RAX.value = EBX (retVal)
0x00000000108a970ce: add $0x10,%rsp
0x00000000108a970d2: pop %rbp
0x00000000108a970d3: test %eax,-0x214e0d9(%rip) # 0x00000000106949000
0x00000000108a970d3: test %eax,-0x214e0d9(%rip) # 0x00000000106949000
0x00000000108a970d9: retq
```

TLAB Integer object allocation, ref in RAX

MarkWord fetch from class and then store compressed OOP to Integer class

final cleanup

RAX contains return value (pointer to Integer instance)
target for default
class IllegalArgumentException no profile -> uncommon -> reinterpret

remap inputs, return back to reinterpreter

```
0x00000000108a9704a: mov %esi,%ebp
0x00000000108a9704c: mov $0x2,%esi
0x00000000108a97051: xchg %ax,%ax
0x00000000108a97053: callq 0x0000000010898b1a0 ; OopMap{off=56}
    ;*new ; - SwitchTest::daysInMonth@108 (line 28)
    ; {runtime_call}

0x00000000108a97058: callq 0x00000000107e7e33c ;*new
    ; - SwitchTest::daysInMonth@108 (line 28)
    ; {runtime_call}
```

then discard tier 3 version

```
138 17 3 SwitchTest::daysInMonth (144 bytes) made not entrant
```
**OSR @10 – On Stack Replacement at bytecode 10**

**tier 4 – C2** (before there was tier 3 OSR @10 because 60416 loops and tier 3) because: backedge_count=“101376” hot_count=“101376”

---

**JVM – Example 2 – compute Assembly Code – Tier 4 OSR**

```java
class SwitchTest {
    int compute() {
        int month = 4;
        int year = 2000;
        int o = 0;
        for (int i = 0; i < 1_000_000; i++) {
            o += daysInMonth(month, year);
        }
        return o;
    }

    static int daysInMonth(int month, int year) {
        return 31; // example value
    }
}
```

---

RSI compiled stack of tier 3 OSR @10

```assembly
0x00000000108a98370: mov %eax,-0x14000(%rsp)
0x00000000108a98377: push %rbp
0x00000000108a98378: sub $0x20,%rsp
0x00000000108a9837c: mov (%rsi),%ebx
0x00000000108a98380: mov 0x18(%rsi),%ebp
0x00000000108a98381: mov 0x10(%rsi),%r13d
0x00000000108a98385: mov 0x8(%rsi),%r14d
0x00000000108a98389: mov %rsi,%rdi
```

---

© 2017 ESW – Lecture 3
loop criteria

```
for (int i=0; i<1_000_000; i++) {
    o++=daysInMonth(month, year);
}
```

```
EBX is local I; 0xF4240 = 1_000_000
```

then there is **inlined** tier 4 daysOfMonth (lookup jump) because the call is **hot** ending with addition into accumulator o

```
0x0000000108a9841a: add %r8d, %r14d ; OopMap{off=189}
    *goto
    - SwitchTest::compute@30 (line 37)
```

reinterpret on end of cycle jump (unstable if_bytecode), save 3 locals to stack
tier 4 – C2

because: count="2" backedge_count="150528"

use combination of full inline, dead code elimination, object escape, loop invariant hoisting, strength reduction

157 23 b 4 SwitchTest::compute (35 bytes)
Decoding compiled method 0x0000000108a97f90:
Code:
[Entry Point]
[Verified Entry Point]
[Constants]
  # {method} {0x0000000012169d638} 'compute' '()I' in 'SwitchTest'
  # 
  # [sp+0x20] (sp of caller)
0x00000000108a980c0: sub $0x18,%rsp
0x00000000108a980c7: mov %rbp,0x10(%rsp) ;*synchronization entry
  ; - SwitchTest::compute@-1 (line 34)

0x00000000108a980cc: mov $0x1c9c380,%eax 30_000_000
0x00000000108a980d1: add $0x10,%rsp
0x00000000108a980d5: pop %rbp
0x00000000108a980d6: test %eax,-0x214f0dc(%rip) # 0x0000000010694900
  ; {poll_return}
0x00000000108a980dc: retq  RAX contains return value (primitive int)
Java Virtual Machine – Performance 32 vs 64-bit

- requires warm-up to utilize benefits of C2 (or C1)
- compilers cannot do all magic → write better algorithms

- 32-bit vs 64 bits JVMs
  - 32-bit (max ~3GB heap)
    - smaller memory footprint
    - slower long & double operations
  - 64-bit max 32GB virtual memory (with default ObjectAlignmentInBytes)
    - faster performance for long&double
    - slight increase of memory footprint
    - compressed OOPs are slightly slower for references upon usage
    - compressed OOPs less memory -> less frequent GC -> faster program
  - 64-bit >32GB virtual memory (large heap)
    - fast reference usage
    - wasting a lot of memory (48GB ~32GB with compressed OOPs)
Java Virtual Machine – CPU and Memory Profiling

» jvisualvm
  • JVM monitoring, troubleshooting and profiling tool
  • included in all JDKs
  • profiled thread limit 32

» profiling
  • CPU – time spent in methods
  • memory – usage, allocations

» modes
  • sampling
    – periodic sampling of stacks of running threads to estimate slowest
    – no invocation counts, no 100% accuracy (various sampling errors)
    – no bytecode (& assembly code) modifications
    – 1-2% impact to standard performance
  • tracing (instrumentation)
    – instrumented bytecode -> affected performance -> affected
      compiler optimizations
assembly code of tier 4 – C2 (before there was very complex tier 3)

inline daysInMonth rootMethodEntry tracking

```
# {method} {0x000000012489e838} 'daysInMonth' '(II)Ljava/lang/Integer;' in 'SwitchTest'
# parm0: rsi = int
# parm1: rdx = int
# [sp+0x70] (sp of caller)
0x000000010c08aa80: mov %eax,-0x14000(%rsp) ; {no_reloc}
0x000000010c08aa87: push %rbp
0x000000010c08aa88: sub $0x60,%rsp ; *synchronization entry
 - SwitchTest::daysInMonth@-1 (line 7)

0x000000010c08aa8c: mov %edx,0x4(%rsp)
0x000000010c08aa90: mov %esi,(%rsp)
0x000000010c08aa93: movabs 0x76c73a180,%r10 ; {oop(a 'java/lang/Class' = 'org/netbeans/lib/profiler/server/ProfilerRuntimeCPU')}
0x000000010c08aa9d: movzlq 0x82(,%r10,4),%r11d ; *getstatic recursiveInstrumentationDisabled
 - org.netbeans.lib.profiler.server.ProfilerRuntimeCPUFullInstr::rootMethodEntry@0 (line 189)
 - SwitchTest::daysInMonth@3 (line 7)

0x000000010c08aa5: test %r11d,%r11d ; *ifeq
0x000000010c08aa8: jne 0x000000010c08b075 ; *ifeq
 - org.netbeans.lib.profiler.server.ProfilerRuntimeCPUFullInstr::rootMethodEntry@3 (line 189)
 - SwitchTest::daysInMonth@3 (line 7)

0x000000010c08aae: movabs 0x76c73e220,%r10 ; {oop(a 'java/lang/Class' = 'org/netbeans/lib/profiler/server/ThreadIdInfo')}
0x000000010c08ab8: mov 0x78(,%r10,4),%r8d ; *getstatic lastThreadIdInfo
 - org.netbeans.lib.profiler.server.ThreadInfo::ThreadIdInfo@4 (line 244)
 - org.netbeans.lib.profiler.server.ProfilerRuntimeCPUFullInstr::rootMethodEntry@7 (line 193)
 - SwitchTest::daysInMonth@3 (line 7)

0x000000010c08abc: mov 0x40(%r12,%r8,8),%ebp ; *getfield thread
 - org.netbeans.lib.profiler.server.ThreadInfo::ThreadIdInfo@9 (line 246)
```

749 Bytes of assembly code for each rootMethodEntry
additional `rootMethodEntry` and `rootMethodExit` trackings for `Integer::<init>` and `Number::<init>`

inlined `rootMethodExit` after `Integer instance.value = retVal`

313 Bytes of assembly code for each `rootMethodEntry`
### JVM – Example 2 – CPU Tracing Outcome

The image shows a screenshot of Java VisualVM with a call tree analysis for the method `SwitchTest.compute()`. The call tree highlights the methods and their invocations, along with the total time and percentage of time each method has consumed.

#### Call Tree – Method `SwitchTest.compute()`

- **RMI TCP Connection(idle)**
  - Total Time: 83,120 ms (100%)
  - Invocations: 1

- **main**
  - Total Time: 8,444 ms (100%)
  - Invocations: 1

- **SwitchTest.daysInMonth** (int, int)
  - Total Time: 5,059 ms (59.9%)
  - Invocations: 100

- **java.lang.Integer.<init>** (int)
  - Total Time: 2,808 ms (33.3%)
  - Invocations: 100

- **java.lang.Number.<init>**
  - Total Time: 1,750 ms (20.7%)
  - Invocations: 100

- **SwitchTest.waitForAnyInputLine**
  - Total Time: 3,384 ms (40.1%)
  - Invocations: 100

- **SwitchTest.main**
  - Total Time: 8,444 ms (100%)
  - Invocations: 1

The screenshot is from a Java program named `SwitchTest` with an ID of 84916. The analysis is showing the time distribution among various methods and invocations, providing insights into the performance of the program.
CPU tracing of `compute` results into much slower code

- no object escape from `daysInMonth` call
- no invariant hoisting
- no strength reduction (full loop remains there)

Object allocation is similar with `traceObjAlloc` injected calls

Recommended approach

- do sampling first
- identify performance bottlenecks (where most time is spent)
  - it could be outside of JVM (e.g. latency of external DB, file system)
- focus with tracing just to identified parts
JVM – Java Mission Control

**jmc** – JRockit JVM, included in commercial JDKs, sampling in Flight recorder
Approach to Performance Testing

» **test real application** – ideally the way it is used
  
  • **microbenchmarks** – measure very small units
    - warm-up – to measure real code, not compilers itself, biased locks
    - keep in mind caching
    - beware of compilers – use results, reordering of operations
    - synchronization – multi-threaded benchmarks
    - vary pre-calculated right parameters affecting complexity – different optimization in reality
  
  • **macrobenchmarks** – measure application input/output
    - least performing component affects the whole application
  
  • **mesobenchmarks** – isolating performance at modular level

» **understand throughput, elapsed and response time**
  
  • outliers can occur – e.g. GC
  
  • use existing generators than writing own
Approach to Performance Testing

» **understand variability** – changes over time
  • internal state
  • background effects – load, network
  • probabilistic analysis – works with uncertainty

» **test early, test often** – ideally part of development cycle
  • ideally some properly repeated mesobenchmarking
  • automate tests – scripted
  • proper test coverage of functionality and inputs
  • test on target system – different code on different systems