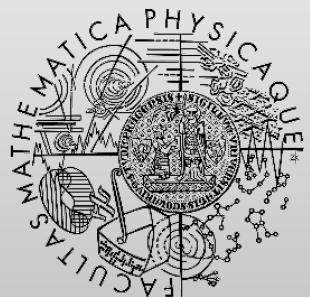


# Bytecode

<http://d3s.mff.cuni.cz>



FACULTY  
OF MATHEMATICS  
AND PHYSICS  
Charles University

*Tomas Bures*  
`bures@d3s.mff.cuni.cz`

- Machine code of a JVM
  - stack-based
  - with constructs for manipulation with classes/instances
- The Java™ Virtual Machine Specification
  - <https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf>
- Instructions Overview
  - [http://en.wikipedia.org/wiki/Java\\_bytecode\\_instruction\\_listings](http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings)



# Example – Basics



```
void spin() {  
    int i;  
    for (i = 0; i < 100; i++) {  
        ; // Loop body is empty  
    }  
}
```



```
Method void spin()  
0  iconst_0      // Push int constant 0  
1  istore_1      // Store into local variable 1 (i=0)   
2  goto 8        // First time don't increment  
5  iinc 1 1      // Increment local variable i by 1  
8  iload_1       // Push local variable 1 (i)  
9  bipush 100    // Push int constant 100  
11 if_icmplt 5   // Compare and loop if (i < 100)  
14 return        // Return void when done
```

# Instruction set – Load and Store



- Load a local variable onto the operand stack
  - *iload, iload\_<n>, lload, lload\_<n>, fload, fload\_<n>, dload, dload\_<n>, aload, aload\_<n>* 
- Store a value from the operand stack into a local variable
  - *istore, istore\_<n>, lstore, lstore\_<n>, fstore, fstore\_<n>, dstore, dstore\_<n>, astore, astore\_<n>*
- Load a constant onto the operand stack
  - *bipush, sipush, ldc, ldc\_w, ldc2\_w, aconst\_null, istruct\_m1, istruct\_<i>, ldc\_<i>, fconst\_<f>, dconst\_<d>*
- Gain access to more local variables using a wider index, or to a larger immediate operand
  - *wide*

# Example – Constants



```
void useManyNumeric() {  
    int i = 100;  
    int j = 1000000;  
    long l1 = 1;  
    long l2 = 0xffffffff;  
    double d = 2.2;  
    ...do some calculations...  
}
```



```
Method void useManyNumeric()  
0 bipush 100 // Push a small int with bipush  
2 istore_1  
3 ldc #1 // Push int constant 1000000  
5 istore_2  
6 lconst_1 // A tiny long value uses short, fast lconst_1  
7 lstore_3  
8 ldc2_w #6 // Push long 0xffffffff (that is, an int -1)  
11 lstore 5  
13 ldc2_w #8 // Push double constant 2.200000  
16 dstore 7  
...do those calculations...
```

# Instruction set – Arithmetics



- Add
  - *iadd, ladd, fadd, dadd*
- Subtract
  - *isub, lsub, fsub, dsub*
- Multiply
  - *imul, lmul, fmul, dmul*
- Divide
  - *idiv, ldiv, fdiv, ddiv*
- Remainder
  - *irem, lrem, frem, drem*
- Negate
  - *ineg, lneg, fneg, dneg*
- Shift
  - *ishl, ishr, iushr, lshl, lshr, lushr*
- Bitwise OR
  - *ior, lor*
- Bitwise AND
  - *iand, land*
- Bitwise exclusive OR
  - *ixor, lxor*
- Local variable increment
  - *iinc*
- Comparison
  - *dcmpg, dcmpl, fcmpg, fcmpl, lcmp*

# Example – Arithmetics



```
int align2grain(int i, int grain) {  
    return ((i + grain-1) & ~grain-1));  
}
```



Method `int align2grain(int,int)`

```
0 iload_1  
1 iload_2  
2 iadd  
3 iconst_1  
4 isub  
5 iload_2  
6 iconst_1  
7 isub  
8 iconst_m1  
9 ixor  
10 iand  
11 ireturn
```

# Instruction set – Execution control



- Conditional branch
  - *ifeq, iflt, ifle, ifne, ifgt, ifge, ifnull, ifnonnull, if\_icmpneq, if\_icmpne, if\_icmplt, if\_icmpgt, if\_icmple, if\_icmpge, if\_acmpneq, if\_acmpne*
- Compound conditional branch
  - *tableswitch, lookupswitch*
- Unconditional branch
  - *goto, goto\_w, jsr, jsr\_w, ret*

# Example – Comparison



```
int lessThan100(double d) {  
    if (d < 100.0) {  
        return 1;  
    } else {  
        return -1;  
    }  
}
```



Method int lessThan100(double)

```
0 dload_1  
1 ldc2_w #4    // Push double constant 100.0  
4 dcmpg       // Push 1 if d is NaN or d \> 100.0;  
              // push 0 if d == 100.0  
5 ifge 10     // Branch on 0 or 1  
8  iconst_1  
9  ireturn  
10 iconst_m1  
11 ireturn
```

# Instruction set – Type conversions



- Widening numeric conversions
  - $i2l, i2f, i2d, l2f, l2d, f2d$
- Narrowing numeric conversions
  - $i2b, i2c, i2s, l2i, f2i, f2l, d2i, d2l, d2f$

# Example – Type conversion



```
void sspin() {  
    short i;  
    for (i = 0; i < 100; i++) {  
        ;           // Loop body is empty  
    }  
}
```

```
Method void sspin()  
0  iconst_0  
1  istore_1  
2  goto 10  
5  iload_1      // The short is treated as though an int  
6  iconst_1  
7  iadd  
8  i2s          // Truncate int to short  
9  istore_1  
10 iload_1  
11 bipush 100  
13 if_icmplt 5  
16 return
```



# Instruction set – Calling a method



- *invokevirtual* 
  - invokes an instance method of an object, dispatching on the (virtual) type of the object. This is the normal method dispatch in the Java programming language.
- *invokeinterface*
  - invokes a method that is implemented by an interface, searching the methods implemented by the particular runtime object to find the appropriate method.
- *invokespecial* 
  - invokes an instance method requiring special handling, whether an instance initialization method, a private method, or a superclass method.
- *invokestatic*
  - invokes a class (static) method in a named class.
- *invokedynamic*
  - invokes a method obtained by calling a bootstrap method

# Example – Calling a virtual method



```
int add12and13() {  
    return addTwo(12, 13);  
}
```



```
Method int add12and13()  
0  aload_0    // Push local variable 0 (this)  
1  bipush 12  // Push int constant 12  
3  bipush 13  // Push int constant 13  
5  invokevirtual #4  // Method Example.addtwo(II)I  
8  ireturn    // Return int on top of operand stack; it is  
           // the int result of addTwo()
```

# Type specification



<i>BaseType Character</i>	Type	Interpretation
B	byte	signed byte
C	char	Unicode character
D	double	double-precision floating-point value
F	float	single-precision floating-point value
I	int	integer
J	long	long integer
L Classname;	reference	an instance of class <classname>
S	short	signed short
Z	boolean	true or false
[	reference	one array dimension

## Examples:

- **double d[][][]**  $\Rightarrow$  [[[D
- **Object mymethod(int i, double d, Thread t)**  
 $\Rightarrow$  (IDLjava/lang/Thread;)Ljava/lang/Object;

# Example – Calling a static method



```
int add12and13() {  
    return addTwoStatic(12, 13);  
}
```



```
Method int add12and13()  
0 bipush 12  
2 bipush 13  
4 invokestatic #3 // Method Example.addTwoStatic(II)I  
7 ireturn
```

# Example – Calling a special method

...

```
class Near {  
    int it;  
    public int getItNear() {  
        return getIt();  
    }  
    private int getIt() {  
        return it;  
    }  
}
```

```
class Far extends Near {  
    int getItFar() {  
        return super.getItNear();  
    }  
}
```



```
Method int getItNear()  
0 aload_0  
1 invokespecial #5  
    // Method Near.getIt()  
4 ireturn
```

```
Method int getItFar()  
0 aload_0  
1 invokespecial #4  
    // Method  
    // Near.getItNear()  
4 ireturn
```

# Invokedynamic



```
static void test() throws Throwable {
    // THE FOLLOWING LINE IS PSEUDOCODE FOR A JVM INSTRUCTION
    InvokeDynamic[#bootstrapDynamic].baz("baz arg", 2, 3.14);
}

private static void printArgs(Object... args) {
    System.out.println(java.util.Arrays.deepToString(args));
}

private static CallSite bootstrapDynamic(MethodHandles.Lookup caller,
                                         String name, MethodType type) {

    MethodHandles.Lookup lookup = MethodHandles.lookup();
    Class thisClass = lookup.lookupClass(); // (who am I?)
    MethodHandle printArgs = lookup.findStatic(thisClass,
        "printArgs", MethodType.methodType(void.class, Object[].class));

    // ignore caller and name, but match the type:
    return new ConstantCallSite(printArgs.asType(type));
}
```

# Instruction set – Instance manipulation

---



- Create a new class instance
  - *new*
- Access fields of classes (static fields, known as class variables) and fields of class instances (non-static fields, known as instance variables)
  - *getfield, putfield, getstatic, putstatic*
- Check properties of class instances or arrays
  - *instanceof, checkcast*

# Example – Instance creation



```
Object create() {  
    return new Object();  
}
```



```
Method java.lang.Object create()  
0 new #1 // Class java.lang.Object  
3 dup  
4 invokespecial #4 // Method java.lang.Object.<init>()V  
7 areturn
```

# Example – Attribute access



```
void setIt(int value) {  
    i = value;  
}  
int getIt() {  
    return i;  
}
```



```
Method void setIt(int)  
0  aload_0  
1  iload_1  
2  putfield #4           // Field Example.i I  
5  return  
Method int getIt()  
0  aload_0  
1  getfield #4           // Field Example.i I  
4  ireturn
```

# Instruction set – Array manipulation



- Create a new array
  - *newarray, anewarray, multianewarray*
- Load an array component onto the operand stack
  - *baload, caload, saload, iaload, laload, faload, daload, aaload*
- Store a value from the operand stack as an array component
  - *bastore, castore, sastore, iastore, lastore, fastore, dastore, aastore*
- Get the length of array
  - *arraylength*

# Example – Array (primitive type)



```
void createBuffer() {  
    int buffer[];          int bufsz = 100;  
    int value = 12;         buffer = new int[bufsz];  
    buffer[10] = value;    value = buffer[11];  
}
```



```
Method void createBuffer()  
0 bipush 100    // Push int constant 100 (bufsz)  
2 istore_2      // Store bufsz in local variable 2  
3 bipush 12     // Push int constant 12 (value)  
5 istore_3      // Store value in local variable 3  
6 iload_2       // Push bufsz...  
7 newarray int   // ...and create new array of int of that length  
9 astore_1       // Store new array in buffer  
10 aload_1       // Push buffer  
11 bipush 10     // Push int constant 10  
13 iload_3       // Push value  
14 iastore       // Store value at buffer[10]  
15 aload_1       // Push buffer  
16 bipush 11     // Push int constant 11  
18 iaload        // Push value at buffer[11]...  
19 istore_3      // ...and store it in value  
20 return
```

# Example – Array (reference)



```
void createThreadArray() {  
    Thread threads[];  
    int count = 10;  
    threads = new Thread[count];  
    threads[0] = new Thread();  
}
```



```
Method void createThreadArray()  
0 bipush 10           // Push int constant 10  
2 istore_2           // Initialize count to that  
3 iload_2            // Push count, used by anewarray  
4 anewarray class #1 // Create new array of class Thread  
7 astore_1            // Store new array in threads  
8 aload_1             // Push value of threads  
9 iconst_0            // Push int constant 0  
10 new #1             // Create instance of class Thread  
13 dup                // Make duplicate reference...  
14 invokespecial #5   // ...to pass to instance initialization  
                      // method Method java.lang.Thread.<init>()V  
17 aastore            // Store new Thread in array at 0  
18 return
```

# Example – Array (multidimensional)



```
int[][][] create3DArray() {  
    int grid[][][];  
    grid = new int[10][5][];  
    return grid;  
}
```



Method `int create3DArray()[][][]`

```
0 bipush 10           // Push int 10 (dimension one)  
2 iconst_5          // Push int 5 (dimension two)  
3 multianewarray #1 dim #2 // Class [[[I, a three  
                                // dimensional int array; only  
                                // create first two dimensions  
7 astore_1            // Store new array...  
8 aload_1             // ...then prepare to return it  
9 areturn
```

# Instruction set – Stack manipulation

---



- *pop, pop2, dup, dup2, dup\_x1, dup2\_x1, dup\_x2, dup2\_x2, swap*

# Example – Array (multidimensional)



```
public long nextIndex() {  
    return index++;  
}  
private long index = 0;
```



```
Method long nextIndex()  
0  aload_0    // Push this  
1  dup       // Make a copy of it  
2  getfield #4   // One of the copies of this is consumed  
           // pushing long field index,  
           // above the original this  
5  dup2_x1    // The long on top of the operand stack is  
           // inserted into the operand stack below the  
           // original this  
6  lconst_1    // Push long constant 1  
7  ladd       // The index value is incremented...  
8  putfield #4   // ...and the result stored back in the field  
11 lreturn    // The original value of index is left on top  
           // of the operand stack, ready to be returned
```

# Instruction set – Monitors



- *monitorenter*
- *monitorexit*

# Example – Exceptions (throw)



```
void cantBeZero(int i) throws TestExc {  
    if (i == 0) {  
        throw new TestExc();  
    }  
}
```



```
Method void cantBeZero(int)  
0 iload_1          // Push argument 1 (i)  
1 ifne 12          // If i==0, allocate instance and throw  
4 new #1           // Create instance of TestExc  
7 dup              // One reference goes to the constructor  
8 invokespecial #7 // Method TestExc.<init>()V  
11 athrow           // Second reference is thrown  
12 return           // Never get here if we threw TestExc
```

# Example – Exceptions (catch)



```
void catchOne() {  
    try {  
        tryItOut();  
    } catch (TestExc e) {  
        handleExc(e);  
    }  
}
```



```
Method void catchOne()  
0  aload_0          // Beginning of try block  
1  invokevirtual #6 // Method Example.tryItOut()V  
4  return           // End of try block; normal return  
5  astore_1          // Store thrown value in local variable 1  
6  aload_0          // Push this  
7  aload_1          // Push thrown value  
8  invokevirtual #5 // Invoke handler method:  
                   // Example.handleExc(LTestExc;)V  
11 return           // Return after handling TestExc
```

Exception table:

From	To	Target	Type
0	4	5	Class TestExc

# Example – Exceptions (nested)



```
void nestedCatch() {  
    try {  
        try {  
            tryItOut();  
        } catch (TestExc1 e) {  
            handleExc1(e);  
        }  
    } catch (TestExc2 e) {  
        handleExc2(e);  
    }  
}
```



Method void nestedCatch()

.....  
.....

Exception table:

From	To	Target	Type
0	4	5	Class TestExc1
0	11	12	Class TestExc2

# Instruction set – Exceptions

---



- Throwing an exception
  - *athrow*
- Try-catch declaration
  - Via special *exception table* associated with a method
- Finally
  - Implemented by the compiler

# Example – Monitors



```
void onlyMe(Foo f) {  
    synchronized(f) {  
        doSomething();  
    }  
}
```

Method void onlyMe(Foo)

```
0  aload_1           // Push f  
1  astore_2          // Store it in local variable 2  
2  aload_2           // Push local variable 2 (f)  
3  monitorenter      // Enter the monitor associated with f  
4  aload_0           // Holding the monitor, pass this and...  
5  invokevirtual #5 // ...call Example.doSomething()V  
8  aload_2           // Push local variable 2 (f)  
9  monitorexit       // Exit the monitor associated with f  
10 return            // Return normally  
11  aload_2          // In case of any throw, end up here  
12  monitorexit      // Be sure to exit monitor...  
13  athrow            // ...then rethrow the value to the invoker
```



Exception table:

From	To	Target	Type
4	8	11	any