Java and the JVM

Martin Schöberl

Overview

- History and Java features
- Java technology
- The Java language
- A first look into the JVM
- Disassembling of .class files

History of a *Young* Java

- 1992 *Oak* for a PDA on a SPARC (*7)
- 1995 Official release as Java – Internet
- 1997 picoJava – Sun’s Java processor
- 1998 RTSJ specification start as JSR-01
- 1999 split into J2SE and J2EE
- 2000 J2ME
- 2002 RTSJ final release
- 2002 first version of JOP ;-)

Java features

- Simple and object oriented
  - *Look and feel* of C
  - Simplified object model with single inheritance
- Portability
  - Java compiler generates bytecodes
  - Runtime systems for various platforms
  - Size and behavior of basic data types defined
  - *Write once, run/debug anywhere*

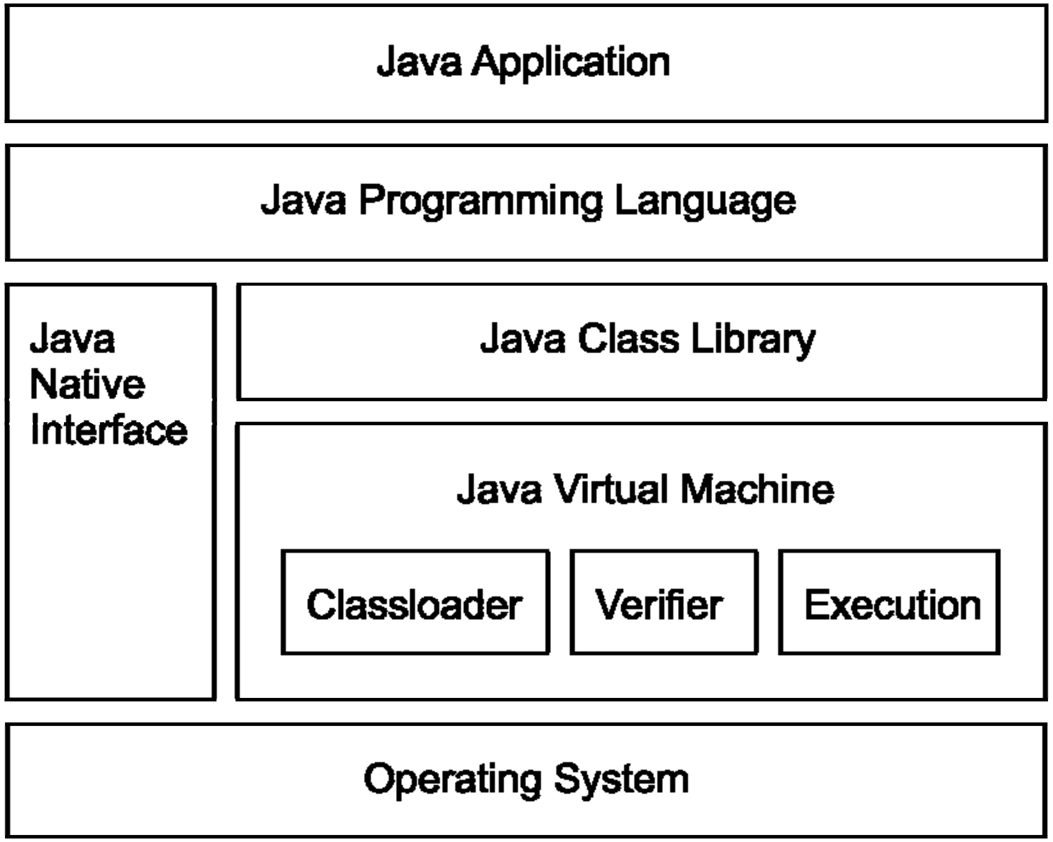
Java features cont.

- Availability
  - Windows, Linux, Solaris,...
  - Embedded systems
  - Compiler and runtime are free
  - Free IDEs: Eclipse, Netbeans
- Library
  - Rich class library
  - Part of the definition
  - Standard GUI toolkit

Java features cont.

- Built-in model for concurrency
  - Threads at the language level
  - Synchronization
  - Libraries are thread-safe
- Safety
  - No Pointer!
  - Extensive compile-time checking
  - Runtime checking
  - Automatic memory management – GC

Java system overview



Java Technology

- The Java programming language
- The library (JDK)
- The Java virtual machine (JVM)
  - Instruction set
  - Binary format
  - Verification

Java Primitive Data Types

`boolean`	either `true` or `false`
`char`	16-bit Unicode character (unsigned)
`byte`	8-bit integer (signed)
`short`	16-bit integer (signed)
`int`	32-bit integer (signed)
`long`	64-bit integer (signed)
`float`	32-bit floating-point (IEEE 754-1985)
`double`	64-bit floating-point (IEEE 754-1985)

Objects

- Everything belongs to an object (or a class)
  - No *global* variables
- Namespace for objects
- Single inheritance
- Interfaces
- Allocated on the heap
- Shared among threads
- No `free()` – garbage collector

What is a Virtual Machine?

- A virtual machine (VM) is an *abstract* computer architecture
- Software on top of a real hardware
- Can run the same application on different machines where the VM is available

The Java Virtual Machine

- An abstract computing machine that executes bytecode programs
  - An instruction set and the meaning of those instructions – the *bytecodes*
  - A binary format – the *class file* format
  - An algorithm to *verify* the class file

JVM cont.

- Runtime environment for Java
- Implementation NOT defined
- Runs Java .class files
- Has to conform to Sun‘s specification

Implementations of the JVM

- Interpreter
  - Simple, compact
  - Slow
- Just-in-time compilation
  - State-of-the-art for desktop/server
  - Too resource consuming in embedded systems
- Batch compilation
- Hardware implementation
  - Our topic!

JVM Data Types

`reference`	Pointer to an object or array
`int`	32-bit integer (signed)
`long`	64-bit integer (signed)
`float`	32-bit floating-point (IEEE 754-1985)
`double`	64-bit floating-point (IEEE 754-1985)

- No `boolean`, `char`, `byte`, and `short` types
  - Stack contains only 32-bit and 64-bit data
  - Conversion instructions

Memory Areas for the JVM

- Method area
  - Class description
  - Code
  - Constant pool
- Heap
  - Objects and Arrays
  - Shared by all threads
  - Garbage collected

Memory Areas for the JVM

- Stack
  - Thread private
  - Logical stack that contains:
    - Invocation frame
    - Local variable area
    - Operand stack
  - Not necessary a *single* stack
  - Local variables and operand stack are accessed frequently

JVM Instruction Set

- 32 (64) bit stack machine
- Variable length instruction set
- Simple to very complex instructions
- Symbolic references
- Only relative branches

JVM Instruction Set

- Load and store
- Arithmetic
- Type conversion
- Object creation and manipulation
- Operand stack manipulation
- Control transfer
- Method invocation and return

Dissassembling Java

- Compile
  - `javac Hello.java`
- Run
  - `java Hello`
- Dissassemble
  - `javap -c Hello`

A Bytecode Example

```
public class X {

   public static void
   main(String[] args) {
        add(1, 2);
   }

   public static int
   add(int a, int b) {
        return a+b;
   }
}
```

```
public static void
   main(java.lang.String[]);
  Code:
   0:    iconst_1
   1:    iconst_2
   //Method add:(II)I
   2:    invokestatic    #2;
   5:    pop
   6:    return

public static int
   add(int,int);
  Code:
   0:    iload_0
   1:    iload_1
   2:    iadd
   3:    ireturn
```

Coding: Avoiding garbage

```
System.out.println("Result = "+i);
```

```
getstatic         #3; // Field System.out:Ljava/io/PrintStream;
new               #4; // class StringBuffer
dup
invokespecial     #5; // StringBuffer."<init>":()V
ldc               #6; // String Result =
invokevirtual     #7; // StringBuffer.append:(LString;)LStringBuffer
iload_1
invokevirtual     #8; // StringBuffer.append:(I)LStringBuffer;
invokevirtual     #9; // StringBuffer.toString:()LString;
invokevirtual     #10;// PrintStream.println:(LString;)V
```

Coding: Avoiding garbage

```
System.out.print("Result = ");
System.out.println(i);
```

```
getstatic       #3; //Field System.out:Ljava/io/PrintStream;
ldc             #4; //String Result =
invokevirtual   #5; //Method PrintStream.print:(LString;)V
getstatic       #3; //Field System.out:LPrintStream;
iload_1
invokevirtual   #6; //Method PrintStream.println:(I)V
```

Java for Embedded Systems?

+ Simpler than C/C++
+ Safer than C/C++
+ Threads are part of the language
- Interpreting JVM is slow
- JIT needs a lot of memory
- GC and real-time?

Summary Java/JVM

- Java language definition
- Class library
- The Java virtual machine (JVM)
  - An instruction set – the *bytecodes*
  - A binary format – the *class file*
  - An algorithm to *verify* the class file

Summary Java Features

- Safe OO Language
  - No pointers
  - Type-safety
  - Garbage Collection
- Built in model for concurrency
- Platform *independent*
- Very rich *standard* library

More Information

- Java
  - James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*, Addison-Wesley, 2000, JavaSpec.
- JVM
  - Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1999, JVMSpec.