

Threading problems in Java - for NetRexx and BSF4ooRexx

René Vincent Jansen
34th International Rexx Language
Symposium, Almere, 20230516

fall semester of that year. (The problem
Quintuple - later dubbed "The Dining
y Tony Hoare - was the examination
e end of that semester.) The rate
EWD-numbers increased was in
not a measure of my productivity:
numbers when I started on manu-
any of them were not completed.
r in Eindhoven I suffered from a
writer's block. Everything I wrote disappeared
before completion into the wastepaper basket,
until I discovered the cause: if I geared my
text to my former colleagues of the Mathematical
Centre in Amsterdam I realized halfway that my

Contents

1 Tiered Compilation

Tiered compilation delivers problems for classic problem determination

2 Race Conditions

Race Conditions exist where access to memory in different threads is not protected ('synchronised')

3 Thread Starvation

Threads can deadlock due to the synchronization mechanism

4 Deadlocks

Unstructured Locking leads to Deadlocks

No 1

Tiered Compilation

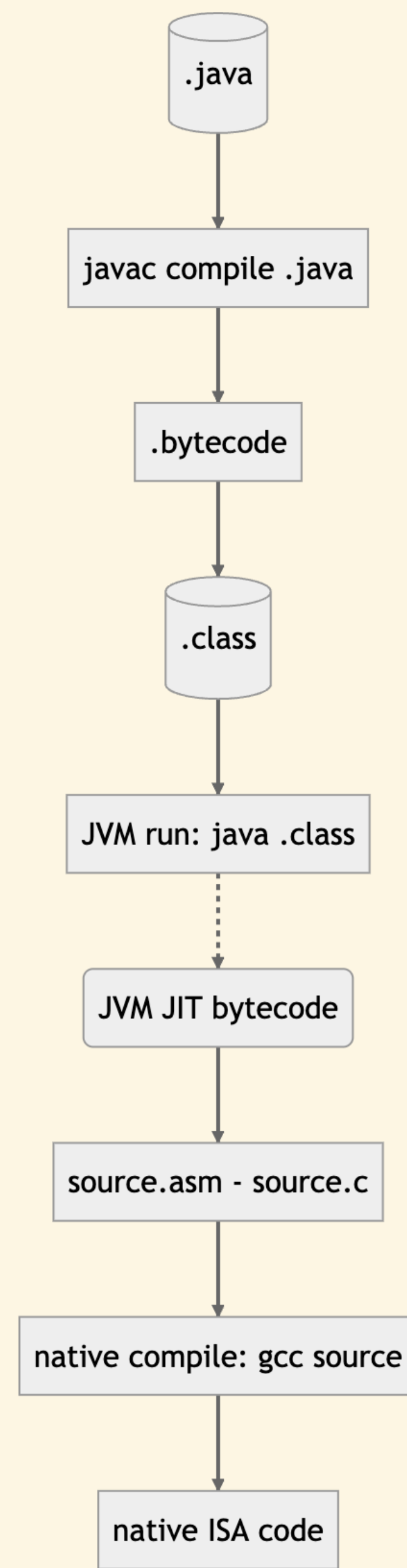
Source, JVM ByteCode, Native Code, Hardware Instruction Set Architectures

Tiered Compilation

Java : compile --> bytecode,
run bytecode in VM
compile --> native

Code Flow

only two files involved: java and class, but many transitions



```
CLASS HELLO {  
    PUBLIC STATIC VOID MAIN(STRING ARGS[]){  
        SYSTEM.OUT.PRINTLN("HELLO JAVA");  
    }  
}
```

Hello World

Looking around

- Compile it: `javac hello.java`
- see the bytecode in the classfile; plain and hex
- disassemble the bytecode with `javap -c hello.class`
- disassemble the bytecode with `gnoloo hello.class`
 - look at the generated `hello.j`
- execute it with the JVM: `java hello`

See the native compilation

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly hello
```

with hex: cafebabe

with hsdis-aarch64.dylib

(download from <https://chriswhocodes.com/hsdis/> or build from openJDK source)

No 2

Race Conditions

Where Threading makes a mess of your memory

```
OPTIONS BINARY
CLASS RACECONDITION
```

```
PROPERTIES STATIC
SHAREDVARIABLE = 0
LOCK=OBJECT()
```

```
METHOD MAIN(ARGS=STRING[]) STATIC
```

```
  THREAD1 = THREAD(INCREMENT())
  THREAD2 = THREAD(INCREMENT())
```

```
  THREAD1.START()
  THREAD2.START()
```

```
  THREAD1.JOIN()
  THREAD2.JOIN()
```

```
  SAY "SHARED VARIABLE = "SHAREDVARIABLE
```

```
CLASS INCREMENT IMPLEMENTS RUNNABLE
```

```
  METHOD RUN()
```

```
    LOOP PROTECT RACECONDITION.LOCK FOR 100000
```

```
      RACECONDITION.SHAREDVARIABLE = RACECONDITION.SHAREDVARIABLE +
```

```
    END
```

Two Threads

Threading can cause race conditions

- We start two threads that sum to
- Compile and Run, see that the answer is not the same two times in a row
- How to solve this problem
 - Monitors
 - Critical Sections
- We see the monitor construct in the JVM Bytecode

No 3

Thread Starvation

or Dining Philosophers

The Edsger Dijkstra Problem

```
    OBJECT[NUMPHILOSOPHERS];  
    PHILOSOPHERS; I++) {  
        THINK();  
  
        = NEW PHILOSOPHER[NUMPHILOSOPHERS];  
        PHILOSOPHERS; I++) {  
            PHILOSOPHER(I, CHOPSTICKS[I], CHOPSTICKS[(I + 1) % NUMPHILOSOPHERS]);  
        };  
    };  
};  
CLASS
```

```
    OBJECT LEFTCHOPSTICK, OBJECT RIGHTCHOPSTICK) {  
        CHOPSTICK;  
        RIGHTCHOPSTICK;  
    };  
};
```

```
    THINK();  
    PICKUPCHOPSTICKS();  
    EAT();  
    PUTDOWNCHOPSTICKS();  
};
```

No 4

Deadlocks

Like database locks but more fierce and hard to diagnose

```

CLASS DEADLOCK
PROPERTIES CONSTANT
LOCK1 = OBJECT()
LOCK2 = OBJECT()

METHOD MAIN(ARGS=STRING[]) STATIC
  THREAD1 = THREAD(TASK1())
  THREAD2 = THREAD(TASK2())
  THREAD1.START()
  THREAD2.START()
  THREAD1.JOIN();
  THREAD2.JOIN();

CLASS Task1 IMPLEMENTS RUNNABLE
METHOD RUN()
  DO PROTECT DEADLOCK.LOCK1
    SAY "TASK1 ACQUIRED LOCK1"
    SAY "TASK1 WAITING FOR LOCK2"
    DO PROTECT DEADLOCK.LOCK2
      SAY "TASK1 ACQUIRED LOCK2"
    END
  END
END

CLASS Task2 IMPLEMENTS RUNNABLE
METHOD RUN()
  DO PROTECT DEADLOCK.LOCK2
    SAY "TASK2 ACQUIRED LOCK2"
    SAY "TASK2 WAITING FOR LOCK1"
    DO PROTECT DEADLOCK.LOCK1
      SAY "TASK2 ACQUIRED LOCK1"
    END
  END
END

```

A clear case of Deadlock

The sequence of events counts

Deadlock breaking

- run with a profiler
 - `java -XX:HeapDumpPath=/Users/rvjansen/test/javathreading -javaagent:/Users/rvjansen/apps/glowroot/glowroot.jar Deadlock`
- look at JVM - thread dump
- when unclear: actually dump the JVM memory image
- format the dump
 - `jhat heap-dump-20230420-125101.hprof <== subst actual name`