Behind you…
A virtual machine!

Yann-Gaël Guéhéneuc
guehene@emn.fr


École des Mines
de Nantes, France


Object Technology
International, Inc., Canada

What is computer science?

- Algorithm computation
- What is an algorithm?
  - A set of mathematical operations
  - A set of instructions acting on data
- What is a computer?
  - A machine computing an algorithm

Computing machine? (1/5)

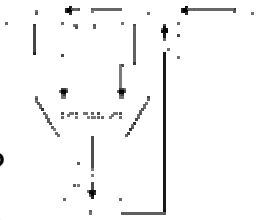
- Turing machine:
  - 1936
  - A finite set of states, of instructions, and a finite alphabet
  - An infinitely long tape
  - Practicable?
    - Enigma
    - `http://www.wikipedia.org/wiki/Enigma`

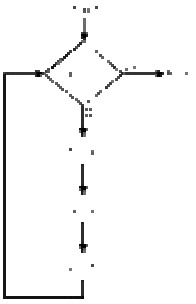
Computing machine? (2/5)

- λ-calculus et π-calculus:
  - 1934 (1936: Undecidable problem!)
  - Everything is function
  - Everything is parameter passing
  - Simple, elegant, mathematical
  - Practicable?
    - Lisp, Scheme, Haskell…

Computing machine? (3/5)

- Register machine:
  - Registers, data flow, instructions, controller





  - Intuitive
  - Practicable?
    - Processors

Computing machine? (4/5)

- Stack machine:
  - A stack, instructions, controller

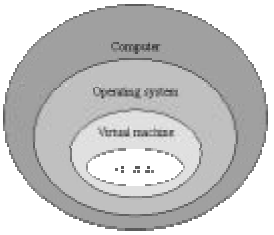
  - Recursive
  - Practicable?
    - Virtual machines

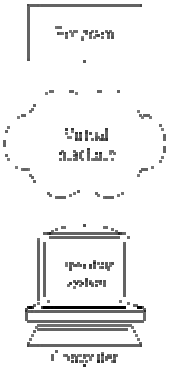
Computing machine? (5/5)

- A virtual machine:
  - A layer standing up between the program and the operating system
  - An interpreter interpreting a program for an operating system
  - A compiler compiling in real-time a program for an operating system

What is it?

- A virtual machine is:





- A virtual machine is not:
  - Without interest!

Why? (1/3)

n languages + *m* operating systems =
n * m compilers

n languages + virtual machine + *m* operating systems =
n compilers
(+ ***m*** free virtual machines…)

Why? (2/3)

- Simplicity of implementation
  - Well…
- Availability
- Maintainability

Why not ? (3/3)

- Performances

The JVM (1/3)

- The Java Virtual Machine:
  - Executes Java byte-codes
  - Loads the main class of the program
  - Runs the `main(…)` method
  - Multi-threading, instantiation, garbage collection…

The JVM (2/3)

- The JVM contains:
  - By method:
    - A frame
  - By class:
    - A constant pool
  - By thread:
    - A stack
    - A register
  - For all threads:
    - A heap
    - A « method » area
    - A « native method » stack

The JVM (3/3)

- The JVM byte-codes:
  - Object model related to object model of the Java programming language
  - Typed op-codes:

```
public int computeGCD(
  final int a,
  final int b) {

  if (b == 0) {
    return a;
  }
  else {
    return
      computeGCD(b, a % b);
  }
}
```

```
Method int computeGCD(int, int)
 0 iload_1
 1 ifne 6
 4 iload_0
 5 ireturn
 6 iload_1
 7 iload_0
 8 iload_1
 9 irem
10 invokestatic #25
     <Method int computeGCD(int, int)>
13 ireturn
```

What else?

- The JVM offers:
  - Sandbox model
  - Class loader mechanism
  - Garbage collector algorithm
  - Security features
  - Java Native Interface
  - Java Platform Debug Architecture
  - Java Virtual Machine Profiler Interface

JNI (1/3)

- Java Native Interface
  - Interface between the JVM and C
- Contrast
  - Sandbox model
  - Performance issue
  - Legacy code

JNI (2/3)

```
package emn.course.vm;

public final class GCD_C {
    public native int computeGCD(final int a, final int b);
    static {
        System.loadLibrary("GCD_C_Impl");
    }
    public static void main(final String[] args) {
        final GCD_C myJNICallToGCD = new GCD_C();
        System.out.println(myJNICallToGCD.computeGCD(6, 18));
    }
}
```


JNI (3/3)

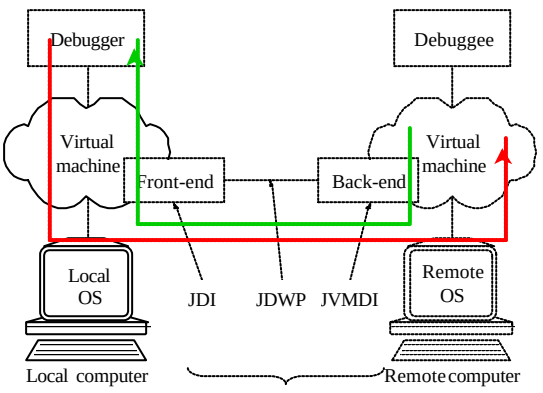
```
#include "emn_course_vm_GCD_0005fC.h"
#include <stdio.h>

jint JNICALL Java_emn_course_vm_GCD_1C_computeGCD(
                    JNIEnv *env, jobject obj, jint a, jint b) {
        if (b == 0) {
            return a;
        }
        else {
            return Java_emn_course_vm_GCD_1C_computeGCD(
                                            env, obj, b, a % b);
        }
}
```

JPDA (1/7)

- Java Platform Debug Architecture
  - To debug / to control a program running in the JVM
  - Three levels:
    - Java Virtual Machine Debug Interface
    - Java Debug Wire Protocol
    - Java Debug Interface

JPDA (2/7)



JPDA

JPDA (3/7)

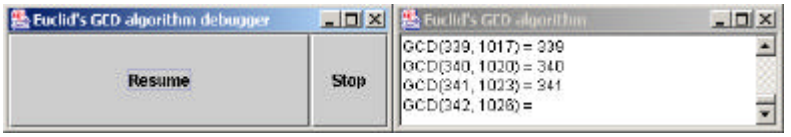
- Experimentations:
  - Development environments:
    - IBM Eclipse
    - Borland JBuilder
    - Sun Forte
  - Research tools:
    - CPPROFJ [Hall, 2002]
    - Caffeine [Guéhéneuc, 2002]

JPDA (4/7)

```
final VirtualMachineManager vmManager =
    Bootstrap.virtualMachineManager();
final LaunchingConnector launchingConnector =
    vmManager.defaultConnector();
final Map arguments = launchingConnector.defaultArguments();
final Iterator iterator = arguments.values().iterator();
while (iterator.hasNext()) {
    final Argument argument = (Argument) iterator.next();
    if (argument.name().equals("main")) {
        argument.setValue("emn.course.vm.GCD");
    }
}
final VirtualMachine vm = launchingConnector.launch(arguments);
vm.resume();
```

JPDA (5/7)

JPDA demo?

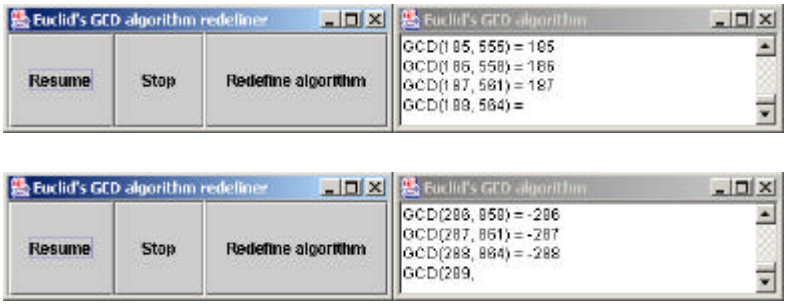


JPDA (6/7)

- Other features include:
  - Full-speed debugging (?)
  - HotSwap
  - Filters
  - Debugging for other languages
  - …

JPDA (7/7)

HotSwap Demo?



JVMPI (1/4)

- Java Virtual Machine Profiler Interface
  - To know the execution profile of a program running in a JVM

JVMPI (2/4)

```
JNIEXPORT jint JNICALL JVM_OnLoad(JavaVM *jvm, char *options, void *reserved) {
    fprintf(stderr, "Initializing the profiler for the Course on VMs.\n");

    // Get JVMPI interface pointer
    if (((*jvm)->GetEnv(jvm, (void **)&jvmpi_interface, JVMPI_VERSION_1)) < 0) {
        fprintf(stderr, "Initialization error in obtaining JVMPI interface
   pointer.\n");
        return JNI_ERR;
    }

    // Initialize the JVMPI interface
    jvmpi_interface->NotifyEvent = NotifyEvent;

    // Enable "class load"- and "thread start"-event notification
    jvmpi_interface->EnableEvent(JVMPI_EVENT_CLASS_LOAD, NULL);
    jvmpi_interface->EnableEvent(JVMPI_EVENT_THREAD_START, NULL);

    fprintf(stderr, "Initialization done.\n\n");
    return JNI_OK;
}
```


JVMPI (3/4)

```
void NotifyEvent(JVMPI_Event *event) {
    const char *class_name;
    const char *thread_name;
    switch(event->event_type) {
        case JVMPI_EVENT_CLASS_LOAD:
            class_name = event->u.class_load.class_name;
            if (!((class_name[0] == 'j' && class_name[1] == 'a' &&
                   class_name[2] == 'v' && class_name[3] == 'a')
                  ||
                  (class_name[0] == 's' && class_name[1] == 'u' &&
                   class_name[2] == 'n'))) {
                fprintf(stderr, "> Class loaded: %s\n", class_name);
            }
            break;
        case JVMPI_EVENT_THREAD_START:
            thread_name = event->u.thread_start.thread_name;
            fprintf(stderr, "> Thread started: %s\n", thread_name);
            break;
    }
}
```

JVMPI (4/4)

Demo ?

```
Initializing the profiler for the Course on VMs.
Initialization done.

> Thread started: Signal Dispatcher
> Thread started: CompileThread0
> Class loaded: com.sun.rsajca.Provider
> Class loaded: com.sun.rsajca.Provider$1
> Class loaded: emn.course.vm.GCD
> Thread started: AWT-EventQueue-0
> Thread started: SunToolkit.PostEventQueue-0
> Thread started: AWT-Windows
> Class loaded: emn.course.vm.GCD$1
> Thread started: TimerQueue
> Thread started: Thread-0
```


And what else?

- Lisp / Scheme
- Pascal
- Prolog
- Squeak
- C# and the .Net platform
- Proof-carrying code
- Strongly-typed intermediate languages

C# and the .Net platform (1/3)

- .Net platform:
  - New
  - Mature
  - Buzz words

C# and the .Net platform (2/3)

- .Net platform:
  - More generic object model
  - Un-typed op-codes
- Need a specialist?

  ⇒ Marc Ségura-Devillechaise

C# and the .Net platform (3/3)

```
.method public hidebysig static int32  ComputeGCD(int32 a,
                                                   int32 b) cil managed
{
  IL_0000:  ldarg.1
  IL_0001:  brtrue.s   IL_0007

  IL_0003:  ldarg.0
  IL_0004:  stloc.0
  IL_0005:  br.s       IL_0013

  IL_0007:  ldarg.1
  IL_0008:  ldarg.0
  IL_0009:  ldarg.1
  IL_000a:  rem
  IL_000b:  call       int32 GCD::ComputeGCD(int32, int32)
  IL_0010:  stloc.0
  IL_0011:  br.s       IL_0013

  IL_0013:  ldloc.0
  IL_0014:  ret
} // end of method GCD::ComputeGCD
```


That's all folks!

- Thank you so much for your attention!

- Questions?
- Comments?