Core Java

3. Java Execution Model

System Model

- Processor supports an Instruction Set Architecture (ISA)
- Examples of ISA: x86, x64, IA-64, Sparc, PPC, ARM
- Programs are a mix of relocatable ISA instructions
- Programs also depend on native libraries
- Dynamic Linker (*ld.so*) loads programs into memory

System Model

- Running natively can give good performance
- Disadvantage is portability
- Should I compile at all?
- I write it so that I can compile it in both x86 and Sparc with native performance on both? (*C* programs)
- But can I compile a program with x86 instructions and run it on Sparc with near native performance?

Traditional Model

- Preprocessing
- Compile to Assembly
- Assemble
- Link
- Dynamic Link
- Run

Compile to Assembly

- Front End (Language Specific)
- Back End (Machine Specific)
- Output of Front End is sent to Back End for Assembly Code Generation
- Can't we keep the Output of the Front End and generate Assembly Code on the Fly?
- Yes we can!

Just-in-Time Compilation

- Store simple operations like addition as machine independent
- Convert them into native code when running
- Java follows the same: Compilation into bytecode (*.class* files) and subsequent Execution
- The Java Virtual Machine is the one which runs your Java program

Advantages

- ISA/Processor independent
- Many operating systems offer compatible libraries/interfaces: *POSIX*
- Java bytecode is actually language independent (and *.NET* is not the only platform which offers this!)
- Write once, run everywhere!
- The Java Virtual Machine specification

Myths

- Java is slow: Happens when the overhead in loading Java libraries and JITC eclipses running time of program. But not always true!
- Low level implementation not possible: Java supports protected native functionality through JNI: Interact with what the OS provides you. Java's GUI interfaces with system libraries

Compilation in Java

- The Java Development Kit ships with the Command Line Java Compiler (*javac*)
- IDEs like eclipse ship with their own builtin compiler (*JDT*)
- Traditionally invoked as:

```
$ javac HelloWorld.java
```

Execution in Java

- The Java Runtime Environment ships with the Java Interpreter (*java*)
- Traditionally invoked as:

```
$ java HelloWorld
```

How do I go about this Course?

- Initially get comfortable with the language, then its' tools
- Attend Lab sessions
- Ask questions
- Read, Think, Discover!

Questions?