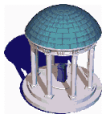


The University of North Carolina at Chapel Hill

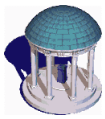
COMP 144 Programming Language Concepts

Spring 2002

Lecture 32: The Java Virtual Machine

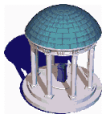
Felix Hernandez-Campos

April 12



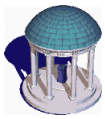
The Java Virtual Machine

- Java Architecture
  - Java Programming Language
  - Java Virtual Machine (JVM)
  - Java API
- We will use the JVM as a case study of an intermediate program representation

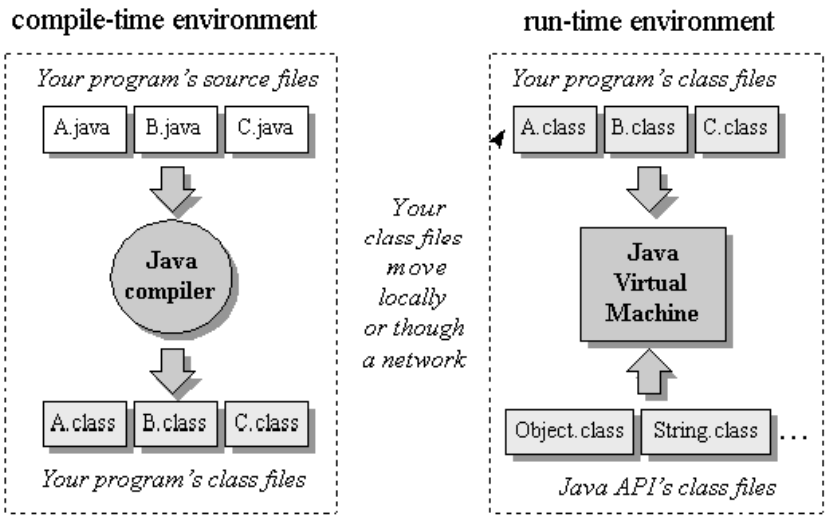


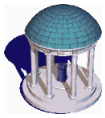
Reference

- The content of this lecture is based on *Inside the Java 2 Virtual Machine* by Bill Venners
  - Chapter 1Introduction to Java's Architecture
    - » http://www.artima.com/insidejvm/ed2/ch01IntroToJavasArchitecturePrint.html
  - Chapter 5 The Java Virtual Machine
    - » http://www.artima.com/insidejvm/ed2/ch05JavaVirtualMachine1.html
  - Interactive Illustrations
    - » http://www.artima.com/insidejvm/applets/index.html



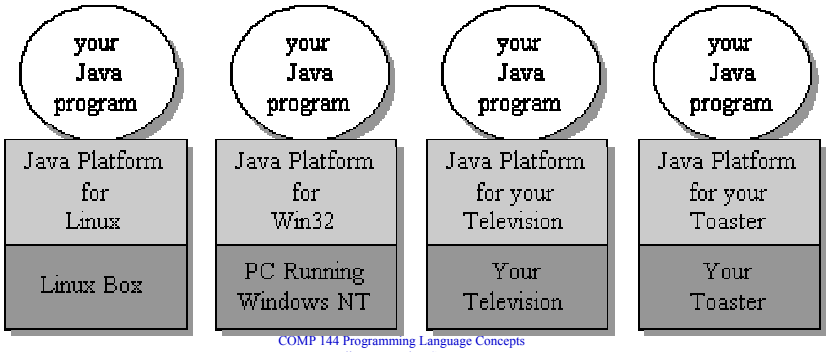
The Java Programming Environment

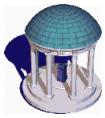




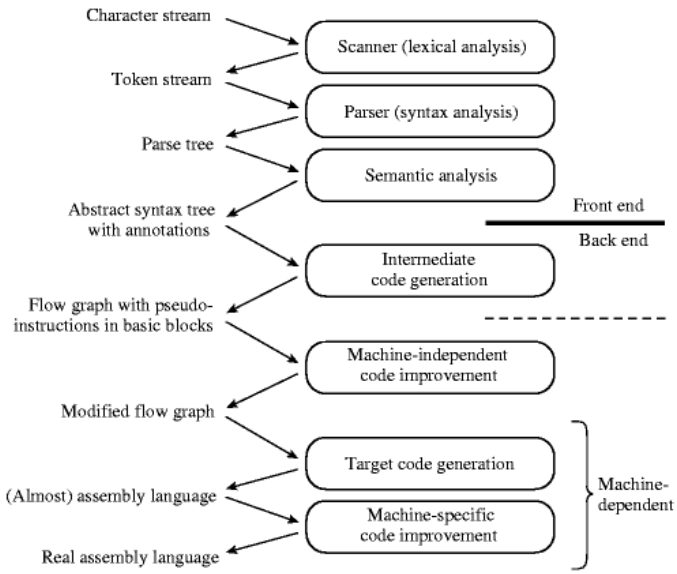
The Java Platform

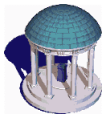
- The byte code generated by the Java front-end is an *intermediate form*
  - Compact
  - Platform-independent



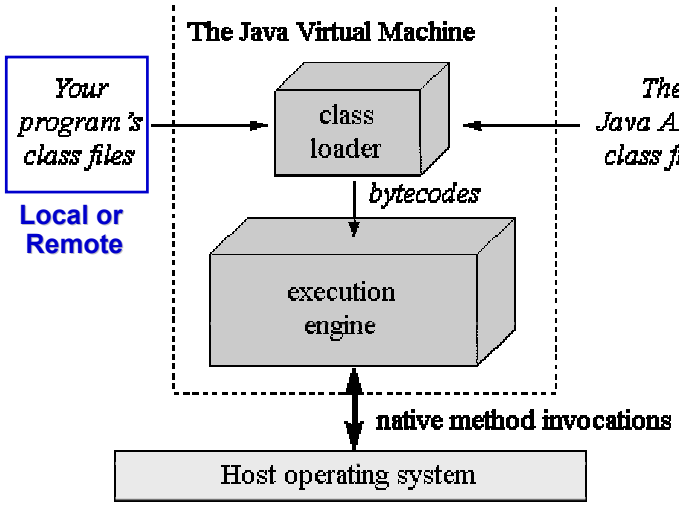


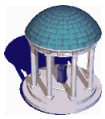
Phases of Compilation





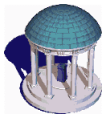
The Role of the Virtual Machime



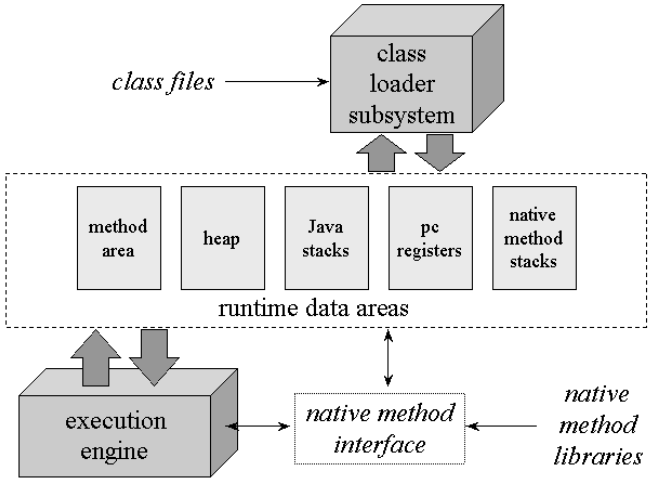


The Execution Engine

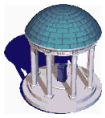
- Back-end transformation and execution
  - Simple JVM: byte code interpretation
  - *Just-in-time* compiler
    - » Method byte codes are compiled into machine code the first time they are invoked
    - » The machine code is cached for subsequent invocation
    - » It requires more memory
  - *Adaptive optimization*
    - » The interpreter monitors the activity of the program, compiling the heavily used part of the program into machine code
    - » It is much faster than simple interpretation
    - » The memory requirement is only slightly larger due to the 20%/80% rule of program execution (In general, 20% of the code is responsible for 80% of the execution)



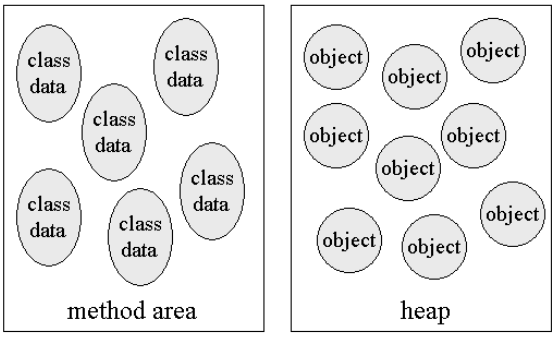
The Java Virtual Machine



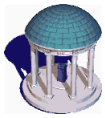
9



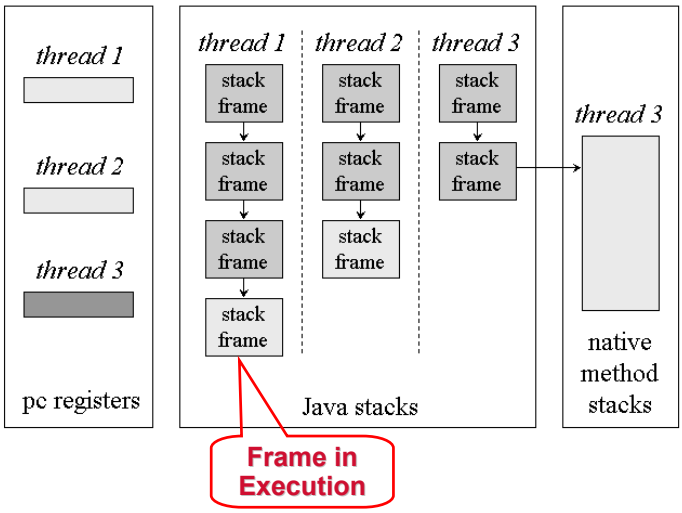
Shared Data Areas



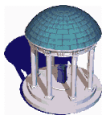
10



Thread Data Areas

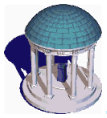


11



Stack Frames

- Stack frames have three parts:
  - Local variables
  - Operand stack
  - Frame data



Stack Frame
Local Variables

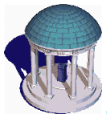
```
class Example3a {

    public static int
runClassMethod(int i, long
l, float f, double d, Object
o, byte b) {

        return 0;

    }

    public int
runInstanceMethod(char c,
double d, short s, boolean
b) {

        return 0;

    }

}
```

runClassMethod()

index	type	parameter
0	int	int i
1	long	long l
3	float	float f
4	double	double d
6	reference	Object o
7	int	byte b

runInstanceMethod()

index	type	parameter
0	reference	hidden this
1	int	char c
2	double	double d
4	int	short s
5	int	boolean b

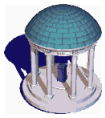


Stack Frame
Operand Stack

Adding 2 numbers

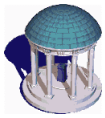
```
iload_0
iload_1
Iadd
istore_2
```

	before starting	after iload_0	after iload_1	after iadd	after istore_2
local variables 0	100	100	100	100	100
local variables 1	98	98	98	98	98
local variables 2					198
operand stack		100	100	198	
operand stack			98		



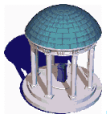
Execution Model

- Eternal Math Example
  - http://www.artima.com/insidejvm/applets/EternalMath.html



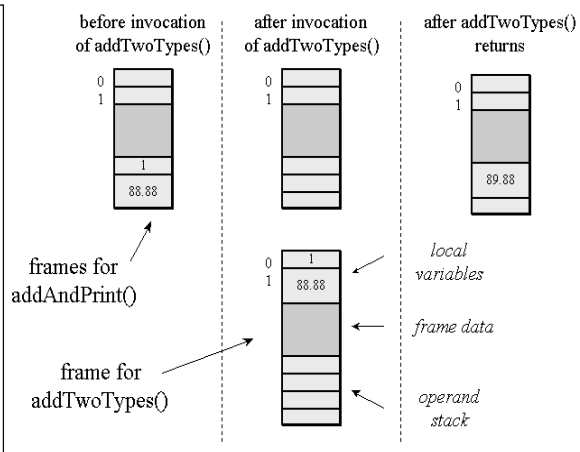
Stack Frame
Frame Data

- The stack frame also supports
  - Constant pool resolution
  - Normal method return
  - Exception dispatch



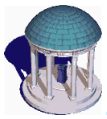
Stack Frame
Frame Allocation in a Heap

```
class Example3c {

    public static void
addAndPrint() {

        double result =
addTwoTypes(1, 88.88);

System.out.println(result)
;

    }


    public static double
addTwoTypes(int i, double
d) {

        return i + d;

    }
}
```



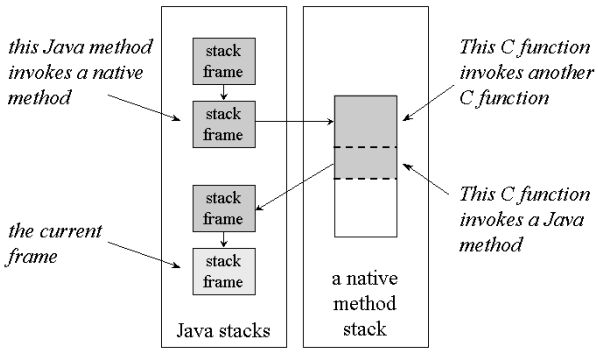
Felix Hernandez-Campos

17

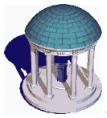


Stack Frame
Native Method

- A simulated stack of the target language (e.g. C) is created for JNI

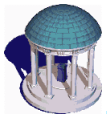


18

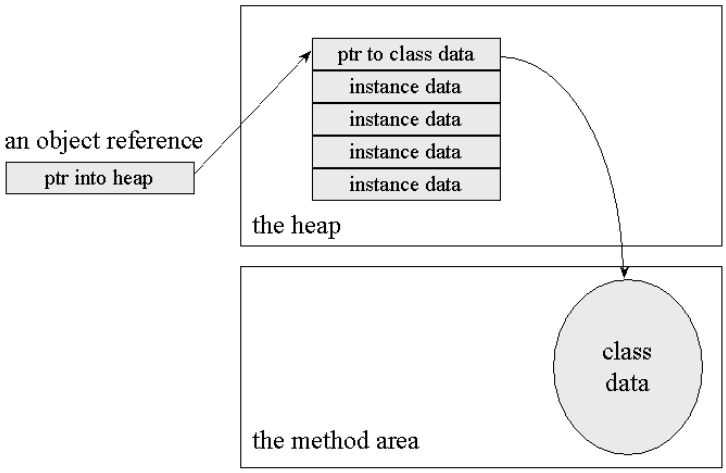


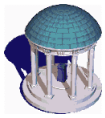
The Heap

- Class instances and array are stores in a single, shared heap
- Each Java application has its own heap
  - Isolation
  - But a JVM crash will break this isolation
- JVM heaps always implement garbage collection mechanisms

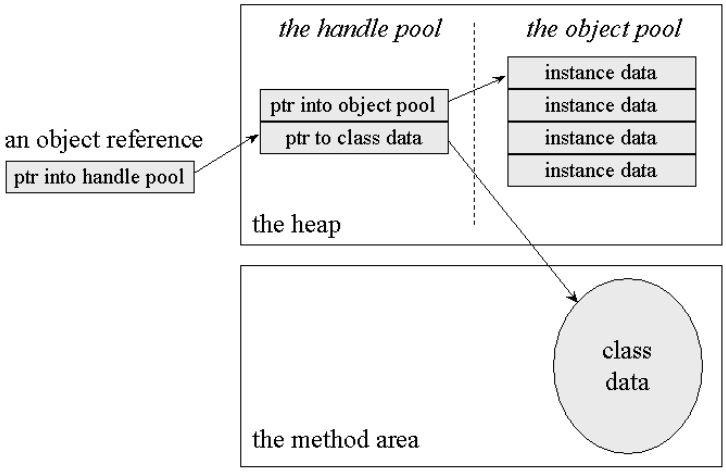


Heap
Monolithic Object Representation

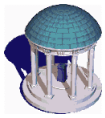




The Heap
Splitted Object Representation

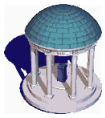


21

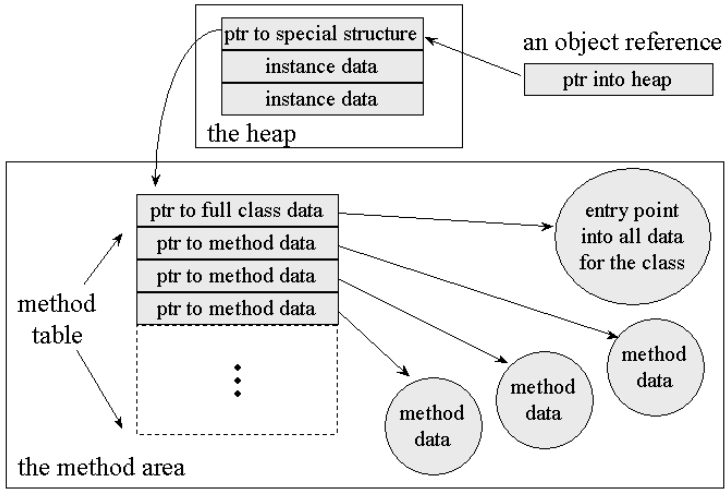


Example

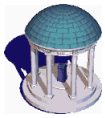
- HeapOfFish
  - http://www.artima.com/insidejvm/applets/HeapOfFish.html



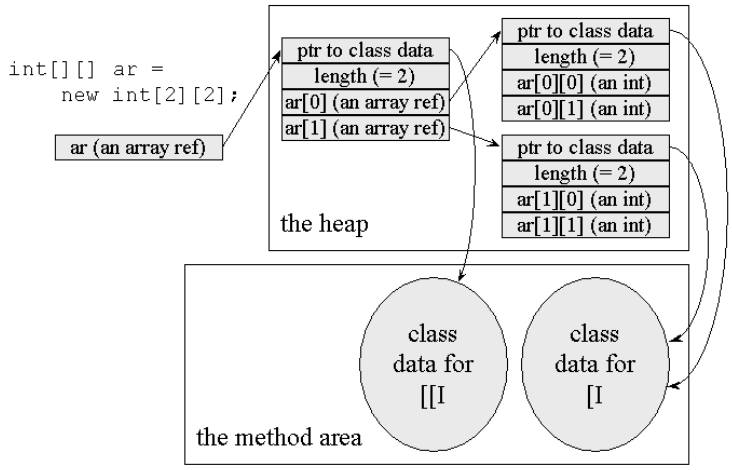
The Heap
Memory/Speed Tradeoff



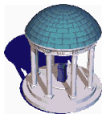
23



The Heap
Arrays as Objects



24



Reading Assignment

- *Inside the Java 2 Virtual Machine* by Bill Venners
  - Ch 1
  - Ch 5
  - Illustrations