The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts

Spring 2003

The Java Virtual Machine

Stotts, Hernandez-Campos

1

The Java Virtual Machine

"Java Architecture"

- Java Programming Language
- Java Virtual Machine (JVM)
- Java API

2

Reference

The content of this lecture is based on *Inside the Java 2 Virtual Machine* by Bill Venners

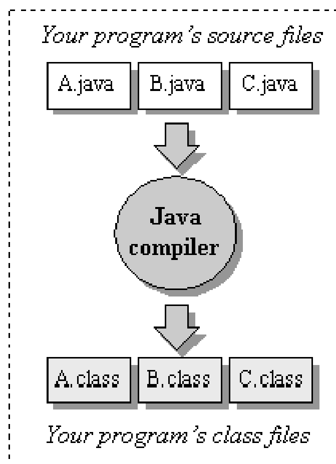
- Chapter 1Introduction to Java's Architecture
  - » http://www.artima.com/insidejvm/ed2/introarchP.html
- Chapter 5 The Java Virtual Machine
  - » http://www.artima.com/insidejvm/ed2/jvmP.html
- Interactive Illustrations
  - » http://www.artima.com/insidejvm/applets/index.html

3

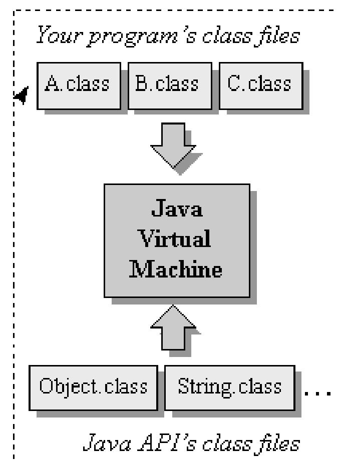
The Java Programming Environment

compile-time environment

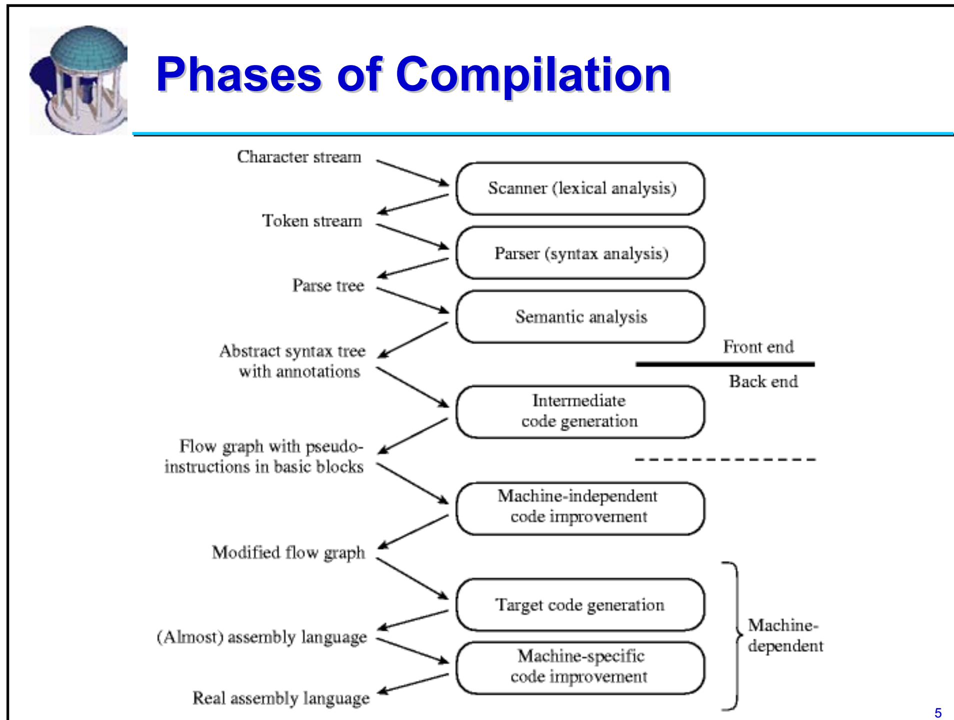
run-time environment

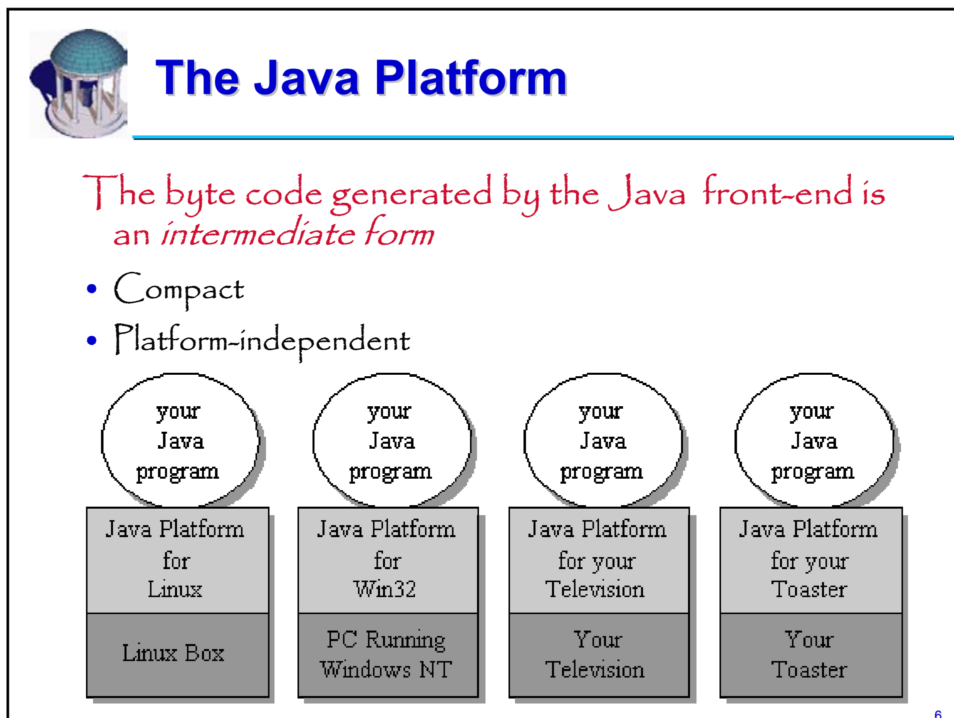


Your class files move locally or though a network



4


Phases of Compilation
Character stream
Scanner (lexical analysis)
Token stream
Parser (syntax analysis)
Parse tree
Semantic analysis
Abstract syntax tree
with annotations
Front end
Back end
Intermediate
code generation
Flow graph with pseudo-
instructions in basic blocks
Machine-independent
code improvement
Modified flow graph
Target code generation
(Almost) assembly language
Machine-specific
code improvement
Real assembly language
Machine-
dependent
5


The Java Platform
The byte code generated by the Java front-end is an intermediate form
Compact
Platform-independent
your Java program
Java Platform for Linux
Linux Box
Java Platform for Win32
PC Running Windows NT
Java Platform for your Television
Your Television
Java Platform for your Toaster
Your Toaster
6

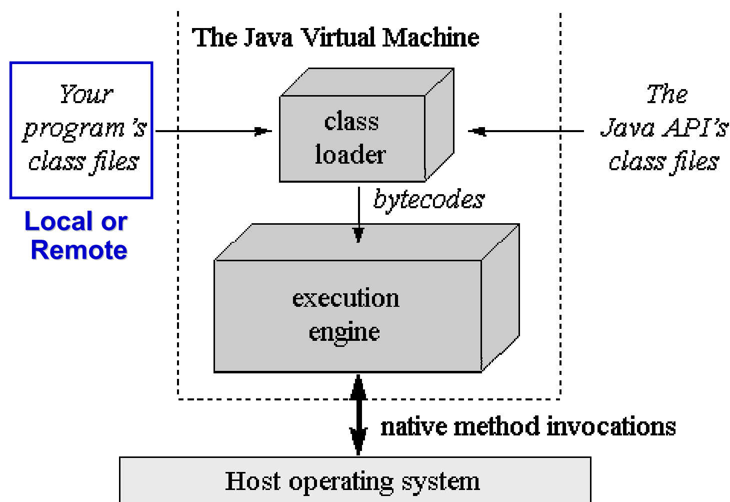
The Class File

Java class file contains

- Byte code for data and methods (intermediate form, platform independent) (*remember byte code?*)
- *Symbolic* references from one class file to another
  - Class names in text strings
  - Decompiling/reverse engineering quite easy
- Field names and descriptors (type info)
- Method names and descriptors (num args, arg types)
- Symbolic refs to other class methods/fields, own methods/fields

7

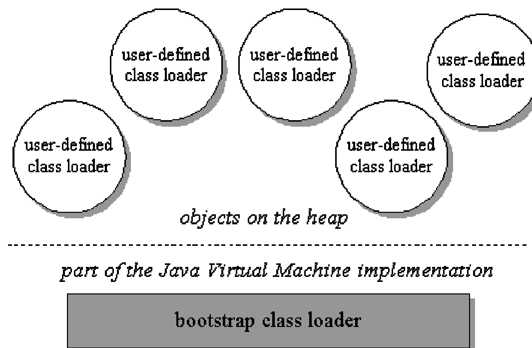
The Role of the Virtual Machime



8

Class Loaders

- Bootstrap (default) loader (in the JVM)
- User-defined (custom) loaders



9

Dynamic Class Loading

- You don't have to know at compile-time all the classes that may ultimately take part in a running Java application.

  User-defined class loaders enable you to dynamically extend a Java app at run-time

- As it runs, your app can determine what extra classes it needs and load them
- Custom loaders can download classes across a network (applets), get them out of some kind of database, or even calculate them on the fly.

10

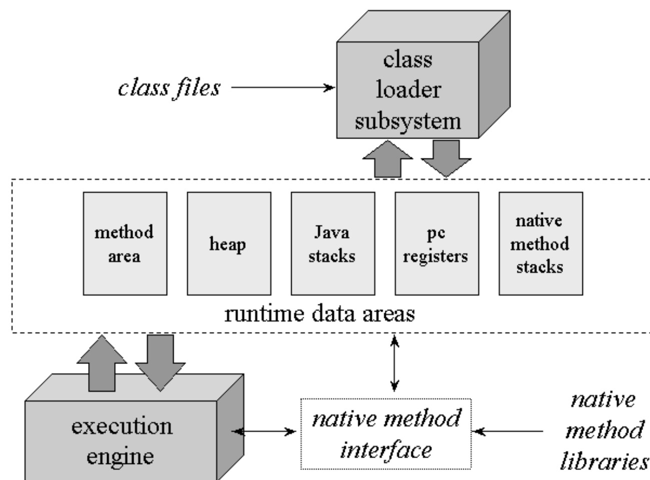
The Execution Engine

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, a little more memory
  - 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)*

11

The Java Virtual Machine



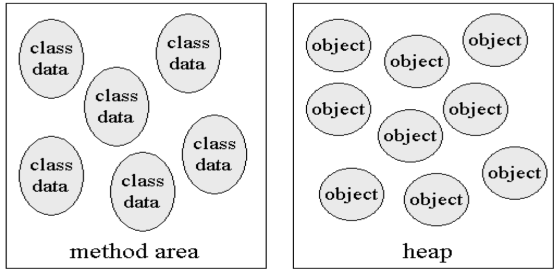
12

Shared Data Areas

Each JVM has one of each:

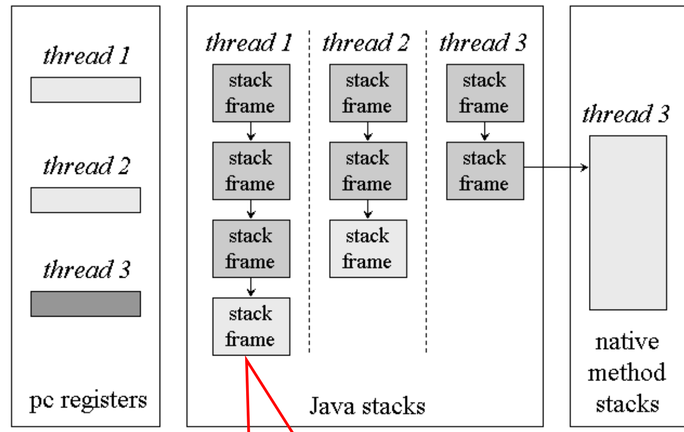
✓Method area: byte code and class (static) data storage

✓Heap: object storage



13

Thread Data Areas



14

Stack Frames

Stack frames have three parts

- Local variables
- Operand stack
- Frame data

15

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

16

Stack Frame
Operand Stack

Adding 2 numbers

```
iload_0
iload_1
Iadd
istore_2
```

Compiler can tell how many slots the op stack will need for a method

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

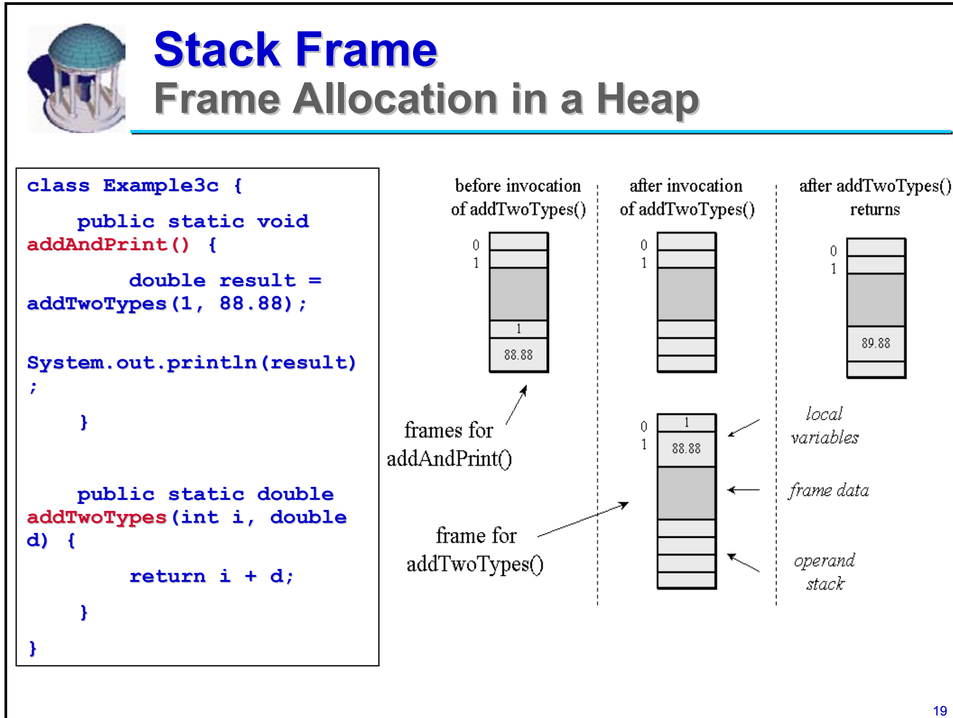
17

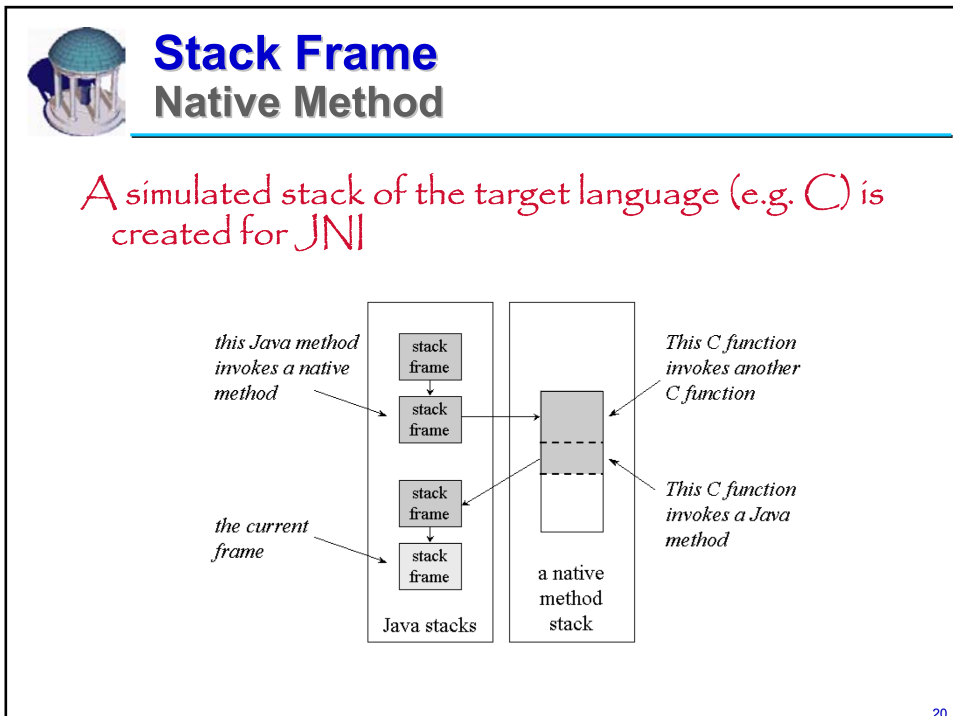
Stack Frame
Frame Data

The stack frame also supports

- Constant pool resolution
- Normal method return
- Exception dispatch

18


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;
}
}
before invocation of addTwoTypes()
after invocation of addTwoTypes()
after addTwoTypes() returns
0
1
88.88
89.88
frames for addAndPrint()
frame for addTwoTypes()
local variables
frame data
operand stack
19

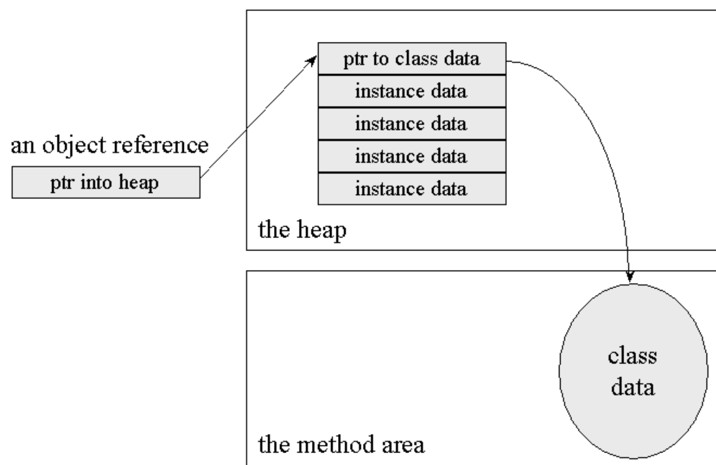

Stack Frame
Native Method
A simulated stack of the target language (e.g. C) is created for JNI
this Java method invokes a native method
the current frame
stack frame
Java stacks
a native method stack
This C function invokes another C function
This C function invokes a Java method
20

The Heap

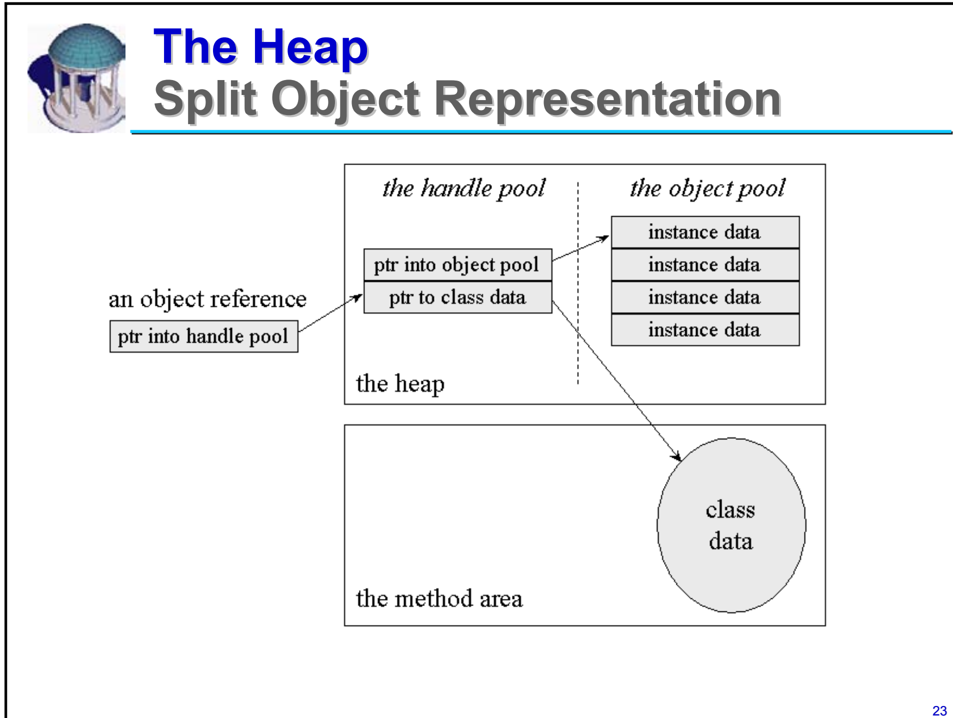
- Class instances (objects) and arrays are stored in a single, shared heap
- Each Java application has its own heap
  - Isolation
  - But a JVM crash will break this isolation
- JVM heaps always implement garbage collection mechanisms

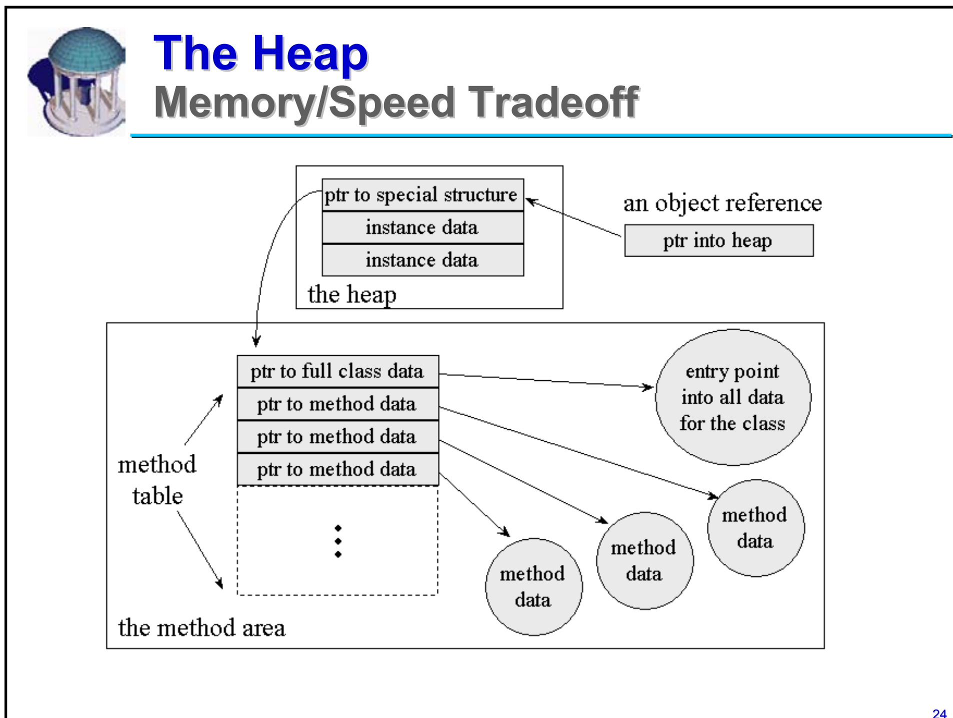
21

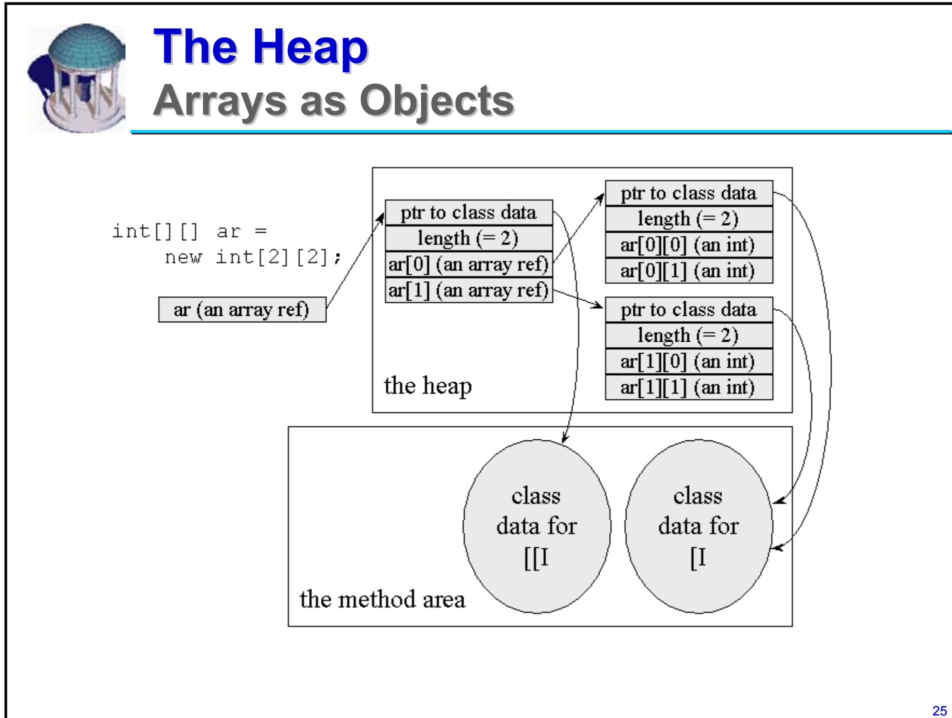
Heap
Monolithic Object Representation



22


The Heap
Split Object Representation
the handle pool
the object pool
instance data
instance data
instance data
instance data
ptr into object pool
ptr to class data
an object reference
ptr into handle pool
the heap
class
data
the method area
23


The Heap
Memory/Speed Tradeoff
ptr to special structure
instance data
instance data
the heap
an object reference
ptr into heap
ptr to full class data
ptr to method data
ptr to method data
ptr to method data
entry point
into all data
for the class
method
table
method
data
method
data
method
data
the method area
24



Examples

HeapOfFish

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

Eternal Math Example

- http://www.artima.com/insidejvm/applets/EternalMath.html
- JVM execution, operand stack, illustration

26

Reading Assignment

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

27