**Java Object-Oriented Concepts**

**Eric V. Level**

The following is a brief guide to core object-oriented concepts in Java.

**Java Variables**

A Java variable is a chunk of memory of some size (usually given in bytes), with a name **<name>** and which can hold a value of some type **<type>**.

Declare a Java variable like this in Java source (textual declaration of a variable): compile-time representation.

**<type> <name>;**

Draw it like this (run-time visual representation of a variable):  
  
 <**name> : <type>**

*<value in memory as as bit pattern>*

**Java Run-time Memory Organization**

There are 3 basic memory areas managed by the JVM at run-time:

Class memory (static area)

Object memory (heap area)

Local memory (stack area)

Class memory – storage for loaded **.class** files (where **static** methods & fields 'live');  
  
Object memory – storage for objects created using **.class** files (where objects and their instance variables and methods 'live' => no **static** memory modifier);

Local memory – storage for local variables (those declared in method bodies);

Local memory holds local variables, created by variable declarations within method code, and used by subsequent code also within method code:

**int temp; … temp = 47;**

Object memory holds created objects, created by **new** operations in method code, and accessed via returned object reference ('safe pointer'):  
  
**MyClass objRef = new MyClass();…**

Class memory – holds loaded **.class** file information, defined by class declarations:

**public class Employee {…}**

**Object Memory Organization**

method 1 header method 2 header method 3 header  
  
 method code method code method code

**age : int**

**name : String**

**income : double**

**age** memory

**name** memory  
(**String** reference)

**income** memory

Instance Methods

Instance Fields

An object has two basic components:

- Instance methods (code headers and associated executable code; the latter may free access/use the object's fields).

- Instance fields (**==** instance variables **==** typed variables within the object's memory);

**Java Code and Runtime Memory Correspondence Example**

Class memory (static area)   
   
Loaded **Employee.class** file   
(defines instance structure)

Object memory (heap area)

Local memory   
(stack area)  
  
  
 **newEmp : Employee**

**public class Employee {**

**main(…)**

**// define instance variables in Employee object  
  
 private int age;**  **private double income;  
 private String name;**

**// declare Employee constructor (initializer)**

**public Employee (int a, double i, String n) {  
  
 age = a;   
 income = i;   
 this.name = n; // this. not needed, but OK**

**B o b**

**"Bob" ref**

**47**

**}**

**50000.**

**// declare instance method**

**public void setAge (int newAge) {  
  
 age = newAge; // change age field within**

**}**

**<reference to Employee   
 object in heap>**

**// declare static method**

**public static void main (String[] args) {  
  
 Employee newEmp   
 = new Employee (47, 50000.0, "Bob");  
  
 newEmp.setAge (48); // Bob is a year older  
 }  
}**

**Memory looks like above after main method executes…**