COM S 211 Computers and Programming (also ENGRD 211)

Fall, spring, summer. 3 credits. Prerequisite: COM S 100 or an equivalent course in Java or C++.

Intermediate programming in a high-level language and introduction to computer science. Topics include program structure and organization, modules (classes), program development, proofs of program correctness, recursion, data structures and types (lists, stacks, queues, trees), object-oriented and functional programming, analysis of algorithms, and an introduction to elementary graph theory and graph algorithms. Java is the principal programming language. Knowledge of classes and objects is assumed.

Topics

- Program Organization and Development
- Object oriented programming and Java basics
- Data Structures: lists, queues, stacks, trees, etc.
- Algorithms: searching and sorting
- Introduction to program analysis
- Introduction to graphs and graph theory
- Recursion and Induction

You should already know…

- Basic Programming
  - Variables, arithmetic, order of operations, etc.
  - Simple control flow (if, while, and for loops)
  - Methods/Functions
- Basic Java
  - Basic types (`int`, `double`, `String`…)
  - Variables
  - Arithmetic and simple casting
  - Objects, Classes, Instances, Constructors
  - `this`, `super`, `public`, `private`, `protected`…
  - Static and instance variables and methods
  - Simple input/output using console and files

Boot Camp

- Wednesday, 1 – 2 PM
- Location TBA
- Covers Java basics (CS100J essentials)

Administrative Stuff

http://www.cs.cornell.edu/courses/cs211/2004su/

Course Staff

- Kevin Walsh – Instructor
- Benjamin Mathew – TA
- Eli Rosofsky – TA
- Meghan Desai – Consultant
- Kevin Canini – Consultant

- Consulting & office hours regularly – see web site
  - KW: directly after class today, Upson 5138

Programmming

- Environments
  - DrJava
  - CodeWarrior
  - MS Visual Studio VJ++
  - Forte
  - Sun's J2SE
  - JBuilder, JDeveloper, GNU…
- Java Version 1.4.1 or higher
- Available in public CIT labs

Course Work

- ~ 6 assignments : programming or pen & paper
- 2 prelims in class
- 1 final exam or project (our choice)
- All work turned in via CMS, on web page
- No late work accepted

Expectations

1. Work hard
   - Attend class every day
   - Do assignments on time (no late work accepted)
   - Do assigned readings and stay current with lectures
   - Get help when you need it
2. Be honest
   - Cheating on exams or assignments will not be tolerated
   - I expect you all to abide by academic integrity standards
   - Know what plagiarism is, especially as regards "programming"

Expectations

1. Work hard
   - Attend class every day
   - Grade the projects and assignments in a timely manner
   - Push you hard
   - Help when you need it
2. Be honest
   - Expect you all to abide by academic integrity standards
   - Recognize plagiarism and other forms of cheating

Primitive Java

Weiss: ch 1

Java : Introduction

```
class Top{
  public static void main(String[ ] args) {
    Work.squares(1, 10);
    System.out.println("Made " + Work.powCalls
                + " calls");
  }
}
```

```
class Work {
  public static int powCalls = 0;

  public static void squares(int lo, int hi){
    for (int i = lo; i < hi; i++)
      System.out.println(pow(i,2));
  }

  public static int pow(int b, int p){ //p>0
    powCalls = powCalls + 1;
    int value = 1;
    for (int i = 0; i < p; i++)
      value = value*b;
    return value;
  }
}
```

class variable:

int

class method:

int x int => void

class method:

int x int => int

(parameters are call-by-value. e.g., copied)

Java : Introduction

```
class Top{
  public static void main(String[ ] args) {
    Work.squares(1, 10);
    System.out.println("Made " + Work.powCalls
                + " calls");
  }
}
```

```
class Work {
  public static int powCalls = 0;

  public static void squares(int lo, int hi){
    for (int i = lo; i < hi; i++)
      System.out.println(pow(i,2));
  }

  public static int pow(int b, int p){ //p>0
    powCalls = powCalls + 1;
    int value = 1;
    for (int i = 0; i < p; i++)
      value = value*b;
    return value;
  }
}
```

```
Welcome to DrJava.
> Top.main(null)
1
4
9
16
25
36
49
64
81
Made 9 calls
>
```

- Program is a collection of *classes*
- Classes contain *members*
  - Variables & methods
  - Should be logically related
- Example:

```
package java.lang;
public final class Math {
  public static final double E = 2.7182818284590451D;
  public static final double PI = 3.1415926535897931D;

  public static int abs(int i) { return i >= 0 ? i : -i; }
  public static int max(int i, int j) { return i < j ? j : i; }
  public static long round(double d) { return (long)floor(d + 0.5D); }
  public static double sin(double d) { … }
  …
}
```

Java : Member Naming

- When referring to a member of a class…
  - Use complete path name : *class.member*
    - Work.powCalls
  - Use relative path name : *member*
    - powCalls
    - But only from within same class
  - (And sometimes need to use package name as well…)
- Member visibility
  - *public:* visible to methods in other classes
  - *private:* visible to methods in same class

Java: Static Binding

- What's in a name?
  - E.g. System.out.println(pow(i,2));
  - Which class has *pow* as a member?
  - Names can mean different things at different times or in different contexts
    - This is what makes OO languages powerful (and confusing)
- Static binding
  - Binding: connecting a name to a member in some class
  - Static: At "compile time", looking only at program text
  - Can check visibility as well
- Dynamic binding
  - Dynamic: At "run time", looking at state of computer
  - Can be hard (or impossible) to determine some bindings without actually running the program

Java: Overloading

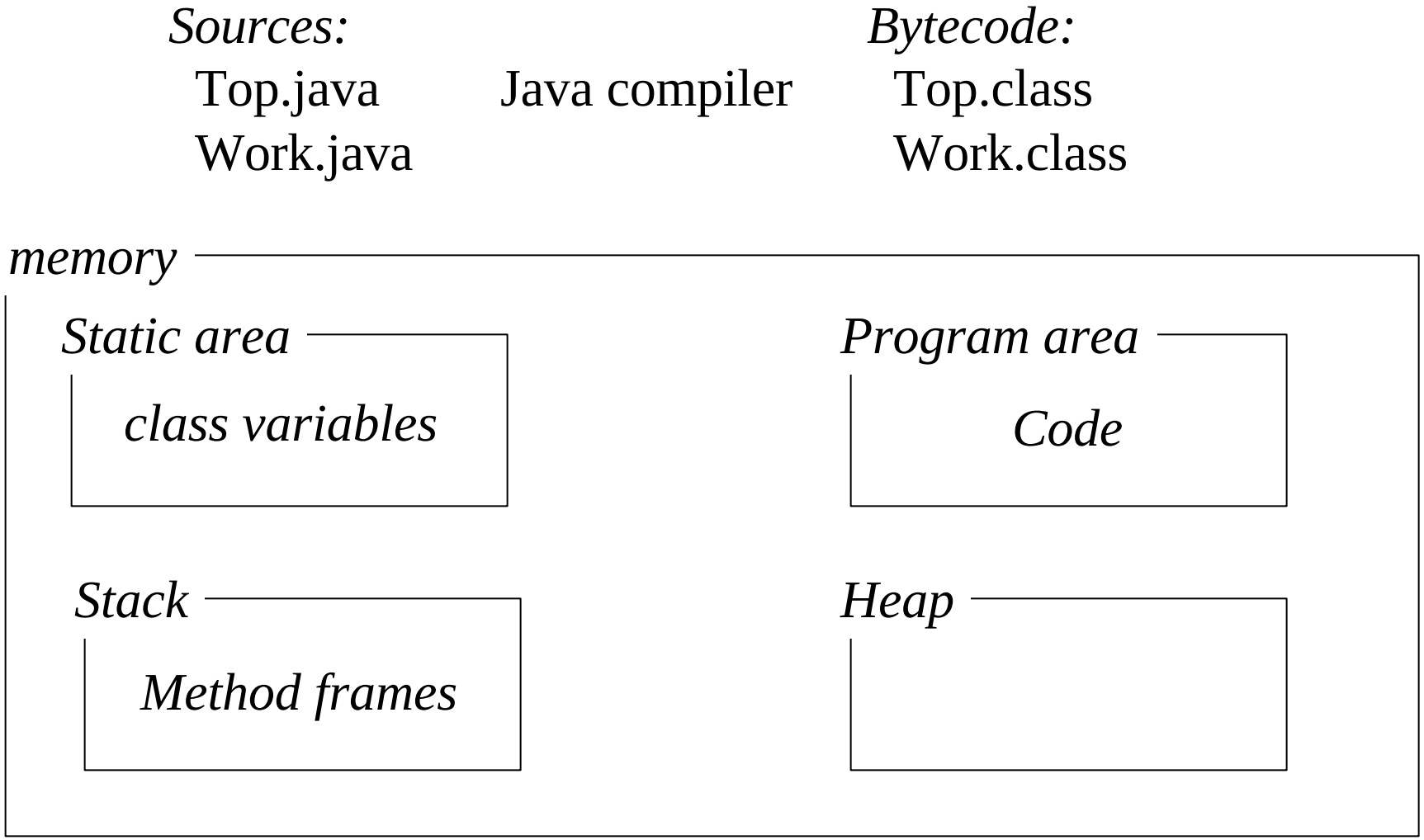
- Can members have the same name?
  - Goal: avoid ambiguity during binding
- Member is unrelated classes: yes
  - Distinguish between them by the class name
- Variables in same class: no
  - No (obvious) way to distinguish
- Methods in same class: only if argument lists differ
  - Different number of arguments
  - Different type of arguments

- Example

```
package java.lang;
public final class Math {
 …
 public static int max(int i, int j) { return i < j ? j : i; }
 public static long max(long l, long l1) { return l < l1 ? l1 : l; }
 public static double max(double d, double d1) {
  if(d != d)
    return d;
  if(d == 0.0D && d1 == 0.0D
    && Double.doubleToLongBits(d) == negativeZeroDoubleBits)
    return d1;
  else
    return d < d1 ? d1 : d;
 }
 …
```

- Compiler chooses which *max* to use
  - Goal: don't loose information
  - *int i; …; max(i, 3.0)* will use *double x double => double*

- Dynamic behavior: what happens when we run the program?



```
class Work {
  public static int powCalls = 0;

  public static void squares(int lo, int hi){
    for (int i = lo; i < hi; i++)
      System.out.println(pow(i,2));
  }

  public static int pow(int b, int p){ //p>0
    powCalls = powCalls + 1;
    int value = 1;
    for (int i = 0; i < p; i++)
      value = value*b;
    return value;
  }
}
```

```
class Top{
  public static void main(String[ ] args) {
    Work.squares(1, 10);
    System.out.println("Made “
                + Work.powCalls + " calls");
  }
}
```

Static area

Stack

```
class Work {
  public static int powCalls = 0;

  public static void squares(int lo, int hi){
    for (int i = lo; i < hi; i++)
      System.out.println(pow(i,2));
  }

  public static int pow(int b, int p){ //p>0
    powCalls = powCalls + 1;
    int value = 1;
    for (int i = 0; i < p; i++)
      value = value*b;
    return value;
  }
}
```

```
class Top{
  public static void main(String[ ] args) {
    Work.squares(1, 10);
    System.out.println("Made “
                + Work.powCalls + " calls");
  }
}
```

Static area

Stack

```
class Work {
  public static int powCalls = 0;

  public static void squares(int lo, int hi){
    for (int i = lo; i < hi; i++)
      System.out.println(pow(i,2));
  }

  public static int pow(int b, int p){ //p>0
    powCalls = powCalls + 1;
    int value = 1;
    for (int i = 0; i < p; i++)
      value = value*b;
    return value;
  }
}
```

```
class Top{
  public static void main(String[ ] args) {
    Work.squares(1, 10);
    System.out.println("Made “
                + Work.powCalls + " calls");
  }
}
```

Static area

Stack

- Why have class variables / static area?
  - Data needed by many classes
    - Math.E, Math.PI, etc.
  - Data that survives method invocations
    - Work.powCalls
- Why have method local variables / stack ?

First Assignment

- Posted today, 6/28
- Due Thursday, 7/1 @ midnight
- Covers:
  - Access to course materials, CMS, newsgroups, etc.
  - Academic integrity
  - Poll: Consulting hours and labs
  - Basic Java (CS 100 level)