Course Details

Object Oriented Specification and Implementation

- Times: Mondays: 4:30 - 7:10 PM
- Place: Robinson B104
- Instructor: Jonathan Doughty
- E-mail: jdoughty@cs.gmu.edu
- TA: Anil Vaitla
- E-mail: avaitla@gmu.edu
What to expect

- the essentials of object oriented programming,
- the fundamentals of object oriented design,
- how to use Java to create object oriented applications
What not to expect

- How to write JavaScript, create applets on web pages, create graphical user interfaces, ...
- A complete / thorough Java course - only the object oriented aspects
- A class solely devoted to design
- These class notes as a substitute for class attendance
Grading Policy

Your grade in this class will be based on:

- Homework and programming assignments - 25%
- A mid-term exam - 20%
- A final project - 25%
- Final exam - 25%
- Class participation - 5%

Assignments are due by the start of the next lecture no extensions.
Honor Policy

You are expected to abide by both the

- George Mason University Honor Code as well as the
- Computer Science Department Honor Code

in completing the requirements of this class.
Expectations for this Course

- Assignments are expected to be done by the individual. Each individual is expected to complete and hand in their own version of programming assignments.

- Full credit if correct and turned in on time. Only partial credit otherwise.

  Where correct == addresses the complete homework specification in the spirit of Object Oriented programming.

  Where on time == by the start of the next lecture.

- Programming assignments are expected to show individual "value-added".

  If you’ve used material you’ve found to help complete the assignment (including material from the text or examples I’ve provided) I expect you to have added to it in a significant way.

- Plagiarism (e.g., copying Java code from books, CDs, or web sources) without proper and complete attribution and adherence to copyrights, etc. is not allowed.
Homework and Final Project

Programming assignment guidelines: What I am looking for:

- Demonstrate that you understand and can apply Object Oriented concepts
- That the program’s components are understandable.

  Good OO programs are nearly self documenting. That means using coding conventions counts.

- That the program compiles
- That the program runs correctly
Java coding conventions

Can be found in

- Sun’s Code Conventions for the Java Programming Language
- Java Language Specification
- Doug Lea’s Draft Java Coding Standard

The main rule: be consistent
**Required Reading**

**Textbook**

*Beginning Java Objects: From Concepts to Code*

Jacquie Barker


Material covered in this class:

- The ABCs of Objects
- Object Modeling
- Transforming Your Model into Java Code

**Supplementary Reading**

Articles from the web; some will be required reading; others are purely for additional information.
Recommended Text
(as a Java reference)

Thinking in Java

Bruce Eckel


- I will not be assigning any reading from TIJ

- The book can be downloaded in electronic form from the mirror sites listed at http://www.mindview.net/Books/DownloadSites
Course website

http://cs.gmu.edu/~jdoughty/cs332/

- Lecture slides
- Assignments
- Updates
What you need for this course

- A Java Development Kit (JDK) - free
- A text editor - any will do
  - e.g., Notepad, vi, pico
  - I do not recommend an Integrated Development Environment
  - On Windows, I frequently recommend http://www.jcreator.com/index.htm (the LE version)
- A web browser - for reading Javadoc Application Programming Interface (API) documentation, class materials
Where you may work

You may work on any machine that supports at least Java version 1.2 or higher:

- On OSF1 / Mason cluster, add
  
  ```
  set path = ( /usr/opt/java130/bin $path )
  ```

  to your ~/.login file

- On Windows, you’ll need to download a copy of a Java Development Kit (JDK). These are free, but large downloads/installations: roughly 30 Megabytes to download, another 43 megabytes to install. I recommend you get the current standard edition: JDK 1.3.1

- On Macintosh, ask me.

- On Linux, likewise you’ll need to download a copy of a Java Development Kit (JDK). I recommend you get either the current JDK 1.3.1 release from Sun or the Blackdown group. If you have questions, ask me.
Questions?
Before we get started
Whirlwind Introduction to Java

C++ / Java

HelloClass.cc

/* One of the simplest complete C++ programs you can write */

#include <iostream>

int main() {
    cout << "Hello class, this is C++!" << endl;
    return 0;
}

HelloClass.java

/* One of the simplest complete Java programs you can write */

public class HelloClass {

    // This is a "method"; notice its signature:
    // return type, name, number and type of arguments

    public static void main(String[] args) {
        System.out.println("Hello class, this is Java!");
    }
}
Neither of the above is **object oriented**
## Fundamentals of Java

### Reserved Words

<table>
<thead>
<tr>
<th>Primitives</th>
<th>Control Flow</th>
<th>Qualifier</th>
<th>Other</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>break</td>
<td>abstract</td>
<td>false</td>
<td>byvalue</td>
</tr>
<tr>
<td>byte</td>
<td>case</td>
<td>class</td>
<td></td>
<td>cast</td>
</tr>
<tr>
<td>char</td>
<td>catch</td>
<td>extends</td>
<td></td>
<td>const</td>
</tr>
<tr>
<td>double</td>
<td>continue</td>
<td>final</td>
<td></td>
<td>future</td>
</tr>
<tr>
<td>float</td>
<td>default</td>
<td>import</td>
<td></td>
<td>generic</td>
</tr>
<tr>
<td>int</td>
<td>do</td>
<td>implements</td>
<td></td>
<td>goto</td>
</tr>
<tr>
<td>long</td>
<td>else</td>
<td>interface</td>
<td></td>
<td>inner</td>
</tr>
<tr>
<td>short</td>
<td>finally</td>
<td>native</td>
<td></td>
<td>operator</td>
</tr>
<tr>
<td>for</td>
<td></td>
<td>package</td>
<td></td>
<td>outer</td>
</tr>
<tr>
<td>if</td>
<td></td>
<td>private</td>
<td></td>
<td>rest</td>
</tr>
<tr>
<td>return</td>
<td></td>
<td>protected</td>
<td></td>
<td>var</td>
</tr>
<tr>
<td>switch</td>
<td></td>
<td>public</td>
<td></td>
<td></td>
</tr>
<tr>
<td>throw</td>
<td></td>
<td>static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>try</td>
<td></td>
<td>synchronized</td>
<td></td>
<td></td>
</tr>
<tr>
<td>while</td>
<td></td>
<td>throws</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>transient</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>void</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>volatile</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Java Syntax

- Control statements (if, else, for, while, do while, switch) - identical to C’s

- Block definitions ( { ... } ) - identical to C’s (scoping rules are enforced, however)

- Relation operators ( >, >=, <, <=, ==, != ) - identical to C’s

- Shortcut assignment, Pre, Post increment / decrement operators ( +=, -=, *=, ..., ++, --) - identical to C’s

- Conditional operators ( &&, ||, ! ) - identical to C’s

- Bit / Boolean manipulation operators ( &, |, ^, ~, <<, >>, >>>) (nearly) identical to C’s
Some Differences between Java and C/C++

- No pointers, at least as far as you are concerned
- No preprocessor, no macros
- No global variables or functions
- No programmer defined operator overloading, (almost) no operator overloading at all
- Arrays are handled somewhat differently. Note that in this class I use object oriented Collection classes instead almost exclusively.
- No multiple inheritance

More information (not required reading)

- Textbook Appendix E
- The Java Tutorial "trails" on Getting started and Learning the Java Language
- Java for C++ programmers - very short summary of differences
- Java for C++ Developers
- Java vs. C++: A Critical Comparison
A Toy Problem Specification

Model enrollment in a CS class:

- Permit student names to be given on the command line
- Assign lab partners as student pairs. An odd student will have no partner.
- List the students in the class (and their lab partner, if any) in reverse order of enrollment.
A Procedural Solution

CSCourseP.java

/** A Java class that models information kept about a CSCourse in
 ** a procedural way.
 **
 ** @author Jonathan Doughty
 ** @version 1.0
 ***/
public class CSCourseP {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("usage java CSCourseP [student names]");
        } else {
            enroll( args );
            assignLabPartners();
            listRoster();
        }
    }

    // fields
    static String studentNames[] = null;
    static int studentIds[] = null;
    static int labPartners[] = null;
    static int last = 0;

    public static void enroll(String[] names) {
        int numberOfStudents = names.length;
        int nextId = 0;

        // Create all arrays
        studentNames = new String[numberOfStudents];
        studentIds = new int[numberOfStudents];
        labPartners = new int[numberOfStudents];

        for (int arg = 0; arg < numberOfStudents; arg++) {
            String name = names[arg];
            studentNames[last] = name;

            nextId++;
            // assign next id
        }
    }

    } // class

} // package

public class CSCourseP {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("usage java CSCourseP [student names]");
        } else {
            enroll( args );
            assignLabPartners();
            listRoster();
        }
    }

    // fields
    static String studentNames[] = null;
    static int studentIds[] = null;
    static int labPartners[] = null;
    static int last = 0;

    public static void enroll(String[] names) {
        int numberOfStudents = names.length;
        int nextId = 0;

        // Create all arrays
        studentNames = new String[numberOfStudents];
        studentIds = new int[numberOfStudents];
        labPartners = new int[numberOfStudents];

        for (int arg = 0; arg < numberOfStudents; arg++) {
            String name = names[arg];
            studentNames[last] = name;

            nextId++;
            // assign next id
        }
    }

    } // class

} // package
studentIds[last] = nextId;

last++;
}
}

public static void assignLabPartners() {

    // Assign every other pair of students as lab partners
    int next = 0;
    int pairs = last / 2;
    while (pairs > 0) {
        labPartners[next] = next+1;
        labPartners[next+1] = next;
        next += 2;
        pairs--;
    }

    // If there were an odd number of students in the array the
    // last one won’t have a lab partner. I need to mark that
    // fact so the odd student’s information can be treated
    // specially when printed.
    if ( (last % 2) == 1)
        labPartners[next] = -1;
}

public static void listRoster() {
    for (int i = last - 1; i >= 0; i--) {
        // Print out information about each student and her lab partner
        System.out.print("Name: " + studentNames[i] +
        " Id: " + studentIds[i]);
        if (labPartners[i] >= 0)
            System.out.println(" Lab partner: " + studentNames[labPartners[i]]);
        else
            System.out.println();
    }
}

What makes it "Procedural"?
An Object Oriented Solution

CSCourse.java

/** A set of Java classes that model information kept about a
** course and students in the course in an object oriented way.
**
** @author Jonathan Doughty
** @version 1.0
**/  
public class CSCourse {

/** The starting point for this application
**/  
public static void main(String[] args) {
    if (args.length == 0) {
        System.out.println("usage java CSCourse [student names]");  
    }  
    else {  
        // Create a new course object
        CSCourse cs332 = new CSCourse();  
        // Ask the course to enroll students, assign lab partners,
        // and list the class roster.
        cs332.enroll( args );  
        cs332.assignLabPartners();  
        cs332.listRoster(System.out);  
    }  
}

// Fields associated with an individual CSCourse object

/** Dynamic collection of references to student objects enrolled
** in the class
**/  
java.util.Vector students = new java.util.Vector();

// Methods that can be called on individual CSCourse objects,
// i.e., messages that can be sent to individual CSCourse
// objects.

/** Enroll students whose names are provided in the input array
** into this course.
**/  
public void enroll(String[] names) {

int numberOfStudents = names.length;
int id = 0;

for (int next = 0; next < numberOfStudents; next++) {
    // Create the student objects themselves
    CSStudent current = new CSStudent(names[next]);
    students.addElement(current);
}

/** Assign pairs of students as lab partners for this course. **/
public void assignLabPartners() {

    // Assign every other pair of students as lab partners
    int next = 0;
    int pairs = students.size() / 2;
    while (pairs > 0) {
        CSStudent student = (CSStudent) students.elementAt(next);
        CSStudent partner = (CSStudent) students.elementAt(next+1);

        student.assignPartner(partner);
        partner.assignPartner(student);
        next += 2;
        pairs--;
    }

    // If there were an odd number of students the last one won’t
    // have a lab partner. I don’t need to do anything special
    // here in the the CSCourse object, a CSStudent itself "knows"
    // when it has a partner or not.
}

/** List the roster of students enrolled in this course to
 ** standard output
 **/
public void listRoster(java.io.PrintStream out) {
    for (int i = students.size() - 1; i >= 0; i--) {
        // Print out information about each student. In this case,
        // simply request each student object reference to identify
        // itself.
        out.println(students.elementAt(i)); // note: no cast to CSStudent type
    }
}
class CSStudent {

    // "static" fields are associated with the class, not individual objects.
    private static int nextId = 0;

    // Fields associated with an individual CSStudent objects.
    String name;
    int id;
    CSStudent labPartner = null;

    // Constructor for CSStudent objects
    public CSStudent(String newStudentName) {
        name = newStudentName;
        id = nextId++;
    }

    // Methods that can be called on individual CSStudent objects,
    // i.e., messages that can be sent to individual CSCourse
    // objects.
    public String getName() {
        return name;
    }

    public void assignPartner(CSStudent partner) {
        labPartner = partner;
    }

    public String toString() {
        String result = "Name: " + name + " Id: " + id;
        if (labPartner != null)
            result += " Lab partner: " + labPartner.getName();
        return result;
    }
}

/** The CSStudent class will enable objects to be created, each of
** which encapsulates information about an individual student. **/

class CSStudent {

    // "static" fields are associated with the class, not individual objects.
    private static int nextId = 0;

    // Fields associated with an individual CSStudent objects.
    String name;
    int id;
    CSStudent labPartner = null;

    // Constructor for CSStudent objects
    public CSStudent(String newStudentName) {
        name = newStudentName;
        id = nextId++;
    }

    // Methods that can be called on individual CSStudent objects,
    // i.e., messages that can be sent to individual CSCourse
    // objects.
    public String getName() {
        return name;
    }

    public void assignPartner(CSStudent partner) {
        labPartner = partner;
    }

    public String toString() {
        String result = "Name: " + name + " Id: " + id;
        if (labPartner != null)
            result += " Lab partner: " + labPartner.getName();
        return result;
    }
}
What makes it "Object Oriented"?
Some Key OO Concepts

Encapsulation:

- A CSCourse object "knows" how to enroll students, assign lab partners, ask students to identify themselves.

- A CSStudent object "knows" how to remember and be able to return her name, assign herself the next available ID, how to remember who her lab partner is, how to identify herself.

Delegation:

- The CSCourse object delegates allocating storage for referencing groups of student objects and being able to return student objects (by index) to the students Vector

- The CSCourse object delegates remembering lab partners and how to identify themselves to CSStudent objects

Composition:

- A CSCourse object is composed of a collection students (an object of type java.util.Vector named students)

- CSStudent objects each are composed of a name, an id, and a lab partner

Responsibility for different aspects of the system has been divided up into different abstract data types.
Assignment for Next Session

Reading - Textbook:

- Chapter 1 - A Little Taste of Java
- Chapter 2 - Abstraction and Modeling

Programming

There are two simple goals of this assignment:

- To insure that you have a working Java development environment
- To communicate to me that you are using the correct version of Java and what sort of system you are working on.

Your problem specification:

- Find yourself a Java2 environment to work in:
  - Unix - OSF1 has JDK 1.3.0 available, though you need to adjust your .login file:
    ```
    set path = ( /usr/opt/java130/bin $path )
    ```
  - Windows - The lab in ST I Room 126 has Java2
  - On your own PC - Java Development Kits are available for free download [http://java.sun.com/j2se/1.3/](http://java.sun.com/j2se/1.3/)
  - For this class you need JDK 1.2.x or better.
  - Other systems you may have access to have other challenges. You may send me email at jdoughty@cs.gmu.edu and I’ll provide suggestions. Don’t wait until the last minute, however.
Check that your Java environment is working correctly by issuing the command line

```
java -version
```

- Using a text editor in your chosen environment:
  - Create a file Me.java
  - Make the source program output:
    - Your name
    - Your GMU ID
    - Your email address
    - The date and time - you decide how to accomplish this; we’ll discuss some possibilities next week.
  - Your Me.java class should include this:
    ```java
    System.out.println(System.getProperty("java.version") + " " + System.getProperty("os.name");
    ```
  - Compile the file to create the file Me.class
    ```
javac Me.java
    ```
  - Fix any reported errors; when the Me.java file compiles cleanly, run the result:
    ```java
    java Me
    ```
  - Hand in next week a listing of the source and the result.

If you run into problems getting either the javac compiler or the java interpreter to work you may find the Java Tutorial lessons Your First Cup of Java helpful, though not required.