CS 330: Syllabus – Fall 2013

Steven Zeil

August 20, 2013

Contents

1 Basic Course Information

1.1 When and Where

This course is online and has no regularly scheduled lectures. There will be a limited number of attendance-optional network conferences (announced on the website).

Website: `http://www.cs.odu.edu/~zeil/courses/cs330web.html`

1.2 Objectives:

This course will explore the techniques of object-oriented programming, analysis, and design. The emphasis will be upon the development of clean interfaces that permit easy modification and reuse of software components. Other techniques, drawn from outside the object-oriented approach, that significantly contribute to this goal will also be discussed.

Students will gain facility in an object-oriented programming language and will learn the constructs that differentiate such languages from others. This course will explore the idioms and styles of object-oriented programming in C++ and Java, with emphasis upon how these contribute to reusable software components.

Students will learn how to use object-oriented techniques in support of programming. In particular, students will be introduced to the process of object-oriented analysis as a means of understanding an unfamiliar problem domain. Students will learn to build and use models, expressed in the Unified Modeling Language (UML) to codify and evaluate that understanding and to evolve system requirements.. Students will learn how to use those models to facilitate a smooth transition from analysis to design and from there to implementation.

1.3 Textbooks:

Required:

- Horstmann, *Object-Oriented Design and Patterns*, 2nd ed., John Wiley & Sons, ISBN 0-471-74487-5
- Any reasonable C++ text that covers classes and inheritance (e.g., your CS250 or 333 textbook)

Optional:

- Fowler, *UML Distilled: A Brief Guide*, 3rd ed., Pearson AW, ISBN 0321193687

2 Communications

Steven J. Zeil	E&CS 3208
(757) 683-4928	Fax: (757) 683-4900
cs330@cs.odu.edu	

Specific course options and policies regarding communications are presented in Navigating The Course Website and Asking Questions.

2.1 Office Hours:

A week-by-week schedule of available meeting times with me can be found at `http://www.cs.odu.edu/~zeil/officehours/`.

Note that I am available during office hours both in person via network conferencing and often has evening hours via network-conferencing only.

3 Course Prerequisites

The prerequisites for this course are:

- CS 250, Problem Solving and Programming II, or CS 333 Problem Solving and Programming in C++
- CS 252 Introduction to Unix for Programmers

or equivalents.

> If you need to review any of the prerequisite topics described below, the time to do so is early in the semester, *before* you need it to understand the lectures or are called to use it in assignments.

Students who have successfully completed those prerequisites should have acquired the following knowledge and skills:

3.1 General Programming Knowledge

Students should be familiar with certain basic programming techniques that are largely independent of any specific programming language:

- using editors, compilers and other basic software development tools.
- basic software design (i.e., stepwise refinement and top-down design)
- software testing
  - the use of scaffolding code (stubs and drivers)
  - selection of test cases for black-box testing
  - head to head testing
- debugging, including the use of debugging output, the use of automatic debuggers to set breakpoints and trace program execution, and the general process of reasoning backwards from failure locations to the faulty code responsible for the failure.

If you are uncertain about your preparation in any of these areas or if you simply wish to review some of them, you may want to visit the CS333 website. (For those of you entering this course from CS250, CS333 is an accelerated web-based course that combines material from both CS150 and 250.)

3.2 C++

I will assume that you are familiar with the basics of C++, including

- the various C++ statements and control-flow constructs,
- the built-in data types,
- the use of arrays, pointers, pointers to arrays, and linked lists,
- the use and writing of functions, and
- the basic use of structs and classes for implementing abstract data types.

Again, if you are uncertain about your preparation in any of these areas, you may want to visit the CS333 website.

3.3 Java

No prior knowledge of Java is expected. In general, CS students at the 300 level should be able to pick up new programming languages with only moderate effort. Because Java is closely related to C++, this is particularly true of students moving from C++ to Java.

If you believe that you need a more structured aid to learning Java, the 1-credit course CS382 presents the basics of the language in a self-paced form. In fact, much of the course material for CS382 will be listed as required reading for this course.

When the CS330 Outline page lists a section of 382 for readings, students are expected to both read the 382 lecture notes and to do the associated 382 labs. You will not be expected to do the 382 assignments or to take the 382 exam.

CS330 will pick up with Java topics more advanced than are currently covered in 382, including multithreading and the development of GUIs and in general will explore how to use Java in a truly Object-Oriented style.

You may, if you wish, choose to register for CS382 and earn credit for it. In that case, you will be doing largely the same set of readings, but will then be obligated to do the assignments and exam for that course.

3.4 Unix

All students in the course will receive accounts on the CS Dept. Unix network, and knowledge of how to work with the Linux servers is part of the course prerequisites (CS 252). This course does not require familiarity with shell scripting. All other topics in CS 252 are required. Some assignments will require the use of software available only on the Linux servers. Others may require (or, at least, be simplified by) use of the X windowing system.

If you feel that you need review of this material, visit the CS252 website, linked above.

3.5 General Computer Literacy

You will be studying techniques in this course for preparing professional-quality software documentation. The key embedded word in "software documentation" is "document". Students taking this course should be able to use word processors and other common tools to produce good quality documents, including mixing text and graphics in a natural and professional manner.

4 Assignments

Assignments for this course will include *programming assignments* (in C++ and Java), which must be done on an individual basis, and *design assignments*, which may be done in small teams.

4.1 Computer Access:

Students will need an account on the CS Dept. Unix network to participate in this class. This account is unrelated to any University-wide account you may have from the ODU's Office of Computing and Communications Services (OCCS).

If you have had a CS Unix account in the recent past, you should find it still active with your login name, password, and files unchanged. If you have had an account and it has not been restored, contact the CS Dept systems staff at root@cs.odu.edu requesting that it be restored.

If you do not yet have such an account, go to the CS Dept. home page and look for "Account Creation" under "Online Services".

All students in this course are responsible for making sure they have a working CS Unix account prior to the first assignment.

Students will have access to the Dept's local network of Unix workstations and PC's in Dragas Hall and in the E&CS building. All students can access the Unix network and the Virtual PC Lab from off campus or from other computer labs on campus.

4.2 C++ compiler

The "official" compiler for this course is the Free Software Foundation's g++ (also known as gcc or GNU CC), version 4.6.3 or higher (with C++11 features turned off). This is the compiler that the instructor and/or grader will use in evaluating and grading projects. If you have access to other compilers, you may use them, but *you* are responsible for making sure that their projects can be compiled by the instructor and/or the course's grader using the official compiler.

You may want to develop your programs on the most convenient compiler and then port it over to the official environment. Please don't underestimate the amount of time that may be involved in coping with subtle differences among compilers.

You can do all work in this course using g++ on the CS Dept Unix servers via ssh/X. If you like, however, you can obtain the g++ compiler for free from a variety sources. Links will be provided on the course Library page.

4.3 Java compiler

The official compiler for this course is Java 1.7 from Oracle. For selected assignments, the instructor may impose limitations on the language and library features that can be employed (e.g., to maintain compatibility with the Java engines built in to the commonly available web browsers).

As with C++, this compiler will be available on the Dept Unix servers, but students may also download and install a free copy on their own machines via links provided on the Library page.

5 Course Policies

5.1 Due Dates and Late Submissions:

Late papers and projects and make-up exams will not normally be permitted.

Exceptions will be made only in situations of unusual and unforeseeable circumstances beyond the student's control, and such arrangements must be made prior to the due date in any situations where the conflict is foreseeable.

"I've fallen behind and can't catch up", "I'm having a busier semester than I expected", or "I registered for too many classes this semester" are *not* grounds for an extension.

Extensions to due dates will not be granted simply to allow "porting" from one system to another. "But I had it working on my office PC!" is not an acceptable excuse.

5.2 Academic Honesty:

Everything turned in for grading in this course must be your own work. Some assignments may be done by small teams, in which case the submitted material must be the work of only those team members. Such assignments will be clearly marked as team assignments. Where teams are permitted, specific guidelines will be given in the online assignment description. In the absence of any such *explicit* statement, an assignment must be performed by a single individual.

The instructor reserves the right to question a student orally or in writing and to use his evaluation of the student's understanding of the assignment and of the submitted solution as evidence of cheating. Violations will be reported to Student Judicial Affairs for consideration for punitive action.

Students who contribute to violations by sharing their code/designs with others may be subject to the same penalties. *Students are expected to use standard Unix protection mechanisms (`chmod`) to keep their assignments from being read by their classmates. Failure to do so will result in grade penalties, at the very least.*

This policy is *not* intended to prevent students from providing legitimate assistance to one another. Students are encouraged to seek/provide one another aid in learning to use the operating system, in issues pertaining to the programming language, or to general issues relating to the course subject matter. Student discussions should avoid, however, explicit discussion of approaches to solving a particular programming assignment, and under no circumstances should students show one another their code for an ongoing assignment, nor discuss such code in detail.

5.3 Attendance:

Attendance at classes is not generally required, but students are responsible for all material covered and announcements made in class. Consequently, if you are going to miss class, be sure to get notes, handouts, etc., from another class member.

5.4 Grading:

Assignments:	40%
Midterm Exam:	25%
Final Exam:	35%

It is my general policy that, should a student perform significantly better [1] on the final than upon the midterm, or should a student have one project grade that is significantly lower than the rest, to waive that single low grade (adjusting the percentages of the remaining grades accordingly).

For the truly curious, some further information on my approach to grading is available.

5.5 Topics

Pre-OO: ADTs and Classes A review of the concept of Abstract Data Types and the use of C++ classes to implement them.

OOAD (Object-Oriented Analysis and Design) 1: Domain Models The Unified Process Model, reading common documents to discover classes, modeling an application domain with CRC cards and UML class relationship diagrams

OOP (Object-Oriented Programming) 1: C++ Inheritance and dynamic binding in C++

OOAD 2: Analysis Models: Use-case scenarios, modeling with UML interaction diagrams

OOP 2: Java Introduction to Java, self-checking tests, inheritance and subtyping in Java

OOP 3: Applying OO Techniques Container classes, Graphic User Interfaces, multi-threading

OOAD 3: UML Models: Making the transition from analysis to design to implementation

[1] a rise in class rank of at least 1.25 standard deviations