CSCI 245 SYLLABUS

COURSE NAME, NUMBER	Programming II: Object-Oriented Design, CSCI 245
SEMESTER, YEAR	Spring 2010
INSTRUCTOR	T. VanDrunen
OFFICE / TELEPHONE / EMAIL	Armerding 112 752-5692 Thomas.VanDrunen@wheaton.edu
OFFICE HOURS	MWF 3:15–4:45 pm; Th 9:00-11:30 am
COURSE WEBSITE	`http://cslab.wheaton.edu/~tvandrun/cs245`

RESOURCES
Savitch, Walter. *Absolute Java*, fourth edition, Addison Wesley, 2010.
McDowell, Charlie. *On to C*, Addison Wesley, 2001.

COURSE DESCRIPTION

A continuation of CSCI 235. Searching and sorting algorithms, their analysis and instrumentation. Software development methodology including revision control and API production. Object-oriented programming, subclassing, inheritance, overriding, and class hierarchies. Abstract data structures including linked lists, stacks, queues, and trees. Software design patterns. Introductory system programming, data representation, and computer organization.

GOALS AND OBJECTIVES

1. Students will be able to analyze the complexity of simple algorithms.
2. Students will be able to manage non-trivial software development projects in an integrated development environment and using version control systems.
3. Students will be able to design and implement non-trivial software systems.
   - Using object-oriented principles and syntax to design and implement type and class hierarchies, including the correct use of abstraction, polymorphism, subtyping, subclassing, inheritance, and overriding.
   - Using UML to manifest the system structure.
   - Using standard data structures.
   - Using widely-known software design patterns.
4. Students will be able to write programs that use basic concurrency.
5. Students will be able to demonstrate their understanding of basic systems, computer organization, and data representation by writing simple C programs.

ASSESSMENT PROCEDURES

1. Quizzes will discipline students to stay current with terminology and concepts in class and help them identify concepts on which they need more work. Specific quizzes will test their ability to analyze algorithms, use and describe Java syntax for object-oriented programming, design class hierarchies, and connect problems with solution patterns.
2. Labs will reinforce the material in class by providing practice and experience. Specific labs will force students to practice software project management, the implementation of class hierarchies, the use and implementation of data structures, and the syntax of the C programming language.
3. Projects will teach students to put ideas from class together for solving larger problems and mark their progress in that ability. Specific projects will require the management of non-trivial software projects, the design and implementation of class hierarchies, the implementation of software design patterns, and system programming. At least one project will be a team project.
4. Tests and the final exam will evaluate students' mastery of the comprehensive material.

Grading:

	weight
test 1	12
test 2	12
quizzes and short exercises	5
labs	10
projects	35
group project	10
final exam	16

SPECIAL EXPECTATIONS

Lab protocol

Thursday, 1:15–3:05 is our lab block. Our laboratory activity will follow a specific protocol called *pair programming*. Two students will work together at one computer, producing a single product, sharing two roles: The *driver* controls the mouse and keyboard and does the actual programming; the *navigator* watches the driver, catches simple mistakes, thinks of ways to test what is currently being programmed, and thinks ahead to the next task in the lab. Students in a pair switch roles between each sub-task, approximately every ten minutes. The program is produced through discussion and collaboration; neither member of the pair should dominate. While you work, your computer will be logged in through a class account; do not log in as yourself during lab unless you are specifically instructed to do so.

Academic Integrity

Since pair-programming is practiced in *labs*, students sometimes find it unclear what constitutes fair collaboration in *programming projects*. You are encouraged to discuss the problem in the abstract with your classmates; this may include working through examples, drawing diagrams, and even jotting down some pseudocode. If you are really stuck on a compiler error or bug (meaning that you have tried to figure it out for a long time are are stumped), you may ask someone to look at your code to help you find it. Sharing test cases is also a great way to help each other.

What is not allowed is sharing code. You may not program together, and you may not watch each other programming for projects, either to give or receive help. Although it says above that erroneous code may be looked at if the student is stuck, *working* code should be not be shown. Think in analogy with problems sets in a math or science course: while it is ok to help each other find the right place to look for the answer or discern why an answer is not working out, you should not give or receive the answer. Moreover, downloading relevant code from the Internet is manifestly dishonest.

While getting *code* from the Internet is cheating, you may find electronic and print resources helpful in getting *ideas* for a project. In this case, think in analogy to avoiding plagiarism in a research paper: if you use resources like this, it is crucial that you cite them in your comments.

Along these lines, if you do receive help beyond what is fair from outside resources (including the Internet) or a classmate, you should give credit. While I reserve the right to deduct points if I judge you to have profited unfairly, I will be very lenient if you are up front and honest about it.

Late assignments

You are allowed a total of two days during the course of the semester—either one assignment two days late or two assignments one day late each. Other late assignments will not be accepted.

Attendance

While I was an undergraduate, I missed a grand total of two classes, and one of them was to take the GRE. I expect the same from my students. Since being a student is your current vocation, since your learning now will affect your ability to support a family and church later in life, and since you, your family, and/or a scholarship fund are paying a large sum of money to educate you, being negligent in your schoolwork is a sin. I do not take attendance, but I do notice. When missing a class is unavoidable, it is courtesy to inform the instructor, ahead of time if possible.

Special needs

Whenever possible, classroom activities and testing procedures will be adjusted to respond to requests for accommodation by students with disabilities who have documented their situation with the registrar and who have arranged to have the documentation forwarded to the course instructor. Computer Science students who need special adjustments made to computer hardware or software in order to facilitate their participation must also document their needs with the registrar in advance before any accommodation will be attempted.

Examinations

There will be two tests this semester, currently scheduled for Feb 19, and Apr 9. The final exam is Wednesday, May 5, at 8:00 AM. I do not allow students to take finals early (which is also the college's policy), so make travel appropriate travel arrangements.

Office hours

In the past I have tried to keep an "open-door" policy, that is, I don't mind if you stop by even when it is not my office hours. However, to help manage my load this semester, I've decided to reserve Tuesdays for uninterrupted work. Accordingly, *please to do not come to my office on Tuesdays without an appointment.* If you need help on a Tuesday, please make an "appointment" by sending me an email asking something like, "May I stop by and ask a question right now?" I will likely say yes. (Also, any time my door is closed, it means I'm doing something uninterruptable, such as making an important phone call. Rather than knocking, please come back in 5 minutes or send me an email.)

CSCI 235

I. Prolegomena **2 weeks**

A. Algorithms
B. Compilers, the VM model
C. Java history, intro

II. Programming fundamentals **4 weeks**

A. Types, variables, expressions, statements
B. I/O, Strings
C. Flow of control
D. Arrays
E. Modularity and methods
F. Recursion

III. Object-oriented fundamentals **5 weeks**

A. Encapsulation
B. Class and object
C. Interfaces
D. Subtype polymorphism

IV. Other topics **4 weeks**

B. Exceptions
C. Collections
D. Streams and file I/O
E. GUI

CSCI 245

V. Algorithms and analysis (CSCI 345) 1.5 weeks

We begin the course with a brief study of algorithms, with the main goal of developing a formal means of categorizing algorithms' complexity and efficiency. Sorting and searching are classical problems used as examples. We also consider experimental measures of algorithms' performance as a point of comparison to the formal approach.

A. Analysis
B. Sorting
C. Searching
D. Instrumentation

VI. Software development (CSCI 335) 1 week

We make a short exploration of good programming practices and tools.

A. Eclipse
B. Revision control (SVN)
C. JUnit
D. Software life-cycle

VII. Object-oriented programming (CSCI 235) 3 weeks

This is the heart of the course, from which the course gets its name. After reviewing the fundamental object-oriented principles of class design, subtyping, and polymorphism, we consider the more advanced concerns of code-sharing, inheritance, overriding, and class hierarchies. We also learn modeling tools and concepts and terms used to talk about good object-oriented designs (responsibility, coupling, code reuse, and refactoring). Some time is also spent looking at several lesser-known aspects or features of the Java programming language.

A. Mechanics
  1. Review of class/object/polymorphism
  2. Abstract classes and class extension
  3. Standard class methods

B. OO design
  1. UML
  2. Aggregation, acquaintance, delegation
  3. Responsibility, coupling, cohesion
  4. Refactoring

C. Java details
  1. For-each loop
  2. Enum types
  3. Nested classes
  4. Generics

VIII. Data structures (CSCI 345) 2 weeks

We undertake a basic study of data structures with two goals in mind: understanding the principles of structuring data (logical ordering versus physical implementation, linked structures versus contiguous memory), and being familiar with the most common data structures.

A. General; linked vs. array-based
B. Stacks and queues
C. Other data structures

IX. Concurrent and event-driven programming 2 weeks

This is a new module in this course; details will be forthcoming.

IX. Design Patterns (CSCI 335) 1.5 weeks

Design patterns are reusable conceptual solutions to common problems in software development. We survey a few of the simpler, widely recognized patterns. We will likely touch on other design patterns elsewhere in the course of the semester.

B. Factory Method
C. Strategy and State
D. Adaptor and Decorator

X. Systems (CSCI 351) **3 weeks**

In the final module of this course, we learn the fundamentals of computer organization—including representation of information, dynamic allocation, and bit manipulation. This is imbedded in a tutorial on the C programming language. We also examine a pseudo-assembly/machine language.

A. C programming
B. Representation
C. Circuits
D. Computer organization

Several running examples recur in labs, projects, and in-class examples: adventure games, hand-held 4-function calculators, predator-prey simulation, and language interpreters.