CSCI 245 SYLLABUS

COURSE NAME, NUMBER	Programming II: Object-Oriented Design, CSCI 245
SEMESTER, YEAR	Spring 2011
INSTRUCTOR	T. VanDrunen
OFFICE / TELEPHONE / EMAIL	Sci 163 752-5692 Thomas.VanDrunen@wheaton.edu
OFFICE HOURS	MWF 1:30-3:30 pm; Th 10:20-11:30 am
COURSE WEBSITE	`http://csnew.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.

INFORMAL DESCRIPTION

Our original intent for this course, as suggested by the catalog copy, was to make the introductory programming course a two-semester sequence. As the course has developed, we have begun to move it away from being a plain continuation of CSCI 235 to a stand-alone intermediate programming course.

This course has always served as a gateway to the field of computer science and to the rest of our curriculum. The hope is to make this course do that more effectively. The course teaches you a little bit of everything. For CSCI majors and minors, it serves as a prerequisite for the next level of courses (CSCI 335, CSCI 345, and CSCI 351). For students from other majors who are are taking this as a terminal experience in computer science and programming, this course serves you a miniature version of the major.

The subtitle indicates one major theme of the course, **object-oriented design**. However, as the course developed, we have increased the prominence of two other themes in response to students' needs (and, to a large extent, in response to student feedback): **systems programming in C** and **data structures**. Minor themes include advanced features of Java, analysis of algorithms, software development, and concurrency.

Since many ideas in these themes are interconnected, we will be pursuing them in parallel, using a spiral approach. The course is not divided into modules, but will iterate through the various topics, continually going deeper and letting the topics illuminate each other.

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.

Grading:

	weight
projects	40%
labs	10%
quizzes	5%
homework	5%
test 1	10%
test2	10%
final exam	20%

SPECIAL EXPECTATIONS

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 pseudo-code. 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.

Collaboration is generally permitted on homework problems, since their purpose is primarily instructive, not evaluative. If you do work together with another student, however, make sure that you contribute equally; the homework assignment will do no good to a passive study partner.

Late assignments

For *projects*, 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. Late projects beyond this will not be accepted.

No credit will be given for late homework problems.

Attendance

Students are expected to attend all class periods *on time*. It is courtesy to inform the instructor when a class must be missed.

Examinations

The final exam is Tuesday, May 3, at 1:30 PM. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements. See the course website for dates for the two tests; if these dates change, I'll announce it in class.

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.

Dress and deportment.

Please dress in a way that shows you take class seriously—more like a job than a slumber party. (If you need to wear athletic clothes, for example, because of activities immediately before or after class, that's ok, but try to make yourself as professional-looking as possible.) If you must eat during class (for schedule or health reasons), please let the instructor know ahead of time; we will talk about how to minimize the distraction.

Electronic devices.

Please talk to me before using a laptop or other electronic device for note-taking. I will discourage you from doing so; if you can convince me that it truly aides your comprehension, then I will give you a stern warning against doing anything else besides note-taking. Trying out programming concepts on your own during classtime is not productive because it takes you away from class discussion; that is what lab time is for. You cannot multi-task as well as you think you can. Moreover, please make sure other electronic devices are silenced and put away. ***Text in class and DIE.***

Office hours. I try to keep a balance: Stop by anytime, but prefer my scheduled office hours. This semester I am trying to reserve Tuesdays for uninterrupted work. This means that if it is possible to hold your question for another day, then that would be great, but if it is urgent (ie, holding up your progress on a project), then no problem, stop by even if it's a Tuesday. Also, any time my door is closed, it means I'm doing something uninterruptable, such as making an important phone call. Do not knock; please come back in a few minutes or send me an email.

Projects See the course schedule for approximate assignment dates and due dates for the projects. Until each project is actually assigned, these may be subject to change. Once the project description is linked-in, it is officially assigned (I will announce this in class or lab), and the due date will not change without notification and good reason.

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, or 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.

Most labs will have a pre-lab reading on the course website. Make sure you read the pre-lab reading *ahead* of time. Quickly glancing over it as lab begins defeats the purpose. The pre-lab reading page will have a place for you to sign-in that you have read it. I will be able to tell when you have done that.

Some advice. In recent semesters I have had some students stumble in this course—in many cases, I feel it could have been prevented. A lot of information needed in future CSCI courses is packed into this semester. The course needs to be taken seriously. Here are a few bits of advice on how best to manage this course:

- **Start your projects early.** The projects in this course are not sit-down/code-it-up/test-test-test/turn-it-in kind of projects. They are think-think-think/design-design-design/plan-tests/code/verify-tests kinds of projects.
- **Read the pre-lab readings.** They are there for a reason. They will make lab experiences much less frustrating.
- **Keep up with the material.** The material in this class keeps on building on itself. If you don't understand something, don't just shrug it off and move on. Even if it doesn't seem like last week's material is being used this week, last week's material is going to come back later.
- When all else fails, **ask for help.** Your instructor, your TA, and many friendly lab rats are there to help you succeed.

The following is a summary of the topics under each major theme. As the course will cover these topics in a spiral approach, see the course website for a schedule.

I. C programming and computer systems (CSCI 351)
The C programming language is an important tool for understanding how a computer works at a low level: how information is represented, how instructions are executed, and how to accomplish computational tasks using the most basic tools.

A. Basic C (types, control structures, etc)
B. Arrays and strings
C. Functions and prototypes
D. Compiling and linking
E. Structs
F. Computer memory and representing information
G. Pointers
H. Dynamic allocation of memory
I. Bit operations
J. A model of machine execution
K. Function call and return
L. Function pointers

II. Analysis of algorithms (CSCI 345)

A crucial part of computer science is the formal modeling of efficient use of resources, especially computational time. We study methods for analyzing the complexity of iterative and recursive algorithms and compare the theoretical results with experimental findings. Along the way we examine well-known sorting algorithms and touch on proofs of program correctness. The core material is all introduced in the first three weeks, but revisited and used throughout the semester.

- A. Sorting
- B. Loop invariants
- C. Algorithmic complexity
- D. Analysis of recursive algorithms
- E. Searching
- F. Instrumentation

III. Java programming (CSCI 235)

This theme is a "post-requisite" of CSCI 235. Basic object-oriented programming in Java programming is reviewed and advanced features are explored.

- A. Review of basic classes
- B. Review of interfaces and polymorphism
- C. Review of static members
- D. Review of linked lists
- E. Review of Java Collections classes
- F. Review of basic GUI in Java
- G. Extended for loops
- H. Enum types
- I. Nested classes
- J. Abstract classes and inheritance
- K. Generics

IV. Software development (CSCI 335)

We learn some tools and techniques for managing software projects, both in C and Java

- A. Makefiles
- B. Eclipse
- C. Documentation generation using Javadoc
- D. Revision (version) control using Mercurial
- E. Software life cycle
- F. Refactoring
- G. (Time permitting) Unit testing using JUnit

V. Abstract datatypes (CSCI 345)

Our study of data structures has 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. Sets
- C. Maps
- D. Stacks and queues
- E. Trees, especially BSTs
- F. Heaps and priority queues

VI. Object-Oriented Design (CSCI 335)

Object-oriented design is about putting the OO features of a language like Java to good use: Designing software to be robust, maintainable, and reusable at a high-level of abstraction (ie, the interaction of classes and objects).

- A. UML
- B. Goals of good design
- C. Refactoring
- D. Design patterns
    1. Template method
    2. Factory method
    3. Mediator
    4. Singleton
    5. State
    6. Strategy
    7. Adaptor
    8. Decorator

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