CSCI 245 SYLLABUS

COURSE NAME, NUMBER	Programming II: Object-Oriented Design, CSCI 245
SEMESTER, YEAR	Fall 2010
INSTRUCTOR	T. VanDrunen
OFFICE / TELEPHONE / EMAIL	Sci 163 752-5692 Thomas.VanDrunen@wheaton.edu
OFFICE HOURS	MWF 9:15-10:20 am; Tu 2:15-3:45 pm; Th 9:00-11:30 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

This semester I'm "rebooting" this course. I'm moving 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. Even though "Object-Oriented Design" is in the course's title, in reality, that is only one of several themes in the course, the others being the analysis of algorithms, software project management, concurrency, computer systems, and systems programming in C.

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
"easy" homework	5
"hard" homework	15
quizzes	15
final exam	15

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 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.

"Hard" homework problems must be done completely independently. Help from outside resources, including the Internet, may not be used. Unless specified otherwise, assume these are closed book.

Collaboration is generally permitted on *"easy" homework problems*.

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.

"Hard" homework problems will not be accepted late. Similarly, no credit will be given to late *"easy" homework problems.*

Attendance

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

Examinations

The final exam is Tuesday, Dec 14, at 8:00 AM. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements.

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.

Office hours.

I try to keep a balance: Stop by anytime, but prefer my scheduled office hours. This semester I am trying to reserve Tuesday mornings for uninterrupted work. If possible, please find another time to stop by with questions (but if the matter is urgent or no other time is possible, then you may still stop by then). 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 a few minutes or send me an email.

Dress and deportment.

Please dress in a way that shows you take class seriously. Do not wear sweat pants or other slumber-party wear to class. (If you need to wear athletic clothes because of activities immediately following class, that's ok, but try to make yourself as profession-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.

Do not use laptops during class (unless you are using it for a presentation or similar purpose). If you feel you will take better notes on your laptop, then talk to me. I will give you a stern warning against doing anything else besides note-taking on your laptop during class. Please make sure other electronic devices are silenced and put away. ***Text in class and DIE.***

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.

Tests, homework, etc. We don't have time for tests this semester. Homework problems and quizzes will take the place of tests. There are two categories of homework: "Easy" homework problems are for your own preparation for class periods, and I will check only that you have done them. We'll go over the answers in class. "Hard" homework problems are to be treated like mini take-home tests. Work on them independently. You may ask me for clarifications, but do not expect help on them. They will be graded carefully. Also, quizzes (given most Thursdays before lab) will test how well you keep up with vocabulary and other concepts.

Specially scheduled days. Two special events will disrupt normal class schedules this fall. On Friday, Sept 17, President Ryken will be inaugurated. On Friday, Oct 1, the new science building will be dedicated. *Our class is not affected by either of these. We will meet as regularly scheduled on both days.*

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 seariously. 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.

I. Algorithms and analysis (CSCI 345) **2.5 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. This is also an opportunity to introduce the fundamentals of the C programing language.

A. Basic C
B. Sorting
C. Loop invariants
D. Algorithmic complexity
E. Searching
F. Instrumentation
G. Managing a C project
    1. Makefiles
    2. Revision control

II. Object-oriented programming I (CSCI 335) **3 weeks**

The technical material in this section is review from CSCI 235. However, we will examine conceptual and design aspects carefully. This will also introduce some software development concerns, including project management.

A. Review
    1. Classes and objects
    2. Interfaces and polymorphism
    3. Java Collections
    4. Linked lists

B. Design concepts
    1. UML
    2. Aggregation and composition
    3. Encapsulation
    4. Responsibility-driven design
C. Managing a Java project
    1. Eclipse
    2. Javadoc
    3. JUnit
    4. Refactoring
D. Design pattern: Factory Method

III. Concurrency (CSCI 335/351) **1.5 weeks**

We learn the basic of writing concurrent programs, including event-driven programming for GUIs.

A. Concepts of concurrency
    1. Threads
    2. Race conditions
    3. Deadlocks
B. Concurrency in Java
    1. The `Thread` class
    2. The `Runnable` interface
    3. Monitors and locks
    4. Synchronized methods and blocks
C. Concurrency patterns
    1. The Actor model
    2. Pipe and filter
D. Event-driven programming and GUIs using Swing

IV. Object-oriented programming II (CSCI 335) **2 weeks**

We learn code-sharing, inheritance, overriding, and class hierarchies. Some time is also spent looking at several lesser-known aspects or features of the Java programming language.

A. Class extension
    1. Basic subclassing
    2. Details
    3. Overriding
B. Advanced features
    1. Enum types
    2. Extended for loops
    3. Nested classes
    4. String interning
    5. Finalizers
C. Generics
D. Design pattern: Singleton

V. Computer memory (CSCI 351) **1.5 weeks**

We look at how information is laid out on computer hardware. We also consider how this is represented in C.

A. Computer representation of information
    1. Binary representation
    2. Binary arithmetic
    3. Bit operations
B. Structs
C. Pointers
D. Dynamic allocation

VI. 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. Trees, especially BSTs
D. Other data structures
    1. Heaps and priority queues
    2. Sets
    3. Hash tables

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

We examine a pseudo-assembly/machine language and consider other aspects of how a program runs.

A. Machine code
B. Function call and return
C. Function pointers

VIII. 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.

A. Introduction and overview
C. Strategy and State
D. Adaptor and 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.