# CSCI 245

## Programming II: Object-Oriented Design

Fall 2013 MFW 12:45–1:50 pm SCI 131

http://cs.wheaton.edu/~tvandrun/cs245

**Thomas VanDrunen** ☎630-752-5692 630-639-2255 ✉Thomas.VanDrunen@wheaton.edu
Office: SCI 163 Office hours: MWThF 3:15–4:45; Th 11:00–12:00.

---

## *Contents*

**CATALOG DESCRIPTION.** A gateway to the computer science major, introducing a range of themes in the field of computer science. Object-oriented programming in Java or a similar language: code reuse with composition and inheritance; generic types; design patterns. Software development: development tools, attribues of good design. Algorithm analysis; searching and sorting algorithms. Abstract data types: stacks, queues, trees, hashing; linked vs array-based implementation. Systems programing in C; pointers and dynamic allocation; model of machine memory, ogranization, and execution. Prerequisites: CSCI 235 or department approval.

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

**PURPOSE OF THE COURSE.** For students taking this as their final computer science course, this equips them with the tools they need for many programming tasks, and it abridges much of the content of other courses in the computer science program.

For students going on in the computer science program, this course prepares them for the programming tasks in later courses and introduces themes carried on throughout the curriculum. This course is the "super-prerequisite" for the computer science major.

**THEMES.** The content of this course is organized under a variety of themes which will be pursued concurrently throughout the semester.

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

- A review of classes, interfaces, polymorphism, and other basic OOP features.
- A review of Java Collections.
- Class extension (subclassing, inheritance).
- Nested classes.
- Generics.
- Enum types.

**Software development.** (CSCI 335) *We learn some tools and techniques for managing software projects, both in C and Java.*

- Software lifecycle.
- Documentation.
- IDEs.
- Revision control.
- Makefiles.
- (Time permitting) Unit testing.

**Object-oriented design.** (CSCI 335) *We put the object-oriented features of a language like Java into good use, learning design principles and programming idioms and patterns.*

- Elements of good software design.
- UML.
- Refactoring.
- Design patterns.
  - Basic creational patterns.
  - The State pattern.
  - The Strategy pattern.
  - The Adaptor pattern.
  - The Decorator pattern.

**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.*

- Loop invariants.
- Analyzing an iterative algorithm.
- Big-oh notation.
- Analyzing recursive algorithms.
- Empirical measurements of performance.
- Sorting.

**Abstract data types.** (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 becoming familiar with the most common data structures.*

- Linked lists and other linked structures.
- Abstract vs concrete types.
- Stacks.
- Queues.
- Binary trees.
- Hashing.

**C programming.** (CSCI 351) *The C programming language is an important tool for understanding how a computer works at a low level. Throughout the semester we will compare implementations of various ideas in C and Java.*

- C basics.
- Functions and prototypes.
- Compiling, linking, preprocessing.
- Structs.
- Pointers and dynamic allocation.
- Strings.
- Bit operations.
- Function pointers.

**Computer systems.** (CSCI 351) *We consider how a computer really works: how information is represented, how instructions are executed, and how to accomplish computational tasks using the most basic tools.*

- Model of computer memory.
- Model of execution.
- Programming in a pseudo-assembly language.
- Model of function call and return.

**Concurrency.** *As multi-core architecture has become mainstream, programming skills are not complete without some competency for writing programs with multi-threading. Moreover, GUI programming cannot be done the right way without an understanding of threads.*

- Threads.
- Race conditions and dead locks.
- Java support for concurrency.
- GUI and event-driven programming.

The above list is in no way chronological. See the course website for a schedule.

## *Course procedures*

**How we do this course.** Class time is used to introduce new ideas and work through examples. I use a lot of handouts. Lab time is used to practice the ideas with help from peers. Most of your work outside of class is spent on projects (one project every week-and-a-half to two weeks). There also are occasional written assignments to reinforce concepts from class and prepare you for the next class. Usually I will check them for completeness and go over the solutions in class. The textbooks are mainly languages references, though occasionally you may be asked to read something before class. Quizzes will be given at the beginning of lab time; their main use is as practice test questions.

**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 at what time you have done that.

**Some advice.** In past 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.

**Grading.** There will be two tests (scheduled for Oct 4 and Nov 15, subject to change) and a final (Wednesday, Dec 18, 1:30 pm).

| *instrument* | *weight* |
|---|---|
| Projects | 40 |
| Labs | 10 |
| Quizzes and homework | 5 |
| Test 1 | 12.5 |
| Test 2 | 12.5 |
| Final exam | 20 |

**Projects.** I estimate 8 projects in this course. See the schedule on the course website for approximate assignment and due dates.

## *Policies etc*

**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 candid 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 late 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. It is courtesy to inform the instructor when a class must be missed.

**Examinations.** The final exam is Wednesday, Dec 18, at 1:30pm. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements.

**Gender-inclusive language.** For academic discourse, spoken and written, the faculty expects students to use gender inclusive language for human beings. This is not actually relevant for this course, but it's an official college policy that the syllabus contain some sort of notice like this.

**Special needs.** *Institutional boilerplate:* Wheaton College is committed to providing reasonable accommodations for students with disabilities. Any student with a documented disability needing academic adjustments is requested to contact the Academic and Disability Services Office as early in the semester as possible. Please call 630.752.5941 or send an e-mail to `jennifer.nicodem@wheaton.edu` for further information.

*My own statement:* 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. Any time my door is closed, it means I'm doing something uninterruptable, such as making an important phone call. Do not bother knocking; instead, 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—more like a job than a slumber party. (If you need to wear athletic clothes because of activities 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. You will need to convince that it truly aides your comprehension. No student has convinced me yet, but if you're the first one then I will give you a stern warning against doing anything else besides note-taking. Trying out programming concepts on your own during class time (for example) is not productive because it takes you away from class discussion. 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.***