CS 350 Syllabus

**Instructor:**Jeff Salvage **E-mail:** [jks29@drexel.edu](mailto:jks29@drexel.edu) **WWW:** <http://www.cs.drexel.edu/~jsalvage> **Lecture:** Tuesdays and Thursdays 3:30 - 4:50 PM (section 001) and  12:30 - 1:50 PM (section 002) - Crossings 151 **Office:** Crossings 112 **Office Hours:** Monday/Tuesday 4:00-4:55PM, Thursday 2:15-3:15. Often I will be in my office on Tuesday from 2:15-3:15 as well.

**Instructor:**Adelaida  A. Medlock **E-mail:** [aalban@drexel.edu](mailto:aalban@drexel.edu)  **WWW:** [http://www.cs.drexel.edu/~aalban](https://www.cs.drexel.edu/~aalban/) **Lecture:** Tuesdays 6:30 - 9:20 PM (section 003) - Crossings 153  **Office:** Crossings 107  **Office Hours:** Mondays 1:00 - 3:00 PM

**Teaching Assistants:**

Sean Grimes ([spg63@drexel.edu](mailto:spg63@drexel.edu))

Kashyap Ashok ([ka566@drexel.edu](mailto:ka566@drexel.edu))

TAs Office hours schedule: <https://www.cs.drexel.edu/clc/>

**Content**

**Week 1:**

* Intro to class
* Brief review of C++ and JAVA

**Week 2:** **Homework Part 1**, Rough Outline of UML, Assigned, Due 4/23 11:59 PM

* Intro to Design Patterns
* Chapter 1
* Chapter 2
* Chapter 3
* Introduction to the first assignment
* Review Quiz: Java Concepts

**Week 3:**

* Lab 1, due in class to be done in class. You do not have to do anything before. Please bring your laptops if you have them.
* Chapter 4
* Chapter 5
* Review Quiz: Chapters 1 – 3

**Week 4:** **Homework Part 2**, Assigned, Due 5/7 11:59 PM

* Chapter 6
* Chapter 7
* Chapter 8
* Chapter 9
* Chapter 10
* Review Quiz: Chapters 4 – 7

**Week 5:** **Homework Part 3**, Assigned, Due 5/21 11:59 PM

* Chapter 11
* Factory Method
* Abstract Factory
* Lab 2, Basic Maze
* Lab 3, Adding Rooms to a Maze
* Review Quiz: Chapters 8 – 10

**Week 6:** **Homework Part 4** Assigned, Due 6/4 11:59 PM

* Review
* Midterm - In class, bring laptop, closed book.

**Week 7:**

* Chapter 13
* Chapter 14
* Chapter 15
* Chapter 17
* Chapter 18
* Chapter 21

**Week 8:**

* Junit demo
* Junit Lab
* Review Quiz: Chapters 11-21

**Week 9:**

* Command Pattern
* Template Pattern
* Builder Pattern
* Lab: TBA
* Review Quiz: Template Command Builder

**Week 10:**

* Putting it all Together
* Review for Final
* Quiz Review

**Final Exam** TBD

**Texts**

We expect that you will have access to these books for assignments.

**Required** Design Patterns Explained:Alan Shalloway & James Trott, Addison and Wesley  **2nd** Edition

**Recommended**Design Patterns : Elements of Reusable Object-Oriented Software  by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides 395 pages (October  1994) Addison-Wesley Pub Co; ISBN: 0201633612    - DON'T BUY UNTIL WE TALK

For some examinations, we will allow you to have a Java reference book to refer  to for details on the language.  The book need not be a comprehensive  guide to class libraries, but it should allow you to write syntactically  correct Java.

**Integrated Development Environment**

The IDE for this class will be Eclipse or the community edition of IntelliJ.  You can download a free copy (having already installed Java on your computer) from [www.eclipse.org](http://www.eclipse.org/).  Being implemented in Java, it should (and does) run on Windows, Sun, Linux, and Macintosh.  The reason why we are using Eclipse is that we intend to use certain Eclipse plug-ins, such as for program testing and style checking, which makes use of such additional features straightforward across platforms. While you are free to develop your implementation projects in any environment you wish, you must submit an eclipse project when an implementation is called for (and possible for other kinds of work as well, as will be announced in assignment handouts).  Programs will be tested by the grader to execute properly in Eclipse.  You will not receive credit for implementation assignments and projects unless your submission can properly compile, execute, and pass required tests.

**Objectives of this course**

This course is about good object oriented design and implementation. In the challenging projects that programmers face, the design stage is where the primary difficulties of programming are worked out. Implementation, the task of turning a design into a working program in a working programming language, is supposed to be very straightforward given a good design.

Object oriented programming(OOP) allows programmers to use write programs that is easier to reuse and easier to extend and maintain over a life cycle of usage. However, this power is achieved at the cost of making the program designer’s task more complicated, due to the richness, subtlety and peculiar limitations of OO classes and class inheritance. A major objective of this course is to bring students to a level of competency in OO design so that they can handle team or individual projects of moderate complexity. Design skills students should acquire include the ability to decide which classes would be appropriate given only a problem specification, designing for ease of future code maintenance, being able to decide upon details of classes from a problem specification, finding well-known *software patterns* from typical problem specifications, and being able to communicate both static and dynamic aspects of designs using the Universal Modeling Language (UML).Design skills should be applicable to any of the OO languages in vogue (e.g. C++, Eiffel, Java).  Grades for design projects will be based, among on other things, on the quality of the design and on the written explanation, description, and justification of the design.

Another object of this course is to make you confident and competent at implementing an OO design of moderate complexity, using coding techniques based upon generally recognized software engineering principles. Reasonable competence in Java includes a bit more than you may have used with Java or C++ in CS 171/172/260/265 (or their equivalent). We will cover additional features and class libraries of Java that are important in object oriented programming: abstract classes, exception handling, important frameworks, etc. as well as coding and testing practices (e.g. naming conventions for classes, variables, constants, and files, pre-conditions, what thorough testing means, etc.). Many of these features can be found in other OO languages.   You will be asked to demonstrate such competence by constructing and thoroughly testing programs of moderate complexity on your own, from a design of your own devising.  Grades for implementation projects will be based, among other things, on the quality of the coding, on correctness, degree of difficulty, quality of tests devised and provided, and written documentation, explanation, and justification of the code written.

A final objective of this course is to make you more self-aware, descriptive, and reflective about your programming and design. By learning about and practicing with design and quality rationales the course will present, you should be able to state, explain, judge, and justify design or implementation decisions in a way that would be generally acceptable among (at least) entry-level computing professionals.  It is also a necessary step towards being able to continue to improve your skill in the future without the need for professors or formal classes.

**Prerequisites**

We expect that all taking this class has had CS 171/172 (programming I and II) , CS260 (Data structures), and a recent version of CS 265 (Advanced Programming Techniques) or their equivalent. The experience in these courses includes the design and implementation of at least 18 programs in C++, including several that involve the use of classes, pointers, and dynamic storage allocation. It should also include the experience of designing several CS 172-like or CS260-like programs in Java requiring familiarity of class creation, basic I/O, and the use of built-in Java class libraries.  The experience in writing these programs is also expected to have given you knowledge of basic coding practices such as: good selection of names, good quality commenting and indentation, separation of classes into separately compiled files, what constitutes appropriate selection of test data, and how to develop a program incrementally (e.g. use of program stubs as placeholders for as-yet unwritten program modules). Experience with recursive programming design and implementation is also expected.

Some assignments will assume that you find it routine to implement programs that use data structures such as trees or linked lists. Some assignments will ask that you will be able to design and implement algorithms to solve simple problems on your own, without extensive hints or outside help. This will probably require a bit more mental effort on your part than many assignments from previous courses, in that you will have to devise the solution algorithm yourself based upon your prior experience with programming problem solutions. You can use the first programming assignment as an indication of what you are expected to have started with. If you have questions, please consult with the instructor.

**Eclipse / IntelliJ and Unix**

Homework will be submitted for test execution within Eclipse, but some activities (such as using CVS) we will assume prior experience with Unix. We shall not be covering Unix usage in lecture but you may ask the instructor or consultants during office hours if you have questions about this.

**Grades and Grading**

Your grade will depend upon a number of different items, cumulatively. The weighting will be roughly as follows. The instructor reserves the right to make modest adjustments (5% or 10% for a category) in the weighting used:

* Homework Assignments 40%
* Labs 10%
* All quizzes must be taken and passed (70% is the passing grade) or you will lose 1 point on your final grade. Quizzes can be retaken if failed.
* Midterm exam 20%
* Final Exam 30%

Students will receive a reduced or a failing grade for the course if in the instructor's judgment their performance on the midterm and final examinations indicates that they cannot express an adequate understanding of the material in the course. One criterion used for "adequate" is achieving at least a 40% mark on the final exam. Good quality in-class participation (being prepared to answer questions posed by the instructor during the class) will be taken into account as a compensating factor for a poor exam performance, but programming assignments will not be.  Part of your assignment will consist of written answers rather than programs. Part of your assignment grade will be based on how you express yourself, as well as what you say. Your writing is expected to be of the quality comparable to technical writing done by a well-regarded computing professional.

In order to pass the course, you must complete the programming assignment. This includes all 4 parts including grading and tabulating.

**Handing in assignments**

Assignments will be made available on the class Blackboard Learn page, and will be due as described in each assignment. There will in the neighborhood of five assignments during the quarter, due at one or two week intervals. The programming assignments will be done in Java 1.8 using Eclipse. There will also be written work that should be submitted as specified in the assignment. Most assignments will be submitted electronically via Blackboard Learn. When assignments call for written responses, they should created using a text processor, written in intelligible, clear and correct English. Poorly written or sloppy work will be returned ungraded. Programs that do not compile and cannot be tested will receive little or no “partial credit”.

Late policy: assignments will be accepted late with a 20% penalty, by the late due date specified when the assignment is released. After that, assignments will be not accepted for credit.

**Class participation**

Part of your grade is based upon being present and having good quality participation in the class discussion. Some times the instructor will call upon people while at other times he may choose among only those who wish to speak up. Good quality class discussion means having done whatever reading has been assigned for that day and done the work of understanding most of it on your own; being able to answer questions or problems posed by the instructor based upon prior work, the reading, and the discussion in class; advancing the understanding of the class by asking questions that expand or clarify issues of interest to many; working on whatever in-class group projects are assigned for the day, and participating in the presentation of the group’s work for the project. It is not necessary to answer all questions correctly or speak up every day to receive a passing grade for participation, but you need to establish in the instructor's mind that you are typically prepared and can respond capably.

**Academic Honesty**

The university's Academic Honesty policy is in effect for this course. You can find the policy at <http://drexel.edu/provost/policies/academic-integrity/>

You must be the sole original author of all assignments and examination solutions in their entirety, unless the instructor explicitly gives you permission to do otherwise in written directions on an assignment or exam. Design and implementation problems should be discussed with the instructor or the teaching assistants only. This means that you should have an implementation plan that leaves enough time for such consultation. As the university's policy explains, penalties up to and including receiving a failing grade for the course with no opportunity to withdraw will be given for first time offenses of plagiarism, fabrication, cheating, or other forms of academic dishonesty. Allowing another student to copy from your work (i.e. helping another student to cheat) is also a violation of the policy on academic dishonesty.

The standards for originality in a program are similar to those of other written works. Programs by different authors show clear and substantial differences as judged by most criteria, including but not limited to: choice of variable and procedure names, line spacing and indentation, choice of program structure, choice of algorithms, ordering of modules, style and content of documentation, module design, and ordering and choice of instructions. The original author of an assignment can explain each detail and how they came to create it on their own. Whenever you use references or sources (i.e. the Internet) for “inspiration” in your programs and designs, you must give a complete and proper citation of the source.  Copying or close paraphrasing from other sources and claiming it is your own original work is a violation of the academic honesty policy.

It is your responsibility to avoid violating the university's policy. If you are unclear as to what the policy means in a particular situation, ask the instructor for clarification ***before*** you hand anything in.

In addition, be clear that if you electronically copy another person’s work and submit it as your own you are in violation of this policy and could be failed for the course on the first occurrence. If you solicit and/or hire someone to do your work, even if the work is not complete you are in a violation of this policy and could be failed for the course on the first occurrence. Any violation of this policy may also result in the matter being handed over to an external judiciary panel. Do not let this happen to you.

**Students Accommodations**

In compliance with the Americans With Disabilities Act of 1990, Section 504 of the Rehabilitation Act of 1973, and Drexel  University’s policies and procedures, the University is committed to the non-discrimination of students with disabilities.

Students requesting accommodations due to a disability at Drexel University need to request a current Accommodations Verification Letter (AVL) in the ClockWork database before accommodations can be made. These requests are received by Disability Resources (DR), who then issues the AVL to the appropriate contacts. For additional information, visit the DR website at <http://drexel.edu/oed/disabilityResources/students/> , or contact DR for more information by phone at 215-895-1401, or by email at [disability@drexel.edu](mailto:disability@drexel.edu)