

CS108 Syllabus

Based on a handout by Nick Parlante

The Course in a Nutshell

CS108 teaches large-scale Object Oriented Programming (OOP) using Java. The course concentrates on OOP design, both in the internal structure of its projects and in the large pre-built OOP libraries they are built on. The course features individual homework projects exercising OOP techniques and culminating in a large final team project. The course has several related themes...

- *Java and OOP programming.* We cover advanced parts of the Java language, and explore OOP design principles. Within OOP, we exercise the modern OOP themes of modularity and inheritance, OOP patterns, and unit testing.
- *Concepts and skills for OOP/GUI libraries.* OOP libraries for Graphical User Interfaces (GUIs) are a nice example of OOP design, so CS108 can use OOP/GUI libraries to make its points. The course explores the design of OOP/GUI systems and how to use them.
- *Programming in an OOP library environment.* Much of the work in an OOP system is orchestrating the behavior of the off-the-shelf library objects. In that context, the most interesting skill is operating within any large body of off the shelf OOP code. Using such libraries requires different skills from the classic from-scratch-design-code-debug cycle.
- *Team programming on a large project with a significant deadline.* CS108 gives exposure to some "real world" programming issues on the final project. The final project will take up about the last couple weeks of the quarter. The goal of the final project will be to use the OOP/GUI system to produce a complete GUI program. The final project will exercise skills in OOP design and testing, team programming, source control, project scheduling, interface design, and completing a large project under a serious deadline.

At the end of the course, you will understand how to construct a significant, fully functional GUI application from scratch as well as having gained general experience with Java programming, OOP design, unit testing, and the use of large OOP libraries.

The Course in a Nutshell — Casual Version

If you ask me in person what CS108 is about, I'm likely to say something like...CS108 is a great and very practical course. You get to play with the modern OOP technology and techniques, and you're pushed to build large, realistic projects with them. For the final project you get to work in teams to pull it all together on the largest project in the undergraduate core before the senior project. It's a lot of work, but when it comes together, it's very satisfying. CS108 is very practical; its skills and technologies will apply to many things you'll want to do in the future.

Prerequisites

Before taking CS108, students should be capable programmers at the level of CS107 or possibly CS106B: you should be comfortable writing and debugging large programs, and working with pointers and recursion. Students should know basic Java at the level of CS106A, and should be familiar with basic OOP at the level of CS106A and CS106B. Students with a strong background but who do not know Java can probably get by, picking up the language in the first week or two.

We can tolerate a wide variance in people's OOP and language knowledge coming into the course. You can work a little harder and just pick that up if necessary. However, it's important that your coding and debugging skills be solid, because it's a hard to "just pick up" those skills as you go. CS107 gives you more programming experience, and in that sense it is helpful before CS108. On the other hand, CS107 does a lot of C++ which is not especially important for CS108.

Programming Projects

CS108 is an applied project course; there are no exams and there are a lot of projects. About half of the work is in the graduated homeworks for the first 7 weeks. The homeworks cumulatively cover the major concepts of OOP and OOP/GUI programming and some other areas of the libraries. The other half of the work is concentrated in the final project for the last three weeks of the quarter. The final project is a large, team project that brings together and exercises all of the material from the first 7 weeks. The final project is programmed in teams of 3 or 4 people, and should require on the order of 60 hours per person. The final project is the final exam.

Online Materials

Many course materials and related links will be available online at our course web page <http://www.stanford.edu/class/cs108/> -- If you are looking for anything course related, look on the web page first. In particular, the course project FAQs are on the course page.

<http://www.stanford.edu/class/cs108/>

(or the alias <http://cs108.stanford.edu>)

Java

CS108 uses Java. For the most part, this frees you to build on whatever platform you wish that supports Java (Solaris, Windows, MacOS X, Linux), although you will need to turn in your work on the leland Unix systems since that's where we do the grading. Mostly, Java has great portability, so you can develop on whatever system you want, and it will run the same wherever we grade it. We will have a separate handout about turning in your code.

There are links on the course page explaining how to install the Java JDK and the Eclipse compiler. There's a page linked off the course page that explains how to compile and run Java on the various platforms. In general we will look at standard, cross-platform Java code. We will cover the most important features of the Java language, but not the whole language (Java has gotten to be a fairly big language).

Readings

There is no required text for the course. Programming against a large OOP library does require reading — it just happens to all be online in the docs and source code. There are links for the many online Java resources on the course page.

You want to get accustomed to finding and reading materials online. They are up to date, searchable, and cross-referenced. Paper copies are quickly out of date and are hard to search. Of course paper copies are nicely portable and look good. Perhaps the best tradeoff is: deal with things online first, and print selections when you feel like seeing them on paper.

Lecture Handouts

There will usually be handouts to accompany lecture. The handout will contain the source code for the day's lecture and outline notes of what I thought I was going to say. The PDF versions of the handouts should be available at least an hour before lecture.

I'll make enough paper copies of the handouts for the people in lecture. Leftover paper copies of the handouts from class also go in the bins down the hall from my office. If I make too few handouts for lecture, I'll make more and put them in the bins after class. Once those run out, please use the electronic versions. We'll make plenty for class time, and when they're gone they're gone.

Teaching Staff

Dr. Patrick Young, Lecturer
patrick.young@stanford.edu
 Gates 194

Teaching Assistants:

Red Daly <reddaly@stanford.edu >,
 Orr Keshet <okeshet@stanford.edu >,
 Sean Meador <smeador@stanford.edu>

Office hours

The exact staff office hour scheduling will be published separately on the course web page. We will have some office hours that are constant every week and we will add extra evening hours for the three days leading up to each homework due date.

Email Questions: cs108@cs.stanford.edu

We'll maintain a centralized e-mail question answering address at cs108@cs.stanford.edu. Please email your questions there. If the answer to a question seems generally interesting, we'll make it accessible to everyone in the FAQ section of the course page. Try to avoid a subject lines like "question" or "help" — use a couple descriptive words "serialization problem" or "listener won't hear".

For many questions, email works great. Sometimes you're seeing some bizarre symptom that we can diagnose in 10 seconds, and our quick answer can save you hours of stumbling around the problem. Sometimes "stumbling around a problem" is educational, but at some point you're just spinning your wheels and getting annoyed. On the other hand, if your question is going to require stepping through code, looking at variables, etc. ...it's probably better to bring it to office hours so we can look at it properly. When framing your question, try to articulate what you are trying to do, what you have tried, and what you think is going wrong. Some questions work well by email. Some questions work best by coming to office hours, or calling during office hours so at least there's a dialog (See "Debugging Skills" below). Since CS108 is such an applied course, we'll try to have as many in-person office hours as we can to help you work through you specific programming problems, but we don't have an armada of helpers like in 106 :(.

Debugging Skills

In addition to all the obvious OOP/GUI material, CS108 has a goal to try to teach those incredibly useful real-world diagnosis, debugging, and coping skills. In office hours, we'll try to bounce the questions back at you in the most useful way: "so what are the symptoms you observe? What are some theories on what might be causing it? What debugging code have you put in to illuminate the situation How could you use the debugger to test those theories?"

Paperless Submission

We will use electronic submission methods for all the homeworks, and all the grading feedback will be electronic as well. The submit program will involve copying your project directory to leland and running a submit script. You will need a leland account to submit the homeworks. There is a separate handout for the submit process.

Late Submissions

Instead of having to ask for extensions on a catastrophe by catastrophe basis, everyone gets **three** calendar "late days" to extend the due dates of any of the weekly assignments (except the last homework). In keeping with the all electronic, 24-hours a day theme of the post-Internet world, late days will be measured in straight calendar days with no distinction for weekends or holidays. If the assignment handout says it is due "Wed Sep 16th" that means it is due by midnight at the end of Wed Sep 16th.

These late days are intended to deal with the ordinary events of student life, both frivolous and serious: 2 midterms that day, inadvertently spent all night playing WarCraft, disk crash, med. school interview, illness, started way too late...After your late days are used up, late work loses pretty quickly— about a half a letter grade per day on that homework. Come and see me in person in exceptional circumstances. Note that disk failure, network outages and other computer problems probably *do not* represent exceptional occurrences. Hoard your late days "just in case," or spend them early and fly with no parachute— it's up to you.

In the grade database, every homework is recorded with both its score and the number of days late it was turned in. The Great Spreadsheet Reckoning at the end of the quarter figures out if you went over your late day budget.

Giving students their own late-day supply seems more fair since all the students are on the same footing. However it means you now need to make your own decisions about when to use a late day, and when to just turn in what you have. The late days should allow you to do a better job and hopefully learn more in the cases where your schedule gets disrupted. However, three late days do not provide too much cushion. No doubt the prudent course is to try to hit the normal deadlines, and reserve the late-days for actual problems.

Honor Code

You are free to discuss ideas and problem approaches with others, but all the work you hand in should be your own creation (or the creation of your team for a team project). **In particular, sharing or copying code is not OK. There are tools we may use that do an extremely aggressive job of finding little sections plagiarism within the submissions.** The plagiarism tools are shockingly good at finding issues, and of course doing your own work is the only way to really learn the material anyway. If a student is in a bad situation, they should come and talk things over with me rather than making a big mistake. I'm reasonable.

If you feel a particular bit of collaboration may have crossed the line, just clearly cite what help you got and from whom in your project's README. You can never get in Honor Code trouble if the help is clearly credited in the README.

If we are using the Foo module, and you find the key 8 lines in the docs or in a book or on the Internet that describe how to call the Foo module best, it's fine to use those lines without comment in your README. For example, if I ask you to read a text file (a very common 8 line task), it's fine to just research that on the Internet. OOP programming is filled with episodes like that. In contrast, if I have asked you to implement or solve a substantial problem, say solving Sudoku, copying 200 lines found on the internet that solves the problem is not ok. If you are not sure, ask a staffer or simply give credit in your README so you are automatically safe.

FAQ

Do I need to take CS107 First?

No. It's useful, but not required.

I have another class at the same time. May I still take CS108?

Yes, but it's not recommended.

I already learned basic Java, why should I bother with CS108?

CS108 covers Java, but it's not a language class. It's about using a language and library to build large projects. The language material is a minor part of the course.

Is this a difficult class? Yes, but it is not harder than CS107. Several of the programming projects are quite rigorous. On the other hand, Java has a way of making things easy compared to C++. You will need to be able to start early, and program and debug independently and diligently. The staff can provide some help, but there is not the unblinking army of helpers the way there is in CS106. The good news is: when they're working right, Java and its GUI classes are awesome program building tools.

Weekly Plan

This schedule is mostly accurate, however things may shift by a day or two depending on how things go. Assignments are often due either Mon or Wed night at midnight in the week indicated on the schedule.

Week/Mon	Topics	Due that week
1 • Sep 21	Introduction. Eclipse features. Modular unit testing with JUnit.	Special section Fri afternoon on intermediate Java.
2 • Sep 28	OOP Design: basic encapsulation, client oriented design, API design, inheritance, patterns.	HW1 Basics/Unit tests (Thur)
3 • Oct 5	Finish OOP design, javadoc. Inner and nested classes. Collection class implementation.	HW2a Tetris early (Tue)
4 • Oct 12	Anonymous inner classes. Basic Swing component setup and controls. Start threading and synchronization.	HW2b Tetris complete (Tue)
5 • Oct 19	More threading. GUIs and threading. Integrating the GUI thread with worker threads.	HW3 Collections/GUI (Thur)
6 • Oct 26	Swing paintComponent() style. Repaint(). More advanced drawing, mouse tracking. JTable MVC data handling pattern, model delegation.	HW4 Threading (Thur)
7 • Nov 2	Document file save/open, serialization, XML. Basic sockets. Techniques for large projects.	
8 • Nov 9	Final project given out. Source control.	HW5 (Tue, no late days)
9 • Nov 16	Software engineering. Java implementation.	
Break • Nov 23	No Classes, Thanksgiving Holiday	
10 • Nov 30	Advanced topics. No lecture Wed.	Final Proj due midnight ending Wed Dec 2nd