

CS351: Design of Large Software

Instructor: Terran Lane

Fall 2006

Tue/Thu 2:00-3:15

ME 218

STATUS Syllabus and Frequently Asked Questions Document

VERSION 1.0

DATE Aug 22, 2006

0 Changelog

Version 1.0 Initial release. Aug 22, 2006.

1 Goals and Subjects (What We're Doing Here)

This class is about designing big software. Well, sort-of big, anyway. I'll say more about what "big" means in the first couple of sessions, but the short version is: large software is any piece of software that's too big to fit in your head at once. (Basically, everything that anybody actually cares about.)

In practice, what we're doing here is trying to learn how to build real pieces of software. Software that does something that you might conceivably care about. Which means that it's usually bigger than you can envision in a single pass. Which means that you actually have to think about design in advance.¹ And design is an art. You learn by doing. I'll teach you what I can. But everybody I know who knows how to seriously hack, learned it by doing. So, in this class, you'll do.

2 Textbook (What to Read When the Lecture Makes No Sense)

Design is really hard to teach. It's even harder to write about. I haven't found a real textbook for this stuff. The "official" (primary recommended) text for the class is:

Core Java 2: Volume 1, Fundamentals, 7th ed. Horstmann, C. S. and Cornell, G.
Sun Microsystems Press, 2005.

It should be available from the UNM bookstore or your favorite online purveyor of fine technical textbooks.

But that's mostly a language manual. Every hacker should have a language manual on the shelf (or 2 or 4), but that won't teach you about *design*. The following also won't teach you design, but they'll help. I recommend them all.

¹No, really. In advance. Like, earlier than the night before it's due. Earlier even than "when you first sit down to code".

Design Patterns, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Addison-Wesley, 1995.

Refactoring: Improving the Design of Existing Code, Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. Addison-Wesley, 1999.

Debugging: The Nine Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems, David J. Agans. American Management Association, 2002.

Effective Java, Joshua Bloch. Addison-Wesley, 2001.

The Mythical Man-Month: Essays on Software Engineering, Brooks. Addison-Wesley, 1995.

These days, there's also an *enormous* amount of useful information on Java syntax, semantics, tools, and design on the web. I encourage you to explore on your own. I maintain a page of some useful resources, but they're far from exhaustive. Look around. Oh, and beware that the web, Source of All Knowledge, is also Source of All Garbage. Read critically.

3 Class Resources (Me, the TA, the Web, etc.)

Person	Email	Office	Office Hrs
Terran Lane	terrancs@unm.edu	FEC325	T/R 3:15 PM–4:00 PM W 9:00 AM–11:00 AM
Blake Anderson	blake.h.anderson@gmail.com	FEC301A	T/F 10:30–12:00

(All times given in the Mountain time zone.)

I will (usually) hold office hours 9:00-11:00 AM on Wednesdays. I will also be available for up to 45 minutes after class for questions, brainstorming, philosophy, etc. If none of those times work for you, I'm also happy to make an appointment to meet with you — send me email. (Mr. Anderson is equally obliging.) I'll put other information about the class on the class website as it becomes available:

<http://www.cs.unm.edu/~terrancs/classes/cs351-f06/>

There is a mailing list for this class: cs351@cs.unm.edu. I *strongly* suggest that you subscribe to this list because I'll occasionally send updates/suggestions/revisions out there. It's also a good place for you to discuss stuff relevant to the class. (Go to <http://www.cs.unm.edu/cgi-bin/mailman/listinfo/cs351> to subscribe.)

If people think that a wiki would be useful, I can arrange to set one up for this class let me know.

Hint: *Please, please, please*, if you're having difficulties in the class, for whatever reason, contact me *early*. I really want all of you to enjoy and succeed in this class, but I can't help you do so if you don't talk to me. Preferably early enough to do something about it — two days before the final is probably too late make any difference! I'm happy to talk to you in office hours, or, if you're more comfortable with written communication, by email.

4 Assignments, Grading, and Handins (What You'll Do)

As I said above, learning design is all about doing. You have to write enough code to learn what works well, what doesn't work so well, and what is a complete and utter disaster. (Hopefully, you'll mostly live in the first of those spaces. But you learn by getting your toes wet in the others occasionally.) So... This is a project class. The biggest chunk of your grade will come from 3 projects, each of which will have multiple milestones. The milestones will be graded independently, but they do tend to build on each other. The last milestone of each project is the final rollout, on which the complete functionality of the project is graded.

There is a lab section for this class. You are expected to attend all of the labs and do the lab assignments. They're good for you. We promise.

There will also be a midterm and final exam. If all project groups are making strong progress on their final project, I *may* cancel the final exam and substitute your midterm grade for your final grade. (So, if you think about it, it's in your best interest to ace the midterm *and* do a great job on the final project.)

There will also be some in-class quizzes. **I WILL NOT LECTURE ON THE QUIZ MATERIAL IN ADVANCE.** I'll warn you about what's coming up, but it's your responsibility to read/study up on the appropriate stuff.

The grade breakdown for the class looks roughly like this. These numbers are subject to revision later, but they give you a good ballpark to start from.

50% Design projects. Two individual, one group.

20% Exams. (Individual)

15% Labs.

15% In-class quizzes, readings, etc.

5 CS351 FAQ

5.1 Is CS351 just about torturing students?

No. Well... [thinks] Nope, still no.

The projects seem big. Sometimes overwhelmingly so. Some people report staying up nights and weekends to get everything done. And you may find yourself going “I don’t know where to even start!” But that’s exactly why you should start early. I’m not kidding. The easiest way to bottom out in CS351 is to procrastinate and not give yourself time to really think. Some of these problems you need to sleep on to see the right way to do it. You have to give yourself time to do that.

You also need to give yourself time to write all the lines of code. This class *isn’t* really about turning out large volumes of code. It’s about turning out the *right* code. But even if you have a beautiful, elegant, concise design, there are still a lot of lines to write. You can’t do it in one night. (Ok, nobody I know has ever done it in one night. Successfully, anyway.)

If you start early, think carefully, and realize when you need to back up rather than forging ahead, then you should be able to finish everything without staying up nights and Sundays.

Well. Not very *many* nights, anyway.

5.2 What do I have to know to take this class?

Basic programming proficiency in Java. Be capable of:

- Writing a program of between 100 and 500 lines of code, including more than one class/module
- Compiling it
- Running it
- Debugging it (!)
- Basic performance testing

You should be familiar with all of the basic Java syntax and semantics: built-in data types, variable declaration, boolean and bitwise operators, control flow structures (if, while, exceptions, iteration), method definition, etc. In addition, you should have be familiar with some of the structures in the Java standard library (especially `java.util`) and I/O via `java.io`.

Of course, you also need basic familiarity with tools — editor, compiler, the UNIX operating system, `jar`, etc. Knowing your way around some form of scripting (sh, perl, python, etc.) would also be helpful, but is not required.

You should have basic understanding of computer architecture and layout, what memory really means, and so on.

You can’t neglect the theory part of the curriculum. Hacking isn’t just about churning out code — you need to understand the first principles in order to write the *right* code. You should be familiar with asymptotic analysis (big-*O* notation), the analysis and properties of basic data structures (lists, trees, hash tables, finite state machines, etc.) and algorithms (sorting, searching, string matching, etc.).

In UNM course terms, that means, basically, CS151, CS251, CS261, CS241, and CS361. CS341 and CS257 are useful to know as well, but not critical.

Finally, you need to be able to write protracted descriptive text in English, including correct spelling, grammar, and usage.

5.3 I heard that only 1337 hax0rz can survive this class.

Not so! Anybody who has gotten this far in CS *can* succeed here. This class *is* a quantum step from CS251 and you have to learn a new batch of skills. But you'll find that you can pick them all up and accomplish things you never thought you could.

5.4 I only speak C++. Can I still take this class?

Sure. But it's on your head. C++ is *deceptively* similar to Java, in that not-really-similar-at-all sort of way. Basically, my position is: if you don't know Java but you think you can pick it up as you go along, go for it. It's all you. But just be warned that we hit the ground running in this class.

5.5 Wait a second. You said something about writing. You mean I have to write *English* in this course? I thought this was a Java programming course!

Yeah, I know that writing is terrifying to many computer scientists. Don't sweat it — it's not that bad. Some of you are already all over this part. The rest of you can learn to do it. We don't require *beautiful* prose, just functional. Even if English is not your first language, I know that you can handle enough to do the kind of documentation we're looking for. And we're here to help advise you on it.

And for the record, no, this *isn't* a Java programming course. It's a software development course. And guess what? If you don't document your software, it's not software. Because it's useless. Nobody will be able to understand it, so nobody will use it, so all of your elegant bits will go on to the Great Bit Bucket In The Sky. And then you'll be unhappy. Not to mention hungry.

5.6 Can I turn stuff in electronically?

Sure. In fact, for your projects you're *required* to turn in your code and other materials electronically. For projects, please archive up your materials (as specified in the assignment) and email them to *both* Dr. Lane and Mr. Anderson.

5.7 What about written reports and documentation? Can I turn those in on paper?

Yes, you're welcome to do so. It does make those easier to read. You still do need to turn in an electronic copy for them as well, though.

5.8 Shiny! I'll ship you my Commodore-64 WordStar 1.3 file right away!

NOOOOOOOO!!!!

Please format your reports, etc. reasonably nicely and send it in either PDF, PostScript, HTML, or plain ASCII. Please do *not* send non-portable document formats. (Source code, of course, should always be in raw ASCII format. Preferably with UNIX newline conventions.)

I DO NOT ACCEPT MS-WORD DOCUMENTS.

Project specs will usually state in what format we expect your submissions. If not otherwise specified, though, you may send us MIME attachments, Zip-compressed archives, tar/gzip archives, or jar files. If you send us an archive (.zip, .tar.gz, .jar), *please* set it up to create a subdirectory for its files, named with your last name and the assignment (e.g., "lane_p1m2" for "Lane's submission of Project 1, Milestone 2"). This prevents your files from clobbering someone else's files. If you end up submitting multiple versions, tack a version number on as well (lane_p1m2_v2).

5.9 Ok, I'll just jar up whatever junk is laying around in my working directory and send it to you then.

What, your mother raised you in a barn? Please try to be a bit tidy. For example, don't send us .class files unless we ask for them. Get rid of temporary files, unnecessary program outputs, and other miscellaneous irrelevant material. *Definitely* don't just package up a raw Subversion working directory — you just end up sending us all the .svn/ administrative directories/files, which are just scurf to us. Check out `svn help export` for how to generate a clean, shippable copy of a Subversion directory. Similarly, arrange your test files and data nicely in subdirectories, isolate your documentation, and so on. Practice good information hygiene. It's the first step toward a healthier, more fulfilling life.

5.10 When is stuff due?

All projects will have defined milestones. The required materials are due at the specified times, *as measured by their arrival in my mailbox, according to my system's clock*. The wise hacker will allow padding time for message transmission.

5.11 What if I *really, really* have to turn something in late?

Ok, life happens. We can't all be on time all of the time. Computers die. Dogs crash. So there's slack built into the system.²

Here's the deal: Say project milestone is due at 2:00 PM on Thurs, Sep 21, 2006.³ By 2:01 PM, it's late. From then 'til 2:00 PM on the 22nd, it's 1 day late. From 2:01 PM on the 22nd 'til 2:00 PM on the 23rd, it's 2 days late. And so on.

Every day late knocks off 20% of the grade value. So if, in that example, you turned your homework in at 11:53 AM on Fri, Sep 22, it would be 1 day late. Suppose the "face grade" of

²Thanks to Dave Ackley for the slack mechanic.

³All times in this example are Mountain time zone.

the milestone is 83/100. Then the “final grade” for the assignment would be $(1 - 1 * 0.2) = 0.8 * 83 = 66.4/100$. Similarly, a 2-day late version of this milestone would get a final grade of $0.6 * 83 = 49.8$. (With a little arithmetic, you’ll see that at 5 days late, you hit 0%. Fortunately, we don’t go into the negatives. At least, not for just being late.)

5.12 20% per day? You call that slack?

No, here’s the slack bit. Everybody gets 3 free “slack days” for the whole semester. One slack day wipes out one late day. So, basically, you get three free late days over the course of the semester.

5.13 When do I tell you where I want to assign the slack days?

You don’t. At the end of the semester, we’ll go back and put in slack days in such a way as to maximize your grade.

5.14 Are leftover slack days worth anything?

Yes, leftover slack days can be redeemed at the end of the semester for an unspecified number of beneficence points.

5.15 What if my iguana dies or my car is struck by lightning in Montana over fall break or I’m kidnapped by aliens or something else that takes more than three days?

Well, we’ll all hope for no emergencies that take more than 3 days. Slack days are *intended* to cover things like colds, broken leg while skiing, etc.⁴ But life does happen. If you have a real, serious emergency, please come talk to me. I’ll be happy to work with you if you have something big happen in life.

5.16 Can I get extra credit on a project?

Sure. Impress us.

I’m happy to give extra credit if you go substantially above and beyond the project spec. What “substantially” means is up to me and the TA.⁵ Sometimes we’ll indicate things that you can do for extra credit in the spec. But I’m happy with creativity. If you come up with a feature that I haven’t thought of, that makes me feel warm and fuzzy. If you want to know whether I’ll consider it “substantial” before you put in the effort, please contact me. I’ll be happy to give you feedback on your plans.

⁴Corollary: don’t try to spend slack days early or unwisely. You may need them for the Martian Death Flu in November.

⁵Hint: Making the background of a widget blue instead of gray is probably not substantial.

5.17 Excellent! My plan to dominate the world by modeling the quantum mechanics of the caffeine molecule in my text editor is underway. I'm sure to get an A+ for that!

Whoa! Hang on there, cowboy! Before you leap into implementing some elaborate feature, be sure that you've got the *required* part of the project under control. No amount of extra credit will make up for tanking on the main specified functionality. Plus, I'll only consider the additional features at all if you've hit at least 80% of the specified requirements. (Yes, that includes documentation. You can't skip documentation in favor of kewl featurze.)

5.18 Can I work with other people in the class on design and stuff?

Yes! Well, sort-of.

Design is a social process. More social than people often give it credit for. It does help to have someone to bounce ideas off of. You're smarter as a group than you are as individuals. You'll do this fully in Project 3, where you'll work closely with a group of your colleagues from the class on all aspects of a project, from design to documentation.

But, for the first two projects, we really want to see how *you* think. So please keep your designs and implementations to yourself. That doesn't mean that you should become a hermit and not talk to anybody else from class. You can certainly discuss the project *specification* with them, but not your *design* or *implementation* that meet that spec. You can also offer a limited amount of debugging help. You're welcome to discuss anything about tools or libraries. You *absolutely must* write all the code yourself and *must not* share your code with anybody else in the class.

We will do a few group design exercises in class. On those, you'll work with your classmates on specific aspects of the project design. Outside of class, though, you should follow the guidelines above and below.

5.19 Ok, that was really fuzzy. I'm confused now. Can you be more specific?

Sure. For the first two projects, you MAY discuss the following with your classmates:

- Project specifications. Examples: "Hey, what does this mean here on Page 7 of the spec?"; "Should the results be sorted forward or backward for output here?"; "Can you find if there's a bound on the runtime for this widget?"; "Can you tell if we were supposed to print the results in Urdu or Serbo-Croatian?"; "Was Prof Lane on magic mushrooms when he wrote this?"; etc.
- Coding style, grammar, spelling, etc. Examples: "Do you think 2 space or 6 space indent is more readable?"; "Why are all your private variables marked with an underscore?"; "Man, camelCase is ugly." "Yeah, but it's a de facto standard, so suck it up."; "Is 'procrastinate' spelled with an 'a' or an 'i'?"; "Could you proof-read my user docs and make sure they're coherent?"; etc.
- Data structures and algorithms, at an abstract level. Examples: "Help me understand how `partition()` works in `QuickSort()`."; "Do you know what data structure supports re-insertion in a queue in $O(\lg(n))$ time?"; "What exactly does a `Map` do?"; etc.

- Anything about tools or libraries. Examples: “Do you know how to get Eclipse to auto-wrap my comment text?”; “How do I build a jar file without using Eclipse?”; “How can I get JUnit to read in a data file only once for an entire test suite?”; “I can’t figure out how to instantiate the HTML parser that he was talking about — do you have a clue?”; “Subversion is complaining `ssh: phoebe.cs.unm.edu: No address associated with nodename` — what the heck does *that* mean?”; “Does anybody know of a library that provides sound support for Java programs?”; “I found this great tool that auto-rewrites all your code backward and in Old High German. How useful!”, etc.
- A *limited* amount of debugging advice. Examples: “I’m missing a semi-colon somewhere and I can’t find it — could you lend me your eyes for a bit?”; “My `iterator()` method is taking forever, but I swear it’s supposed to be $O(1)$ — any idea what’s going on?”; “The results are always coming out sorted by the phase of the moon instead of by the Mayan calendar holidays — can you help me figure out why?”. *Note*: this DOES NOT include “Can you debug line 1 of this method for me? Now can you debug line 2? Now can you debug line 3?...”

You MAY NOT do any of the following:

- Copy code from another person.
- Copy code from the Internet or another source. (If there’s some code that you would *really, really* like to use, please check with us before you do it.)
- Attempt to disassemble, decompile, or otherwise reverse engineer compiled example programs.
- Read someone else’s paper program listings (e.g., discarded in the trash).
- Allow another person to copy your code.
- Leave your class files world readable. (!)
- Copy/plagiarize text from another person, the Internet, a published work, etc.
- Attempt to steal another person’s code (e.g., by cracking, password sniffing, etc.)
- Use an external Java library (i.e., libraries other than those included with the JDK) without documenting it.
- Violate copyright or license agreements on external libraries. *If you use external library code, it is your responsibility to understand and comply with the appropriate copyright and license issues.*
- Violate the University policy on acceptable computer use (included in this document by reference from <http://www.unm.edu/~ubppm/ubppmanual/2500.htm>).

For the third, group project, substitute “your group” for “your” above.

Most importantly, if you have *any* questions about whether some specific collaboration is permissible, *please ask me before you do it.*

5.20 What about cheating?

Ok, this one is serious. No joking here.

You're all adults and this is an upper-division course, so I shouldn't have to say this, but it's worth being up front and explicit so that there are no mistakes.

I do not tolerate cheating.

My general feeling is that being caught cheating should be more painful than not having done the assignment at all. Therefore, I will generally *at least* assign a negative penalty equal to the full value of the assignment if I discover someone cheating on an assignment (though I reserve the right to assess a greater penalty, including outright failure from the entire course and/or referral to the university for discipline). I.e., if an assignment is worth 10% of the final grade, the individual would receive not zero credit for the assignment, but *-10%*. In cases where multiple students were involved, all students aware of the activity will be punished. And, of course, if you're clever enough not to get caught, then you're clever enough not to need to cheat in the first place.

I do not issue "Withdraw-Pass" to students I've found cheating. If you're caught cheating, you either complete the course, or take a "Withdraw-Fail".

Please complete, sign, and return this form to Professor Lane or the TA. (Keep the previous pages for your reference.) Your programs will not be graded until we have received this signed agreement.

CS351 Fall 2006 Student Declaration

Your full name (*print legibly*): _____

Your student ID number: _____

Declaration

I have read and understood the CS351 Syllabus and FAQ for Fall 2006,
and I agree to abide by its contents.

Signature _____ Date _____