

# CS351: Design of Large Programs (Java)

Instructor: Terran Lane

Fall 2005

Tues/Thurs 2:00-3:15

Mitchell Hall 211

## 1 Goals and Subjects

This is a class about how to think about design.

Previous classes (CS152 and CS251 and such) have introduced you to the syntax and semantics of Java and taught you how to write basic programs in this language. If you want to write bigger programs, though – tens of thousands to millions of lines of code – you need to be able to think about design in a more sophisticated way. The goal of this class is to help you learn to think that way, or at least, to make a good start.

The problem is that it's hard to teach thinking directly. I can't download my thought processes directly into your brain. (For which, I'm sure, you're infinitely grateful.) So we have to go about it indirectly. One critical way to learn how to do design is to – well, to *do* it. That is, to learn design, you have to design programs of non-trivial scope. And then you have to *build* them, to make sure that your design actually works (and to find the places where it doesn't and fix them). Therefore, the central component of this class is a set of large design and implementation projects.

The point of making the projects large is not to make you type 'til you have aches in your fingers (in spite of what you may have heard about this class), but to plunge you into a domain where design is critical – where you *must* think about how the program is structured. If you don't think *first*, you'll find yourself coding yourself into corners or producing large amounts of muddled code that achieves nothing. More pragmatically, you'll find yourself spending *much, much* more time on a program than you really need to.

*Remember: Think first. Then code.*

But beyond doing, another important way to learn to think about design is to learn the design tools and tricks that other hackers have developed over the decades. So we'll look at things like idioms, design principles, patterns, tools to support design and development, and techniques like refactoring and testing methodologies. We'll also discuss issues like robustness, security, efficiency, scope, documentation, requirements, and so on.

**A final word to the wise:** The #1 piece of advice I've had about this class is: start early. "Failure to achieve a minimal working program is almost always due to starting the problem too late and trying to complete it in one long burst." (Ackley, 2004).

## 2 Prerequisites

You *must* have completed CS251, CS152, and CS201 (or their equivalents) before entering this class. You *should* have a strong background (251-level) in Java *before* entering this class. If you already have a strong programming background in another language (e.g., C++) it is *possible* to pick up enough Java to succeed in this course, but it is extremely difficult to do so. We will hit the ground running and I expect all students to have a good working knowledge of Java syntax

and semantics the day you walk into the class. If you have questions about your own preparation, please see me individually.

### 3 Textbook

#### Required:

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

This book is widely available — you should be able to get it at the UNM bookstore, Amazon, or your favorite online or local purveyor of fine CS reference works.

#### Optional, but recommended:

*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.

I didn't order any of these in to the UNM bookstore, but all are, I believe, available via special order or through Amazon and other online booksellers and some are available in the UNM libraries.

### 4 Class Resources (Me, the TA, the Web, etc.)

Person	Email	Office	Office Hrs
Terran Lane	terran@cs.unm.edu	FEC325	T,R 3:15 PM–4:00 PM W, 9:00 AM–11:00 AM
Stefano Markidis	markidis@cs.unm.edu	FEC301A	Wed, 2:00 PM – 4:00 PM

(All times given in the Mountain time zone.)

Either of us are also available by appointment (send email).

**Class web page:** The project specifications, example data, homework assignments, solutions, and other notes will be available on the class webpage at: <http://www.cs.unm.edu/~terran/classes/cs351-f05/>

**Mailing list:** [cs351@cs.unm.edu](mailto:cs351@cs.unm.edu)

(Go to <http://www.cs.unm.edu/cgi-bin/mailman/listinfo/cs351> to subscribe.)

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

## 5 Assignments and Grading

This class will be graded on a curve. That said, the ultimate measure of success in software design and engineering is a *functional and correct program*. Thus, there is a floor function built into the curve – completely dysfunctional programs will not receive passing credit, regardless of the relative performance of the rest of the class.

This is a *group work* class. While this prospect inspires fear and loathing in many students, I hope that you'll find that you are more intelligent as a group than you are as individuals. There will be both individual and group work components to your final grade.

Your grade will be based on four components:

**50%** Design projects – two individual<sup>1</sup> and one group project.

**20%** Exams (individual) – one midterm and one final.

**15%** In-class quizzes (individual and group)

**15%** Labs

All group work has a corresponding individual component, preventing freeloading. It will not be possible to pass the course purely on the basis of your group members' performances.

### 5.1 In-class Quizzes

There will be a number of in-class quizzes throughout this class. **I WILL NOT LECTURE ON THE QUIZ MATERIAL IN ADVANCE.** The purpose of these quizzes is to ensure that you are prepared to think about and discuss design issues at a high level, without having to spend valuable class time working on basic syntactic and semantic details. **I EXPECT YOU TO HAVE READ AND THOUGHT ABOUT THE APPROPRIATE MATERIAL IN ADVANCE.**

Each quiz will have an individual component and a group component. The same material will be asked on both sections of the quiz – the group section gives you a “second shot” at the material, with the aid of your peers to help you fill in gaps in your own understanding. Your grade on any quiz will be the average of your individual grade with the group grade.

---

<sup>1</sup>See Section 6 for refinements to this statement.

## 5.2 Group Performance Assessments

At the end of the group project, each member of the group will assess the performance of every other group member. Assessment will be on participation/cooperation, contribution to the progress of the group, and knowledge. This score will contribute to the individual grade.

# 6 Statement of Policy

This section contains a description of the policies for handins, program functionality, documentation, late handins, and academic honesty. I expect all of you to read it and apply it in this class.

## 6.1 Project Handins

Projects must be handed in by email. Paper copy will *not* be accepted. Each project you hand in *must* be in the following format, unless otherwise specified:

- Create a directory of the form `lastname_p#` where `lastname` is your last name and `#` is the number of the project being turned in. So, for example, if I'm handing in project 2, I would make the directory `lane_p2`.
- Copy all code, documentation, README, analyses, graphs, etc. into that directory.
- Create a compressed tarball of that directory. For example,

```
> cd ..  
> tar czf lane_p3.tar.gz lane_p3
```

- Email the tarball to both me *and* the TA as a MIME attachment (often just called an attachment by email programs). Check the documentation for your email program to determine how to do so.

The turn-in time of your program is determined by the timestamp at which it *arrives* in my mailbox (not necessarily the same as when you sent it.)

## 6.2 Functionality and Correctness

Every program longer than 5 lines has bugs. It's a fact of life. Part of your job is to squish as many bugs as you can, and your grade will be based, in part, on how well you do so (i.e., how well your program meets the project spec, passes test cases, etc.)

That said, it is critical that programs basically *work*. In industry, if you roll out a program that doesn't compile, or compiles but crashes immediately, or crashes on the primary required functionality, you'll be canned in short order. In this class, if your program doesn't even work minimally, we will fail it.

## 6.3 Extra Credit

It is possible to attain extra credit on any of the design projects in this class. To do so, you must add functionality, testing, or documentation that goes substantially above and beyond that required by the assignment. The definition of “substantially” is up to the instructor and TA, but if you’d like our opinion before you implement something crazy-hard, please feel free to ask. I’m impressed by initiative and I will happily give you feedback on proposed features. Also, sometimes assignments will make clear certain features that are worth extra credit. These are not the *only* ways to get extra credit, but they’re good suggested ways to do so.

*However* it is always important to complete the customer’s *required* functionality before adding spiffy *optional* functionality. And, critically, optional functionality must never *detract* from required functionality. Thus, we will only consider a feature/test/document for extra credit if the rest of the project has already achieved at least 80% credit.

## 6.4 Documentation and Style

Contrary to popular belief, software design is *not* primarily about code, or even algorithms and data structures. It’s about meeting people’s needs. Software that’s incomprehensible is worse than useless, it’s actively obstructive. Your software must, therefore, be documented both at the user level (user manuals) and internally (code documentation). The code itself must adhere to reasonable style and formatting guidelines.<sup>2</sup> Even when you’re writing code that (you think!) only you will ever use, you should still write good documentation and comments. It’s amazing how strange your code will seem to you after you set it down for 6 months.

I expect you to use correct grammar, punctuation, and spelling in all documentation. For code-level documentation, often phrases and sentence fragments are appropriate, but there’s no excuse for misspellings or incorrect usage within your comments. For user-level documentation, complete, grammatically correct sentences and correctly formed paragraphs are essential.

You will be graded, in part, on usage, spelling, grammar, etc., as well as on thoroughness and comprehensibility of your documentation. Being a computer hacker does not excuse you from natural language skills!

## 6.5 Late Handins and Slack Time

Assignments, including homeworks, reports, assignments, etc. are due at the time specified in the assignment. For assignments due in class, they are due at the *beginning* of class. Project specifications will include a due date and time.

Late programs will be accepted at a penalty. Specifically, for every 24 hours *or fraction thereof* that the program is late, you will be docked 15% of the score of the program. For example, if the program is due at noon on a Friday, and your hand-in arrives at 2:43PM on Saturday, it is two days late and can achieve a maximum score of 70%. If we grade the program and find that it has an “intrinsic” score of 82%, then its final grade will be 57.4% ( $0.82 \cdot 0.7$ ). Note that a program greater than 6 days late ends up at 0% anyway ( $7 \cdot 0.15 = 1.05$ ). Some assignments may also have

---

<sup>2</sup>I don’t have hard and strict requirements on code style, but I expect that you’ve all seen a few examples of reasonable style – indented blocks, descriptive variable and method names, judicious use of whitespace, internal comments where necessary, etc.

an absolute “drop dead” date, after which the assignment *will not* be accepted at all. For example, if an assignment has a drop-dead date of Tuesday, noon, and it arrives in our mailboxes at 1:30PM on Tues, it receives 0%.

I realize, though, that everybody has emergencies (illness, scheduling crisis, family or work issues, etc.) come up and everyone needs a little slack now and then. So you’re allowed 3 late or “slack” days for the whole semester. These are “free” late days in which you are assessed no penalty. So, for example, you can turn in one program two days late and one one day late for no penalty. You do not have to request to use your slack days; at the end of the semester, we will assign them automatically to cover your actual late days in the way most advantageous to you (taking into account number of days late, value of projects, etc.) If you have a *serious* issue that requires longer delay, please speak to me directly. I will deal with these kinds of issues on a case by case basis and *all cases must be documented*.

Unused slack days may be redeemed at the end of the class for an unspecified amount of good will points.

I do *not* give incompletes for sloppiness, disorganization, or inefficiency. Incompletes are allowed only for *serious* and *documented* emergencies. Please contact me directly if you have such a situation.

## 6.6 Academic Dishonesty (and Legitimate Collaborations)

It shouldn’t be necessary to mention this in an advanced class, but the issue has come up in the past so I feel that I should state my policies clearly up front.

Dishonest behaviors, including but not limited to plagiarism, copying of another student’s work/code (or providing your own to another), group consultation on individual projects or work, etc., will not be tolerated. Specific types of cheating include, but are not limited to:

- Copying code from another person.
- Copying code from the Internet or another source.
- Examining the source of any implementations of the Java standard libraries (e.g., the Sun JDK source).
- Attempting to disassemble, decompile, or otherwise reverse engineer compiled example programs.
- Reading someone’s paper program listings (e.g., discarded in the trash).
- Allowing another person to copy your code.
- Leaving your class files world readable. (!)
- Copying/plagiarizing text from another person, the Internet, a published work, etc.
- Attempting to steal another person’s code (e.g., by cracking, password sniffing, etc.)
- Discussing proofs or mathematical methods with another person.

- Failing to document the use of external Java libraries (i.e., libraries other than those included with the JDK).
- Violating copyright on external libraries.
- Violations of the University policy on acceptable computer use (included in this document by reference from <http://www.unm.edu/~ubppm/ubppmanual/2500.htm>).

For the group project, substitute “group” for “person” above.

If you have *any* question of whether a behavior is acceptable, please *ask* me about it *before* you do it.

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). 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”.

That said, this *is* an upper-division course, you’re all adults, and design is a fundamentally much more social activity than people usually give it credit for. So there *are* aspects of the class in which you’re encouraged to talk to one another:

- The group project is, obviously, group work – you’re *expected* to share design, code, work, etc. with the other members of your group.
- You may discuss the *problems* but not the *solutions*. For example, you can discuss the project *specification*, what’s being asked for, etc., but not the *design* of a project, specific data flows, etc.
- You may discuss coding style, grammar, spelling, etc., with other people.
- You may discuss algorithms/data structures in general (e.g., “Help me understand how `partition` works in `QuickSort()`”), but not the role of such things in specific projects.
- You may discuss how the *tools* (e.g., `emacs`, `javac`, `CVS`, etc.) work, including limited amounts of debugging assistance (“What the heck does `javac` mean when it says ‘`Foo.java:192: non-static method foo() cannot be referenced from a static context?`’”)

Exams are, of course, completely individual work.

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

*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 2005 Student Declaration**

Your full name (*print legibly*): \_\_\_\_\_

Your student ID number: \_\_\_\_\_

### **Declaration**

I have read and understood the CS351 Statement of Policy for Fall 2005,  
and I agree to abide by its contents.

Signature \_\_\_\_\_ Date \_\_\_\_\_