

CS351: Design of Large Programs (Java)

Instructor: Terran Lane
Spring, 2005
Mon/Wed 3:00-4:15
TAPY 201

1 Goals and Subjects

Previous classes such as CS152 and CS251 have introduced you to the basic syntax and semantics of the Java programming language, as well as elements of program design. The purpose of CS351 is to attain *mastery* of both the Java language and design and engineering of large programs from an object oriented perspective. Although we will develop programs exclusively in the Java language, we will discuss and examine deeper design issues that span *all* programming languages. Mastery is a difficult thing, and comes only with time and real, hands-on experience. This class will give you some of that experience by giving you the opportunity to design and build a few programs of nontrivial scope. This class really represents a *beginning* on a lifelong road to the kind of true mastery that comes only by grappling with many real-world design issues, solving them yourself, and seeing how other (more experienced) programmers have solved them.

Along the way, we'll examine issues like robustness, security, identifying goals and requirements, project scope, testing methodology, proper use of object orientation, and support tools. And, just in case you think that this is a class purely about "hacking", let me assure you that your theory classes should not be forgotten when you enter this class — a significant part of good design is understanding your tools, including your algorithms. We'll spend some effort analyzing the algorithms at the heart of the code you'll be building here.

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:

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

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

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

Person	Email	Office	Office Hrs
Terran Lane	terran@cs.unm.edu	FEC325	M,W 4:15–5:00 W, 9:00AM–11:00AM
Jason Brown	jbrown@cs.unm.edu	FEC301A	M,W 2:00–2:45; 3:30–6:00 (by appointment) T,R 3:30–5:00

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-s05/>

Mailing list: cs351@cs.unm.edu

(Go to <http://mail.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 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 individual¹ and one group project.

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

15% In-class quizzes (individual and group) – approximately four or five during the semester.

15% Labs

All group work has a corresponding individual components, 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.

¹See Section 6 for refinements to this statement.

5.2 Group Performance Assessments

At the end of each class 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:

1. 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`.
2. Copy all code, documentation, README, analyses, graphs, etc. into that directory.
3. Create a compressed tarball of that directory. E.g.,

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

4. 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.

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).

6.3 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). 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!

6.4 Late Handins

Assignments, including homeworks, reports, assignments, etc. are due at the time specified in the assignment. For assignments due in class (e.g., reports on readings), 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 20% 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 60%. If we grade the program and find that it has an “intrinsic” score of 82%, then its final grade will be 49.2% ($0.82 \cdot 0.6$). Note that a program greater than 5 days late ends up at 0% anyway ($5 \cdot 0.2 = 1.00$). Some assignments may also have an absolute “drop dead” date, after which the assignment *will not* be accepted at all. E.g., 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.) **Be wary of using up your slack days early!** The first project is the *easiest* – if you use up all of your slack days on it, you may find yourself in trouble later. At the end of the semester, unredeemed slack days may be exchanged for an unspecified number of brownie points.

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*. 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.

6.5 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 work, etc., will not be tolerated. Specific types of cheating include, but are not limited to:

1. Copying code from another person (for individual projects) or from another group (for group projects).
2. Copying or referring to code from the Internet or another source.
3. Reading someone’s paper program listings (e.g., discarded in the trash).
4. Allowing another person to copy your code.
5. Leaving your class files world readable. (!)
6. Copying/plagiarizing text from another person, the Internet, a published work, etc.
7. Discussing proofs or mathematical methods with another person.
8. Attempting to steal another person’s code (e.g., by cracking, password sniffing, etc.)
9. Failing to document the use of external Java libraries (i.e., libraries other than those included with the JDK).
10. Disassembling or otherwise reverse engineering someone else’s code base.
11. Violating copyright on external libraries.
12. 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:

- I encourage and expect you all to discuss design (but not code!) with each other. Especially with your group members, but also with other members of the class.
- The reading assignments are intended to be group discussion work. I encourage (and even *require*) you to discuss them among yourselves before coming to class. The corresponding report is also a group assignment, and should include the names of everyone who participated. (Please, if you didn’t participate in developing critique, don’t put your name on it.)
- The final project is, obviously, group work – you’re *expected* to share design, code, work, etc. with the other members of your group.
- 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 specific code by which the algorithm is implemented.
- You may discuss how the *tools* (e.g., `emacs`, `javac`, `CVS`, `eclipse`, 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`’?”)
- You are welcome to talk to the CS departmental tutors for help on debugging, algorithms, specifications, APIs, and libraries.

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 Spring 2005 Student Declaration

Your full name (*print legibly*): _____

Your student ID number: _____

Declaration

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

Signature _____ Date _____