The Term Project

A term project is required of all CSCI E-10b students who have enrolled for **graduate credit**! The completed project is due no later than 9:00 AM on **Monday, May 11 2020**.

A detailed term project *proposal* is due prior to 9 AM on **Monday, April 13** (see section VII of this handout, below). These are firm deadlines. Note that you MUST have your teaching assistant's approval before you begin working on a term project (which is why we are asking you to submit a formal proposal).

I. General Information

Your term project is to be an original program written in *Java* which will run in the *Cloud9* environment.[1] Your project must make intelligent use of original Java ***classes*** (and may, of course, employ ***arrays, Vectors, ArrayLists*** and other data structures), as well as implement an event-driven GUI using Java's "Swing" and AWT classes.

Although we will recommend several possible term project topics, it would be ideal if you tackle a suitable problem that is both important and interesting to you! Designing your own programming problem (or research paper) and solving it is a much more creative experience than writing program solutions for assigned problems. The fact is that year after year, students who have completed introductory courses in computing at Harvard have reported that their term projects were the most valuable part of the course!

[1] Just to repeat myself again: anyone who has registered for graduate credit (e.g., prospective or current ALM in IT candidates) must complete a *Java* project. Ordinary "term papers" may not be substituted for a *Java* project. Undergraduate students may complete a project for some "extra credit." You can also use an integrated development environment for Java, such as *NetBeans*. Just be sure to test your program out in *Cloud9* before you submit it.

II. Choosing a Topic

For some of you, this may be the hardest part of the entire assignment. A number of general topics and specific programming problems which we recommend[2] are listed at the end of this handout. If you don't like any of our suggestions, try to think of areas in which you have experience — there really is not enough time during the last few weeks of this course in which to learn a new subject and to then write a program in that area.

For example, if you do not know how to play checkers, it would be unwise to attempt to write a program that plays an intelligent game of checkers. Furthermore, if you are unfamiliar with the area you choose to work in, you may very well start on a project that is simply impossible! If you are bewildered, sit down with your teaching fellow and "brainstorm" — some of the best projects have had the most unlikely beginnings, I assure you.

One of the most perplexing — and most important — aspects of the topic-choosing stage is to formulate a set of reasonable, limited objectives. We have learned that students can drastically underestimate the difficulty of the problem they choose to solve, and the amount of time necessary to complete the project. If you insist on undertaking a grandiose project, be sure to plan it in stages — i.e., as a basic program, plus some "frills" — so that if you run out of time or patience (or computer memory), you will still have a complete project to hand in. Do not attempt to shoot the moon, and miss. I repeat, your teaching fellow can probably help you pick something of the right size — so I urge you to discuss your ideas as soon as possible.

Another possible source of difficulty is that we will have just finished covering the elementary data structures in the *Java* programming language by the time we expect you to write up your project proposal. If you have had no other programming experience than that provided through this course, then I suggest you read one of the recommended *Java* textbooks — specifically those chapters that deal with *Java's* elementary data structures (arrays, classes, vectors, and files). Without a working knowledge of these topics, it will be difficult for you to determine what sorts of projects are feasible.

[2] Watch out! Some are a lot more difficult than others!

III. Doing the Project

Your goal is, of course, to construct a program or set of programs which runs. But that goal will **not** be attained by sitting down at the computer as soon as you think your idea is clear in your own mind!

You *must* plan your program[s] carefully on paper! Think about your hand-written version before you begin typing it in. One hour of solitary reflection can save you many hours in front of a computer screen, as you haplessly try to debug illogical *Java* code.

Avoid writing long, repetitive programs! A 50-line program that exhaustively checks 50 similar cases is no more interesting than a 5-line program which checks 5 cases. Avoid programs that require typing lots of data — your time using the computer is probably severely limited, and you could better spend it working on the design of the program itself. Once again — see your teaching fellow early and often for help — 15 minutes of our guidance could save hours of your own time.

The size of your finished product is not especially critical. A small, but well-planned, elegant, ingenious and well-documented program is to be preferred to a long, rambling, poorly structured monstrosity (as if you didn't already know). Even a four or five page *Java* program could be an excellent project, **IF** the code is tightly knit, and well thought out.

If you find it absolutely, positively impossible to finish your work by the due date, please remember that a program can be interesting and useful without being a finished product, *so long as it was well thought out to begin with.* Your project is not going to be evaluated simply on the basis of how many hours we think you spent in the terminal room, nor on the sheer size of the program[s].

IV. What To Hand In

Your project should not be regarded as a puzzle for the course staff to figure out. It must be well-commented and annotated, of course. But it takes far more than this to communicate a term project's structure and usefulness, so you must give some thought to the *write-up* that will accompany your program listings, screen captures, and “monitor sessions” — well in advance of the due date.

We expect at least several pages of written explanation (in English prose) and some examples of the program's operation, in addition to the program listings (with their associated comments and other documentation). **Flowcharts** can be very useful. Many of the best write-ups in past years have carefully incorporated all of the aforementioned elements into a single, unified presentation — as opposed to a disorganized document containing haphazard program listings separate from descriptions of what the programs do, etc. Make certain that your write-up does justice to your programs!

If you feel you have done something that is just too dynamic for the printed page — e.g., a particularly clever "graphics" project — arrange for an appointment with your teaching fellow so that you can demonstrate your program. **But don't do this as a substitute for a lucid write-up.** You can also arrange for your TF to run your programs without your being there in person, of course.

V. Lest There Be Any Confusion ...

The publication *Rules Relating to College Studies* explicitly prohibits students from submitting "the same paper in substance in two or more courses without the prior written permission of the instructors involved". Sometimes, joint projects with other courses have been approved, but they must be cleared with Dr. Leitner *in advance.* If your term project idea is based on a published article (or perhaps a programming project in the Reges text), please be certain to acknowledge the reference[s]!

Beware of plagiarism: the CSCI E-10b course staff is familiar with most ***Java*** textbooks and related publications, so don't even think of trying to pass off someone else's work as your own.

As mentioned, completed term projects are due by 9:00 AM on Monday, May 13 2019. There will be positively no exceptions made except in the case of illness, and with prior approval. You have had more than enough experience by now to imagine just how overcrowded the computer facility can be just when you need it most; or how a machine can crash unexpectedly — sometimes destroying an hour or more of work — and preventing you from running your programs just when you had planned. We hope that most of you will attempt to complete your projects early, so that you can enjoy a peaceful and boring end of semester, and leave the competition for those precious hours on the computers to your less far-sighted peers. Although we will not be offering any material incentives for early term projects, we think the advantages are evident.

VI. Criteria for Evaluating the Term Project

Your project will be graded on a scale of 0 to 15 points. As we have been saying, the size of your project is not really important. In fact, you should probably try to prevent your programs from getting much longer than around 7 to 10 printed pages, in total.

Your project will be evaluated in terms of the following criteria, each of which carries equal weight:

(a) Does the program do what it is supposed to do?

In other words, if you make some claims about what your program does, be certain it works the way you say. We occasionally receive projects that have been tested out in a grossly incomplete manner. Though it might appear to the student that his or her program runs correctly, it is quite possible that when we try it out, we will discover some simple case[s] which cause one or more procedures to "bomb out".

In addition to working for “normal” inputs, you will be graded on how well your program is *idiot-proofed* and how *user-friendly* your program appears to be. While it is unreasonable to expect you to completely foolproof a term project, you should nonetheless include tests that ensure the user will be prevented from typing in information that causes the program to malfunction. If, for example, there is a part of your program that expects the user to input a value between 1 and 5, then your program should test whether, in fact, the person using the program did type in a proper value — and if [s]he did not, to report this condition to the user, and take appropriate action!

Try to do exhaustive testing of the program as part of your term project. Any features not tested will not be counted as part of the project. Needless to say, your program should not contain any syntax errors, nor execution errors with normal inputs.

(b) Level of Difficulty

A program is interesting if it solves a problem that is difficult or impractical to solve by hand, or if it provides new insight into an old problem. Some of you will be tackling rather difficult problems — those problems which have no obvious solution. At the other extreme, some students may choose to try and solve a rather simple problem for which there are approaches known beforehand.

Obviously, an individual who works on a program that plays intelligent chess for just a few moves has tackled a much more complicated problem than someone who writes a program to play a complete game of intelligent **Nim** (or some simple parlor game). Incidentally, to write an intelligent chess-playing program is all but impossible.

(c) Presentation

Your write-up **is** of importance to us! Your presentation should include a clear description of the project and how it works. This can be, to some extent, an updated and more specific version of your proposal (which we will describe shortly). Also included under this "presentation" heading are such factors as the clarity with which you documented your programs (don't bother writing foolish and unnecessary comments next to every *Java* statement). Remember, *flowcharts* and UML diagrams can be of great help in your presentation.

Your write-up should be type-written (or very neatly hand-printed). They are generally anywhere from 5 to 15 pages in length. But be reminded once again: We are not asking for quantity, so keep the "bull" to a minimum.

(d) Originality of Ideas

You could try doing something completely original, such as a simulation program of some real-world or imaginary environment that you have defined. Or you might choose to program a well-known game such as tic-tac-toe, which has been done many times before. Of course, should you choose a problem that may appear "unoriginal", it is still conceivable that your algorithm[s] provide a highly innovative solution to the problem.

Criteria such as imagination, creativity and originality are, of course, difficult to define. You will know your project has these characteristics if, in response to your proposal, your TF pounds his or her forehead and mumbles, "Why didn't **I** think of that?" Even a very creative and original project gets stale after a while, so think twice before you produce the same project that a friend of yours produced last year!

(e) Programming Sophistication

Yes, my friends, you will be graded also on programming *style.* Please try to obey the rules of "top down" and structured programming that we have emphasized in this course. In particular, this involves modularizing your project by using methods. It really is unreasonable for any individual *Java* program to be much longer than 20 or so lines. Break up your programs into smaller routines. Form the habit of writing methods to do small tasks so that your program will be uncluttered, highly readable and easily modifiable.

A well-structured main program might simply consist of four or five statements — each of which is a "call" upon a *Java* method (which, in turn, calls upon other methods) to accomplish various tasks. Particularly cunning data structure representation of information is something to watch out for. The many examples we will have done in lecture and in section should provide ample motivation. In addition, you have already been handed out (or you'll shortly be handed out) a document that summarizes the most important rules to obey in writing a stylish *Java* program. Be sure also to pay careful attention to the Graphical User Interface (GUI) you produce; it should be clear, uncluttered and easy to use!

VII. Write a Project PROPOSAL

In order to make certain that you start thinking about a project right away, we are asking each of you to produce a write-up of several pages stating a firm proposal — together with some clear plans — for your term project. The proposal is a short, typewritten paper which describes:

- ❐ What your project is supposed to do.
- ❐ Why anyone in their right mind would want to write a program to do what yours is going to do.
- ❐ The form of the input and output, perhaps with a sample dialogue.
- ❐ The main algorithms and data structures, perhaps partially specified in flowchart form.
- ❐ A list of the main subprograms you will probably need, including a brief (1 sentence) description of what each one does.

Note that this proposal **will** be graded, commented, and returned to you like other homework assignments, so PLEASE give it some thought. You can, of course, propose more than one project, and have your TA choose the one [s]he feels is best.

Your proposal will be returned to you in one week (or less) after you turn it in. If your proposal is unacceptable — or has serious flaws in it — you should speak to one of the staff about your ideas before you begin working with *Java*. If your proposal is accepted right away, you will have roughly 4 weeks to complete the project, which should be ample time. Note, however, that you will also be working on at least one more regular homework assignment during this time to get you more conversant with *Java*!

If you hand in a poorly written or otherwise unacceptable project proposal, your TA will notify you immediately by sending "computer mail" on the mainframe system. It is your responsibility to check for computer mail and get in touch with your TA, if necessary. It is our goal to be sure that every student knows what his or term project will be (if any) by the end of April.

We advise you to reread this handout carefully, and in its entirety. To assist you in your search for a suitable term project, you might find it helpful to look at some of the optional texts for this course (on reserve in Sever Hall's Grossman Library). Computer magazines are also a potential source of project themes, but remember that a project that interests you and involves a field you know something about is likely to turn out much better than a project you chose just to complete a requirement. Consider your field of study (if applicable), other courses, your job, hobbies, sports, extracurriculars, etc., when searching for a topic.

The programming project for this course is similar in many respects to term papers assigned in other Harvard courses. The qualities of a superior piece of work are nearly the same for both types of assignment: proposing and solving an interesting problem in a manner that shows imagination, creativity and mastery of the material. Most of you know what these qualities are for an ordinary term paper; we hope to have successfully elucidated how to apply them to a computer programming assignment. Make your project proposal as detailed as possible, and if your ideas are derived from any "outside" sources, remember to properly give credit to those sources (we again warn you that *plagiarism* really is a serious matter around here). It is very easy for us to use Internet search engines to uncover verbatim copying or even to track down rephrased ideas that have been derived from someone else's work.

Remember, your term project proposal is due prior to 9 AM on Monday morning, April 15. Feel free to turn it in sooner than that (perhaps by emailing it to your teaching fellow).

THE EARLIER YOU SUBMIT YOUR PROPOSAL AND GET IT APPROVED, THE LONGER YOU WILL HAVE TO WORK ON YOUR PROJECT!

VIII. Some Past Projects, and Other Possible Topics

Most of the topics listed below are far too broad to be a single project.

A. Games

Games of almost every conceivable variety, attacked from almost every possible angle. You can set up your program to play against a human opponent, or to aid a human playing against another human, or to play against itself, or to learn from its own (or its opponent's) mistakes. etc. Here are some of the games that have been used ... you can extend this list indefinitely:

Checkers; dominoes; tic-tac-toe in 2 or 3 dimensions; Hangperson; Mastermind; backgammon; roulette, Risk; baseball, football, tennis; poker; bridge, etc. A few years ago an e10b project simulated the handheld electronic Merlin toy!

B. The Arts and Letters

The trick here is to find an area in which there are some more-or-less formal rules. The following should provide you with some ideas:

Music	A program that composes music in some identifiable style, say blues, 12-tone, Baroque, etc.
	A program that completes harmonic progressions.
Languages	A program that can write grammatically correct sentences in some language (a subset of a natural language, perhaps).
	A program which conjugates verbs in French, Latin, etc.
	A program that parses or diagrams simple sentences in some language.
	A program that performs intelligent "syllabification" or grammatical

	transformations with some natural language.
Poetry, etc.	A program that composes verse according to some fixed scheme (or rhyme or meter).
	Haiku
	A program that scans verse, searching for Latin hexameters (for instance).
	A program that composes plots for mysteries from the familiar elements; or write stereotyped "fairy tales".

C. Scientific Algorithms

There are lots of ideas available from various sources. We mention just a few:

Mathematical	Formal manipulation of polynomials, power series, or other symbolic expressions; integration by parts; plotting; finding maxima; solving systems of linear (or other) equations.
Logic	Proving theorems in the propositional calculus.
Chemistry & Physics	Predict chemical reactions or subatomic particle interactions.
Computer Science	Write a very simple compiler or interpreter; compare sorting algorithms; minimize finite automata, etc. One of the teaching fellows is likely to be particularly helpful in this category.

D. Simulations

Simulation is an attempt to predict events in the real world by creating a computer program to mimic or model them. The outcome of simulation experiments are used by decision makers as plausible predictions. You can simulate almost anything under the sun — or the sun itself, for that matter.

Simulations tend to be the most popular kind of term project because they lend themselves to most any topic of interest. Here are just a few examples:

Population	Make some simple assumptions about how some species prey one each other, and watch their numbers grow and shrink (perhaps using Java "graphics"). The game of **Life** is a good example and we've received a few terrific implementations of this in recent years.
Ecology	The interactions of food chains, pesticides, burning of fossil fuels, etc.
Climate	Test different strategies for spending a fixed amount of money on reservoirs, food warehouses, flood control systems, etc., so as to keep us alive for the next 100 years (or until the melting Arctic ice floods us out).
Economic Systems	The stock (or options, commodities, etc.) market. The inventory of a small store or a large chain. Waiting lines of customers for a particular service or facility. The inputs and outputs to/from any sector of our (or some other) economy.
Energy	How long before we completely deplete our current reserves of certain non-renewable resources?
Biology/Physiology	Test out some simple model of the nervous system of an organism (old issues of *Scientific American* probably have some examples). Test a real or imaginary urban or inter-urban mass transit system for efficiency. Lay out one-way streets for Harvard Square to minimize rush-hour congestion; simulate how construction repairs mess up traffic and pedestrian patterns in affected areas.

E. The Unclassifiables

A program to give information on bus, subway or airline routes.

Biorhythms; astrology; the occult.

Gambling — predict point spreads of football games, or the order of finish at Suffolk Downs.

Election results — how *do* the networks do their predicting?

Teaching machines — take some small subject which you know well, and see if you can write an interesting instruction program. Try to find a subject that does not need long sentences for input or output, such as verb conjugation in a foreign language.

Proportional representation — the City of Cambridge and Harvard University use the complicated *Hare System* for certain elections. Write a program to do the counting, and try it on some simple data to see if it really does result in proportional representation.

Of course, you may use one of the many lecture or section demonstration programs to generate ideas of your own.

In the course's lecture *Cloud9* directory (available by clicking on the "Java Lecture Examples" page on our course website), you will find a folder named **ProjectSamples**. Inside that folder are individual folders containing executable Java .class files for a few cool projects that were completed in recent years. Have a look!

Best of Luck — and don't forget about that term project proposal!