

## Final Project

### 1 Overview

During the final weeks of CS230, you will have the opportunity to work on an individualized final programming project in an application area of interest to you. **Your completed final program is due at the end of Final Exam period, on Tuesday, December 21, at 4:30pm.**

There are five intermediate stages involving work that you submit, which will also be graded. You will receive feedback from the Instructors on each intermediate stage. This handout provides overall guidelines for the project and the intermediate stages, and lists some possible problem areas that you may consider. You are allowed to work in pairs on this project, although the final program must be more extensive in this case, and you will need to document the contributions of each partner. Additional guidelines regarding joint projects are given below.

### 2 General Guidelines

For this programming project, you will design and implement an entire program largely from scratch. You can receive advice and help from the Instructors on your design, implementation and testing, but you should complete the programming on your own as much as possible. You are allowed to use any of the standard classes that are in the Java SDK as well as any classes that have been created or provided in CS230 this semester (for example, the implementations of various enumerations and collection classes that you use in your program). It is expected that all other parts of the program will consist of code that is developed by you from scratch. Unless you are given explicit permission from one of the Instructors, it is not acceptable to use code from other resources, such as books, the Internet, previous semesters of CS230, and so on. Using code from these other sources without prior approval from the Instructors will be considered a violation of the Honor Code.

As part of your project, your program should involve at least two non-trivial Abstract Data Types (ADTs) that you design and implement on your own. You should discuss your ideas about ADTs with an Instructor in the early phases of your project. You are additionally encouraged to use ADTs that we studied in class: linked lists, binary trees (or a more general kind of tree), stacks, queues, priority queues, sets, bags, and tables. Graphs are a common data structure we have not covered that are useful in many projects. If your project involves the manipulation of a large amount of data, you may want to store this data in a text file. Drawing upon your CS111 background, you may also use graphics. You are expected to write a fairly extensive program that is larger than the programs that you completed in assignments during the semester.

Every project is required to have some sort of Graphical User Interface (GUI). It is also required that your project be accessible as an Java applet, so that it can be embedded in a web page and added to an on-line museum of student projects. You will learn more about GUIs and applets in a few weeks.

### 3 Project Phases

There are six phases to the final project, each of which requires you to either submit text and/or code or talk about your project.

#### **Project Phase 1: Project Proposal (Due Fri. November 5) [5 Points]**

For the first phase of the project, you should hand in a proposal of the final project you plan to undertake. This initial description should be written in English. The proposal should include the following:

1. A summary of the problem that your program will solve or application that your program will implement.
2. A description of what you expect will be the overall input and output of the program. If there is a graphical user interface, draw a picture of what you expect the interface to look like.
3. A description of what ADTs will be used and what information they will store. Include a brief justification for each of your choices.

If you are unsure of what to include in your initial description, please ask one of the Instructors. We strongly encourage you to discuss your choice of project topic and data structures with an Instructor before submitting this first phase of the project.

#### **Project Phase 2: Program Outline (Due Tue. November 23) [5 Points]**

For the second phase of the project, you should hand in a detailed description of the structure of your program:

1. A list of the classes that you expect to use in your project. Some of these classes should capture the basic objects that exist in the problem. There may also be classes that embody the graphical user interface, or the `main()` method. This list should include the classes of all ADTs that you plan to use in your project, including those provided in CS230 or by the Java SDK. As you proceed with your program development, you may discover other classes that would be useful to define for your application.
2. For each class that you are defining on your own, you should include a description of the purpose of the class and a preliminary contract specifying any public constructor, instance, and class methods and any public instance and class variables.
3. A high-level summary of how you will use the classes in your program to achieve the goal of your project.

#### **Project Phase 3: Detailed Program Skeleton (Due Fri. December 3) [10 Points]**

In this third phase, you will submit a detailed skeleton of the entire program, with some completed code. The amount of completed code should be at least 25% of the anticipated final size of the program. You should have skeletons for all the classes you are defining in your program. Each class skeleton should contain headers for each method and a stub implementation for each method (which might not do anything “real”). At least some of your method bodies should be fleshed out,

though they may be incomplete or contain logical bugs. However, your skeleton should not contain any syntactic bugs – each of your classes should compile without error.

If you have made any decisions that deviate from your Phase 1 or Phase 2 plans, you should briefly explain your decisions.

#### **Project Phase 4: Informal Presentations (In Lab, Tuesday, December 7) [5 Points]**

During the final Lab class, each of you will share your project ideas with other students in your Lab. It is expected that you will not have a completed program at this point, but you can describe your general problem or application, choice of data structures, and user interface. You might also comment on what aspects of the project were especially interesting or challenging. These presentations should last 5-10 minutes.

#### **Project Phase 5: Code Review (Tuesday, December 14) [25 Points]**

In this penultimate phase, you should submit a draft of your final program. By this time, most of the functionality of your program should be working. The purpose of this review is for the instructors to have time to give you feedback on your project that you can incorporate into the final version. For this part, you should submit both a hardcopy and softcopy of all your program files. (Please submit the softcopy to the `review` directory in the `drop` folder.)

#### **Project Phase 6: Final Program and Demonstration (Due Tuesday, December 21 by 4:30pm) [50 Points]**

In this final phase, you should submit your final, complete, working program. Submission consists of completing the following three parts by 4:30pm on Tuesday, December 21.

1. *Hardcopy* Submit a hardcopy of all of the code files in your project, as well as a description of how to run your program.
2. *Softcopy* Submit a softcopy of the entire program directory to the `project` subdirectory of your individual drop directory on puma (be sure to verify that the directory is complete after dropping it off). For team projects, please indicate whose `project` directory holds the final version of the project.
3. *Demonstration* You must schedule a meeting with Lyn and Stella to demonstrate your final program and talk about your experience with its development.

Your final program should encompass principles of good program design, such as the effective use of classes and methods, informative names for variables and methods, and code that is efficient and concise. With regard to documentation, you should provide at least one comment at the beginning of every method, describing what the method does, and at least one comment at the top of every class file, briefly describing the class. Additionally, your final directory should contain a README file that (1) gives the name of the project and the name(s) of the implementer(s); (2) describes what the program does; (3) explains how to run the program; and (4) describes any known bugs in the program.

## 4 Additional Guidelines for Joint Projects

It is possible for two students to work together on a single programming project. In this case, the size and complexity of the planned project should be more substantial than what is expected of a typical single-student project, requiring about twice the amount of programming effort as the single project. The project ideas listed below are all examples of programs that have been completed by single students in the past. These ideas can also form the basis for joint projects, but you should work closely with Lyn and Stella during the early phases of the project, in order to make sure that the plan for the project is substantial enough to support joint work. In addition, the Phase 2 and Phase 3 submissions should include a plan of work that indicates how the programming effort will be distributed between the partners. Some programming for a joint project can be done together, but there should be parts that each individual student is largely responsible for developing. As the program is written, the efforts of each student should be documented in the comments to the code (i.e. each class file should include a comment stating which student was primarily responsible for its implementation). This kind of additional planning and documentation is common in real-world software development projects that are implemented by a team of programmers.

## 5 Project Ideas

The following are some initial project ideas. You are welcome to choose a project outside of this list, but should discuss your idea with an Instructor.

### Games

Build a program that allows a user to play a game. Examples of games include card games (e.g., Poker, Hearts, I Doubt It), and board games (e.g., Mancala, Checkers, Scrabble, Parcheesi). In multi-player games, you may want to design a computer player in addition to allowing multiple players to play the game.

### Text Processing

Write a program that processes text, such as a text formatter, spell checker, or automatic indexer. A program like a spell checker will require a dictionary of known words, which can be provided for you.

### Web Page Processing

Write a program that downloads web pages and extracts information from the web pages. For example, you might download and save all New York Time articles from the current day that contain specified keywords.

### Voting Programs

The Wellesley College administration needs to update the program for counting ballots from Academic Council elections. One choice is to reimplement the program (now written in Basic) in Java. Another choice is to design and implement (part of) an electronic voting system.

## **Natural Language Parser**

Write a program that can accept a simple English sentence as input, and parse the sentence into its basic constituents. Such a program would use a set of simple rules of English grammar, combined with a simple dictionary of words that your program can recognize, to build a system that recognizes the parts of speech of the words in the input sentence and constructs a representation of the basic components (noun phrase, verb phrase, etc).

## **Database Problems**

Write a program that allows a user to work with a database of stored information about a particular topic. The user should be able to add new entries to the database, modify entries, and retrieve information. The user could interact with the program through a graphical user interface. For example, you could manipulate a database of Wellesley students with their photographs. If you choose a database problem, you will need a source of data that you can get easily from elsewhere; do *not* plan to enter data on your own!

## **Mini Expert System**

Build a program that embodies expert knowledge about a domain, and can use this knowledge to make inferences from data provided by the user. For example, a simple medical expert could collect data about a patient's health and try to diagnose the cause of the patient's illness. An expert system could also identify a class of objects based on observed properties, or make decisions about actions to perform in response to a current situation. Many expert systems build their knowledge into simple rules.