**Phase 1**

 收藏此页

**Introduction**

If you have not done so already, please be sure to carefully read:

* [overview](https://edge.edx.org/courses/course-v1:UBC+CPSC210+all/jump_to_id/0e029c40da154109bb973e42d5ae13c7)
* [description of the project structure](https://edge.edx.org/courses/course-v1:UBC+CPSC210+all/jump_to_id/489b22fb24ba40f0abdc3d6e8621254c)

**Phase 1 of the term project is worth 25% of your overall project grade.**

**Academic Honesty**

All the code that you submit at each phase of the term project must be

* code that you have written yourself since the start of this term (you *must not* make use of *any* code that you have written in the past)
* code found in or based on any of the projects that we present to you in lecture during the term, provided you insert a comment into your code to attribute the source

Please ensure that you do not provide other students with access to your code.  By doing so, you risk enabling academic misconduct for which you will be held accountable. Typically, the same grade penalty is applied to the student who enabled access to their code, as to the student who used the code.

**Beware**: a number of students have violated this policy in the past and have paid the price.  If you find the project challenging, ask for help from a course instructor or from one of the TAs *assigned to the course this term*.  Do not be tempted to copy code from someone else and do not, under any circumstances, allow another student access to your code in any way.  We have a tool that identifies unusual similarities in code - it works really well and we *will* be using it.

**General Requirements**

You should commit and push your code to GitHub after completion of each task, if not more frequently. By pushing your code to GitHub you are effectively making a backup of your work which you'll be able to retrieve if anything happens to the code on your laptop.

All code must be appropriately documented:

* each class should have a class-level comment (that appears just before the class declaration) to describe the information that the class represents
* every method must be documented with requires, modifies and effects clauses, as appropriate (note that simple setter and getter methods, and test methods, do not have to be documented)

You code must adhere to our style guidelines.  Note that for the term project, the maximum method length has been increased from 20 to 25 lines.

*Note*: full marks for code coverage can be achieved only if there is at least one branch (i.e. if/else) or loop in the model code.

**Main Tasks**

Start by cloning the repository project\_xxx from GitHub where xxx is your CS ID.

**Task 1**

Your first task is decide what you'd like to create for your term project.  We recommend that you give this some thought as you'll be working on the project that you select for the rest of the term.  This may seem daunting but we'll be giving you guidelines along the way and your TA will be there to offer guidance. There are some broad criteria which you must adhere to in the design of your personal project:

* all code must be written in Java
* you must design a Java desktop application (no Android applications, for example)
* ultimately, your application must have a graphical user interface but you must start by designing a console based application (as described below)

If you are stuck for ideas for your project, please try to discuss ideas with your TA first. If time goes by and you are still stuck, consider:

* A to-do list application. Keep track of things to do, cross things off.  Possible features include different views showing all items, or just completed items, or only outstanding items, or lists sorted in different ways.
* A recipe manager application. Keep track of your recipes. Possible features include different categories of recipes; listing all recipes that use a particular ingredient; visualizations of your recipe collection.
* A fantasy sports team application. Enter statistics for players/games, and determine how your fantasy team is doing. Possible features include multiple teams and visualizations for how teams are doing (a leader board, for instance)

Now that we've thrown those out there, we recommend that you use one of these suggested projects as a last resort. Try to come up with your own idea for a term project, so that you have something unique to talk about at a future job interview. As this is a personal project, once the course is over, you'll be able to publish it on GitHub.com, so that others can view your work. For that reason, we strongly encourage you to think carefully about the design of your code - remember it's written for people to read (that includes a future employer!), as well as to be run on a computer.

**Task 2**

Use the Project view in IntelliJ to browse your project and note that it has a file named README.md.  Your first task is to write one or two paragraphs at the start of this file that describe the project that you propose to design this term.  Your proposal should provide answers to the following questions:

* What will the application do?
* Who will use it?
* Why is this project of interest to you?

The .md extension on the file name tells us that this is a *markdown* file.  We're not expecting you to become markdown experts but you should at least learn how to:

* add a title and subtitle
* apply bold or italic fonts
* insert a bulleted list

You must make use of at least two of these three elements in your proposal. Note that the README file that we provide to you includes examples of each of these elements. A guide to these and other elements of basic markdown syntax can be found at <https://www.markdownguide.org/basic-syntax/> - be sure to focus on the markdown elements and the corresponding rendered output but ignore the HTML. The use of markdown is prevalent and it is therefore well worth learning at least the basics.  The course web page, for example, is almost exclusively written in markdown.  Markdown is also used to format the feedback you receive from AutoTest.  IntelliJ has a markdown renderer built in, so you will be able to preview your README page.

Note that the README page is displayed by default on your repository's homepage on GitHub - this is the first thing someone will see when they look at your repository, so it's worth putting some effort into making it look good.

When you are done, commit and push your code to GitHub.  Remember to use a commit comment that accurately describes the changes that you have made in the current commit.

**Task 3**

During each phase of the term project, you will write *user stories* that describe how someone can use your application to produce a specific outcome.  You will then design the code necessary to make that user story a reality in your application.  Here are some examples of user stories that you might write for this first phase of the term project:

In the context of a to-do application:

* As a user, I want to be able to add a task to my to-do list
* As a user, I want to be able to view the list of tasks on my to-do list
* As a user, I want to be able to mark a task as complete on my to-do list
* As a user, I want to be able to delete a task from my to-do-list
* As a user, I want to be able to see the number of incomplete and number of completed tasks on my to-do list

In the context of a recipe manager application:

* As a user, I want to be able to add a recipe to my collection
* As a user, I want to be able to view a list of the titles of the recipes in my collection
* As a user, I want to be able to select a recipe in my collection and view the recipe in detail
* As a user, I want to be able to select a recipe in my collection and rate it on a scale of one to five stars

In the context of a fantasy sports team application:

* As a user, I want to be able to create a new team and add it to a list of teams
* As a user, I want to be able to select a team and add a new player to the team
* As a user, I want to be able to select a team and view a list of the players on that team
* As a user, I want to be able to select a player on a team and add a new statistic for that player, for example, the number of goals scored in a game
* As a user, I want to be able to select a team and list all the players on that team

Note that in each of these sample applications, there is at least one user story that involves adding multiple Xs to a Y (e.g. adding an item to a to-do list) where X and Y are classes that you must design yourself (so Y cannot be of type ArrayList<E> for example). By *multiple*, we mean that it will not be sufficient to always add a fixed X to a Y but this must be a dynamic process (i.e. in most cases this will mean there is no fixed size of Xs but this is not a requirement). You must include at least one such user story in this phase of the application along with three others of your choosing. Note that you will be allowed to modify and expand on your user stories as the term progresses and as the scope of your term project evolves.  So you should view this as a first iteration.

Add a "User Stories" subtitle at the end of your README.md document and list your user stories.  When you are done, commit and push your code to GitHub.

**Task 4**

Make your user stories a reality!

Before you get started on this task, carefully review the [Teller](https://github.students.cs.ubc.ca/CPSC210/TellerApp) application that we'll be studying in more detail later in the term. Notice that the Account class represents an account in our banking domain and is found in the model package, with the corresponding test class in the tests folder.  The TellerApp class provides the console based user interface and is found in the ui package.  The application nicely demonstrates the division between code that comprises the model and code that comprises the user interface. You need to maintain this same separation in the code that you write.  Also notice that the Main class has no fields and that the Main.main method contains a single statement that creates an instance of the TellerApp class.  Your Main class and main method should be similar, comprising no more than a few lines of code. Please keep in mind that the Teller application does not contain all the features necessary to obtain full marks on this phase of the term project - it serves only to illustrate how you can separate code that models your domain from code that implements the user interface.

Start by doing a domain analysis for your application. What information is changing and what is constant? Identify the different types of data that you will need to represent the information in your domain.  So, for example, the to-do application will need a Task class to represent a single task and a ToDoList class to represent the tasks that have been added to the to-do list. You should then consider the methods that each of these classes must have if you are to be able to provide the functionality described in your user stories.  So, if the user must be able to add a task to their to-do list, the ToDoList class will need an addTask method. The classes that you identify at this stage in your design must be added to your model package with the corresponding tests in the test folder.

The classes MyModel and MyModelTest were provided with the starter code.  Either delete these files or rename them to something more appropriate for your particular application.

Note there must be no System.out.println statements in the code that models your domain and no code that reads user input from the keyboard.  These statements are used to present/gather information to/from the user and are therefore part of the user interface which you'll write in the next task.

When you are done, commit and push your code to GitHub.

**Task 5**

The next step is to add the user interface.  At this stage you *must* design a console application.  *Do not* attempt to design a graphical user interface.  All the code for your user interface must go in the ui package.  As user interface code is more difficult to test, in general, note that you are not required to design tests for your ui code on this term project. However, you must ensure that all your methods are appropriately documented.

When you are done, commit and push your code to GitHub.

**Grading**

When you are ready, commit and push your code to GitHub.  Request feedback from AutoTest by making the following comment on your commit:

@autobot #project

As you are all working on individual projects, AutoTest will obviously not give feedback on the correctness of your code. However, it will check that your code adheres to the CPSC 210 coding style and will measure the degree to which your tests cover your code. The code coverage score that you receive from AutoTest will be used, *in part*, to determine your overall grade for this phase of the term project as follows:

* 20 pts - (code coverage score \* 20) - so 100% code coverage yields a score of 20, whereas 0% code coverage yields a score of 0
* 10 pts - clear and correct documentation has been provided
* 70 pts - realization of four user stories relevant to the proposed application (roughly equally weighted); to obtain any of these marks at least one of the user stories that has been realized must be of the form "add multiple Xs to a Y" as described above.

Note:

* any attempt to "fake" the code coverage score (e.g. calling a method but not using appropriate assertEquals, assertTrue and/or assertFalse statements to verify the expected behaviour) will result in an automatic zero for code coverage
* you must get at least 50% on this phase of the term project.  If you don't achieve that grade, you will have to continue to work on this phase of the term project and request a regrade at your next lab for a maximum score of 50% on this phase of the term project
* we recommend you check the "General Requirements" section above to make sure that they have been met

During your demo for this phase of the term project, you will be asked to:

* checkout a copy of the code that you pushed to GitHub *prior to the deadline*
* show your TA the code coverage score received from AutoTest
* run your tests and demonstrate that they all pass
* run your code to demonstrate that your user stories have been realized