EECS 1720
ASSIGNMENT 1 (and precursor for assignment 2)

The overarching goal of assignments 1 & 2 in this course is to: (1) design and construct a supporting set of classes that model necessary components of a simple interactive application/game, and (2) design/integrate this into a functional graphical user interface (GUI) that can be used to configure and launch/demonstrate various features of the application/game.

The application itself can be one of two types:

1. a content management/creation-based application (that allows for "records" or "content items" to be generated, saved and recalled)
   OR

2. a simple "dedicated card/board game" that works with a specific set of "cards" and "players" in order to monitor the progress of a player working toward the game goals.

For example:

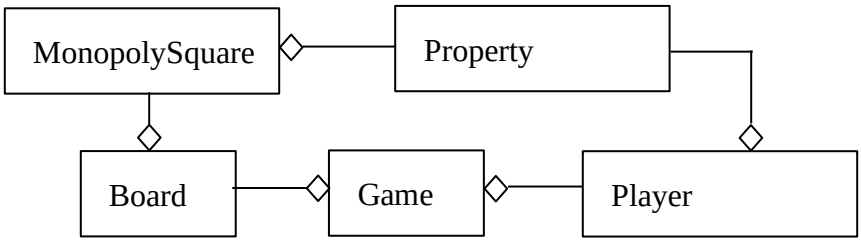
1. If opting for a content/record creation system, you might choose to design and build a journaling/scrapbooking application (which can collate notes and/or graphical content), a to-do/memo or shopping list tool, or a booking/ordering based system (like a flight booking/ scheduling or interactive calendar) style application, or an application that can pull data records from a source, and formats/organizes that data into one or more browsable components.

   [**NOTE**: *you are restricted from re-creating a standard painting/drawing application such as that explored in EECS1710.. if you take this route, however, making an application that can spawn geometric patterns from a set of initial parameters is ok, but the application would have to have the ability to save/record those parameters & content as a record, and ultimately re-load the content*]

2. If opting for the dedicated card/board game, you are required to model the various sorts of cards that form that game (using several ~3+ classes), and other aspects that may be key to playing (such as certain draw/discard piles, player inventories, or if a board game, the types of squares on the playing board).

   The classes should have one or more components embedded that enable the state of the entity to be rendered visually (e.g. like the front/back of a type of card). It is not important that you render all possible versions of a card, but rather, have at least 1-2 types implemented to demonstrate this capability. Likewise, perhaps the rendering of a user's "Hand" (set of deal/held cards) which involves multiple cards.

For instance, imagine you want to model aspects of the classic board game "Monopoly". Entities you would need to model with classes might include: MonopolySquare/Property (a class that models a typical monopoly square) – which is usually associated with a "property", which have attributes like street name, mortgage cost, rent pricing (with/without houses/motels), etc. There might be also be a Player & Game class that models aspects of an individual (like their current bank balance, how many properties they hold, etc.). Alternatively, there might be a class that models the cards associated with Monopoly (i.e. Community Chest or Chance – which have other attributes associated).



Alternatively, there is a dedicated-card game equivalent called "Monopoly Deal" http://monopolydealrules.com/index.php?page=play

In this version of the game, there are parallels to the original board game, but the goals and entities are different.. everything is a card, but you have money cards, property cards, and action cards (see link above for more information). The idea is to first build some classes to support these entities (give them the appropriate fields/attributes), and think about the sorts of behaviours (constructors and methods) you want to support for a given card – e.g. you might have a class for an ActionCard, with appropriate getters to access its information, or a classes that support the piles that support game play (the Deck/Discard pile), a Player (that holds money cards and current properties).

The goal is to think about relevant classes and how you would model the parts of the game, and to build those classes (or a partial set).

Another option might be to build an RPG card-based game (either of your own design or to try and model/create one that already exists). In such games, there are character cards to model, weapons/items cards, action cards and enemy/monster cards. Again, the idea is to try and design classes to support these, and any elements of gameplay that might be needed (e.g. Discard piles, Player, Deck, Game, etc).

You can find a list of common (notable) games here (to get ideas).
https://en.wikipedia.org/wiki/List_of_dedicated_deck_card_games

[**NOTE**: you are restricted from modeling and creating any games that use a standard deck of cards.. i.e. typical face cards (King,Queen,Jack) and Ace, 2-10. So no poker, blackjack, etc. The cards have to be "dedicated" to the specific game. For example, Scopa (the Italian card game) is ok, but preferred would be things like an RPG role-playing game (e.g.

*** it will be more straight-forward to work with a card-based game than a board game, and ultimately, elements of gameplay may also be simpler. Try to look for card-game versions of board games (if interested in a board game).*

In **Assignment 1**: you will focus on designing the concept/classes to support key entities of the app/game (at least 3 different classes: 2+ for entities, and 1+ for general operation/gameplay).

E.g. in monopoly deal, a gameplay class (e.g. Game) may store all the Players, and their Hands (Bank/Property) info, and may have methods like hasWonGame(Player), or a method to check whether the game has been won. Or a method like getNextCard() associated with the deck, or dealHand(), or addToHand() that operate on a Player. It is entirely up to you (there is no right/wrong design).

You will attempt to design these classes and diagram them (in UML form), then implement these classes so that they may be constructed as objects, initialized, and have methods to support getting/setting properties, and 2-3 specific gameplay operations (like those described above). The code you need to build is similar to that you did in lab 3 (to support the pacman ghost object… although in this, you have to think about multiple object types. USE HAS-A RELATIONSHIPS ONLY, i.e. no need to use inheritance (is-a) relationships in assignment 1 (they will naturally be utilised in assignment 2).

You DO NOT HAVE TO IMPLEMENT THE FULL FEATURE SET OF THE APP/GAME – just pick a few to implement, to show how one might work with these objects within a java application.

In **Assignment 2**, the classes will then be extended and integrated with UI elements & event handling to support both basic configuration of the game and the specific chosen elements of gameplay (the assignment may not necessarily offer a complete/polished game experience, however should be able to ultimately demonstrate certain key elements.

Assignment 1 deliverables:

1) **A 1-2 page written description of your project (what the purpose of the application will be, and what sorts of graphical elements and interactions you are hoping to support once completed)**

   The description should reference any sources you have used in your design (e.g. is it built upon an existing java card/board game? If so, you need to include the source, and outline how you are modifying this design – ***there MUST be some modification***, you cannot simply submit code you have discovered online)

2) **A UML diagram outlining any classes that you have designed for to support and encapsulate expected Entities you will need to use for the application, their properties/fields, constructors and methods (there should be at least 3+ classes, 2+ of which are entities that model game objects, and 1+ that model app operation/game play).**

   The UML diagram should show full class features and relationships between the classes (HAS-A only).

3) **Implementation of the above entities as several classes (or at least 3 of the above, if your design exceeds the above requirements).**

   Again, 2 of the classes should relate to objects used by the application (like a record/booking/order, or the user/player), and one class relating to the application/game itself (or a key aspect relating to gameplay, like a turn / deck / round).

   There should also be a way to output (to the console) the state of each object – one of which outputs an object visually (e.g. using RasterImage & Graphics2D, or the PApplet class from Processing.

Assignment 1:

RUBRIC (Marking Scheme) - 50 points total = worth 5% toward final grade

- Meets Goal 1 (10 points)
- Meets Goal 2 (10 points)
- Meets Goal 3 (10 points)
- Logical/Coherent Design (10 points)
- Aesthetics/Creativity (10 points)

***** ASSIGNMENT 2 DELIVERABLES AND REQUIREMENTS TO FOLLOW**

Assignment 1 & 2 Resources:

Java API: https://docs.oracle.com/javase/8/docs/api/
Java AWT/Swing: https://www.javatpoint.com/java-awt
Java Swing: https://docs.oracle.com/javase/tutorial/uiswing/components/index.html
Java FX: https://docs.oracle.com/javase/8/javafx/api/toc.htm
Processing: http://processing.github.io/processing-javadocs/core/

SUBMISSION- DUE 11:59pm, March 3, 2023

Submit the following files to assignment link: **"a1" on web-submit**
https://webapp.eecs.yorku.ca/submit/

- **Academic Integrity Statement** (+ group members - 2 MAX per group)
  - *see end of this document*
- **Pdf/Docx** files for task 1 (project description) & 2 (uml diagram)
- **Exported project file** (*.zip) including your java source files for task 3

EECS 1720M – W2023
ASSIGNMENT 1/2

ACADEMIC INTEGRITY STATEMENT

We (the undersigned) hereby confirm that this assignment represents the sole work of the individuals listed below.

We (the undersigned) confirm that this work has been completed in adherence to the Senate Policy on Academic Honesty, without unapproved collaboration or the use of unpermitted aids or resources.

We recognize the importance of academic integrity and understand that there is no tolerance towards academic dishonesty within the Lassonde School of Engineering. We are aware that any suspected breaches will be reported to the Academic Honesty unit within the Student Welcome and Support Centre, and may result in additional penalties in accordance with the Academic Honesty Policy.

Name: First, Last	**Login Name:** (indicate the login used for submission)	**Student No.:**	**Signature/Date:**