This assignment is intended to be done by your team of two (or in some cases three) students. You may collaborate on answers to all questions or divide the work for the team. In any case, the team should review the submission as a team before it is turned in.

Project 4 is the final evolution of the **Raiders of the Lost Arctangent** (or RotLA) game simulation. You may reuse code and documentation elements from your Project 2 and 3 submissions. You may also use example code from class examples related to Project 2. In any case, you need to cite (in code comments at least) any code that was not originally developed by your team.

Part 1: UML exercises and Semester Project Proposal – 25 points

Provide answers to each of the following in a PDF document:

1) (5 points) Provide a one-page project proposal for Projects 5/6/7 (aka the Semester Project). A typical project will involve development of a user interface, a data source, and internal program logic for operations. Examples of past projects are on Canvas here: https://canvas.colorado.edu/courses/87273/discussion_topics/961119

   Your program can be a web or mobile app, a game, a utility, a simulation – really anything that can be demonstrated for its operation. It must be in an Object-Oriented language (sorry C folks) as you will be required to demonstrate OO patterns, but the language does not have to be Java. Generally, for scoping the size of the program, each team member should plan to implement two to four use cases or functional program elements in projects 6 and 7. Project 5 will be a design effort with required deliverables to be detailed later. Your proposal should include:
   a) Title
   b) Team Members
   c) Description paragraph
   d) Language choice (including any known libraries or frameworks to be used)
   e) List of 2 to 4 functional elements per team member (ex. Login screen, Game piece graphics, etc.)

   In addition to the project proposal submission, please add an entry to this Google Doc to reserve your project (try not to directly duplicate another team's work): https://docs.google.com/document/d/134o2KmJSFf5hQseZhBbkAFmyKW9bJPbuoVFAQh8eEAg/edit?usp=sharing

2) (10 points) UML Sequence Diagram from Project 3. Select at least four top level active objects for your simulation (Ex. Adventurer, Creature, Room, System.out, etc.). Create a sequence diagram that shows the primary message or method invocations between these objects for the tasks performed in a typical game turn. The sequence diagram can be just the happy path, it does not have to show error conditions. The diagram should show object lifetimes during the operations.

3) (10 points) Draw a class diagram for extending the RotLA simulation described in Project 4 part 2. The class diagram should contain any classes, abstract classes, or interfaces you plan to implement. Classes should include any key methods or attributes (not including constructors). Delegation or inheritance links should be clear. Multiplicity and accessibility tags are optional. You should note what parts of your class diagrams are implementing the three required patterns below: Factory, Singleton, and Command.

Part 2: RotLA simulation extended – 50 points (with possible 10 point bonus)

Using the Project 2 and 3 Java code developed previously as a starting point, your team will create an updated Java program to simulate extending functionality of the RotLA game. The simulation should generally perform as previously enabled in Project 2 and 3 allowing for the simulation code changes to be refactored as follows:

Change Summary

- Change to a human-controlled Interface for an Adventurer
- Changes to Adventurers
- Changes to Treasures and End of Game
- Command Pattern for interacting with the user and the simulation functions
- Singleton Pattern for selected objects (using lazy and eager instantiation)
- Factory Pattern for creation of Adventurers and Creatures
- 15 required JUnit test assertions

Creating an Interface for an Adventurer

Instead of the game controlling 4 random adventurers, the game will be modified as follows:

A user will start a game by selecting an Adventurer type to play (Brawler, Sneaker, Runner, Thief) and will provide a custom name to be used for the Adventurer.

All interactions with the user will be via the command line with prompts for numeric or text inputs. Present the Facility drawing of the Rooms to the user prior to asking for commands.

In a turn, the user will have the Adventurer execute a command. The Adventurer starting the turn in an empty Room may be commanded to Move (to a connected Room), Search (for Treasure), or Celebrate (the user will randomly celebrate using Project 3 celebration subclass rules). The Adventurer starting the turn in a Room with Creatures may Fight (all Creatures) or Move (to a connected Room). If an Adventurer moves from a Room that has one or more active Creatures the Adventurer automatically suffers 1 damage per Creature in the Room the Adventurer is leaving.

All Creatures will take their turn normally (as in Project 2/3) after the Adventurer performs a command.

Changes to Adventurers

The new user-controlled Adventurer has a damage capacity based on type: Brawlers 12, Runners and Thieves 10, Sneakers 8

Adventurers with Stealth Combat skill have a 50% chance of avoiding damage in every combat they lose (this replaces the previous 50% chance of avoiding combat altogether).

Adventurers with Quick Search skill will not skip searches.

All Search skill checks will be easier by 3: Careful goes to 4+, Quick to 6+, Careless to 7+ to find a treasure.

Runners will be modified slightly – instead of getting two turns, they may move two Rooms in a turn (all other types can only move one Room per turn) – the user controlling a Runner can decide to interact with the Room they move into or be prompted to immediately move again into another connected Room. If they decide to move to a second Room, they can ignore any Creatures or hazards in the first Room they moved to – they just run through it.

Changes to Treasures and End of Game

When an Adventurer finds a Portal in a Search, they immediately are moved to a random Room in the four levels (not 0-1-1) as if they had moved to it in their turn. When an Adventurer finds a Potion, they immediately receive a 1 point damage bonus, which may allow them to have more damage capacity than they start with.

The game ends when the Adventurer enters Room 0-1-1 after having left it at the start of the game. The Adventurer wins if they enter Room 0-1-1 after (1) finding one of each type of treasure (Sword, Gem, Armor, Portal, Trap, Potion) OR (2) all Creatures are defeated. The Adventurer loses immediately if defeated by having 0 damage remaining OR by going to Room 0-1-1 without meeting the Treasure or Creature victory conditions above.

Command Pattern for interacting with the user and the simulation functions

You must implement the commands to the Adventurer from the user via prompts to the console. Each valid command (Fight, Move, Celebrate, Search) must be implemented using your interpretation of the Command pattern. You should clearly demonstrate a Command abstract class or interface, and a set of concrete subclasses or interface implementations for each of the commands. Invokers or Receivers for the Commands are your option.

Singleton Pattern for selected objects (using lazy and eager instantiation)

You must create the Logger and Tracker objects as Singletons, creating one using a lazy instantiation, and one with an eager instantiation. Logger and Tracker should function using the Observer pattern largely as implemented in Project 3, creating Logger-n.txt files, etc. You may use Singleton elsewhere as appropriate.

Factory Pattern for creation of Adventurers and Creatures

Your Adventurer and Creature creation should be encapsulated in creation focused Factory pattern code. You should clearly request different types of Adventurers and Creatures as products of subclassed concrete creators. You may use Abstract Factory if you feel it better fits your implementation.

15 required JUnit test assertions

You must implement 15 JUnit assertions as described in the bonus work for Project 3. You must capture the results of running the tests with a screen print or other output and include it clearly in your repo. See Project 3 for support links for adding JUnit assertions to your project.

Outputs, Comments, UML Class Diagram Update

Captured output: Run a single game for at least 12 turns (win or lose), demonstrating and capturing an example of each valid command line interaction (Move, Fight, Search, Celebrate) with the user and also using the board render report for each turn from Project 2, and the Tracker object output from Project 3. Capture this output in a text file called SingleGameRun.txt (can be from console cut and paste or direct write to a text file). Also capture all Logger-n.txt files for the run that are created from the Logger object. Also capture evidence of your JUnit run of your 15 assertions when you test your code in a JUnitResults file of readable type.

Identify OO Patterns: In commenting the code, clearly identify your implementations of Factory, Singleton, and Command.

Include a UML class diagram update: Also include in your repository an updated version of the RotLA UML class diagram from 4.1 that shows your actual class implementations in project 4.2. Note what changed between part 4.1 and part 4.2 (if anything) in a comment paragraph.

There may be possible error conditions that you may need to define policies for and then check for their occurrence. You may also find requirements are not complete in all cases. Document any assumptions in the project's README file.

Bonus Work – 10 points for a Java-based Line Graph

There is a 10-point extra credit element available for this assignment. For extra credit, import a charting library to create a line graph of game events. The graph should be generated at the end of a game. The X axis should be the turns in the game from 1 to the end. The Y axis should be count values. The lines should represent number of Treasures found, number of Creatures active, and damage level of the Adventurer. The graph should capture enough turns that the graph has at least 12 data points.

Java charting libraries in common use include: JFreeChart (https://www.jfree.org/jfreechart/), charts4j (https://github.com/julienchastang/charts4j), and XChart (https://github.com/knowm/XChart). To receive all bonus points, graph generation code must be clear and commented, and the output for the graphs must be captured as images in a PDF or other viewable files included in your repo.

Grading Rubric:

Homework/Project 4 is worth 75 points total (with a potential 10 bonus points for part 2)

Part 1 is worth 25 points and is due on Wednesday 10/12 at 8 PM. The submission will be a single PDF per team. The PDF must contain the names of all team members.

Question 1 will be scored on completeness of your proposal (-1 per missing items). You will be contacted if there are concerns or questions about your proposal.

Question 2 will be scored based on your effort to provide a thorough UML sequence diagram that shows the flow of your Project 3 simulation. Poorly defined or clearly missing elements will cost -1 to -2 points, missing the diagram is -10 points.

Question 3 should provide a UML class diagram that could be followed to produce the RotLA simulation program in Java with the new changes and patterns. This includes identifying major contributing or communicating classes and any methods or attributes found in their design. As stated, multiplicity and accessibility tags are optional. Use any method reviewed in class to create the diagram **that provides a readable result**, including diagrams from graphics tools or hand drawn images. **The elements of the diagram that implement the Command, Singleton, and Factory patterns should be clearly annotated.** A considered, complete UML diagram will earn full points, poorly defined or clearly missing elements will cost -1 to -2 points, missing the diagram is -10 points.

Part 2 is worth 50 points (plus possible 10 point bonus) and is due Wednesday 10/19 at 8 PM. The submission will be a URL to a GitHub repository. The repository should contain well-structured OO Java code for the simulation, SingleGameRun.txt, the Logger-n.txt files, proof of JUnit run in JUnitResults of appropriate readable file type, the updated UML class diagram for Part 2, the bonus Line Graph image (if provided) and a README file that has the names of the team members, the Java version, and any other comments on the work – including any assumptions or interpretations of the problem. Only one URL submission is required per team.

20 points for comments and readable OO style code: Code should be commented appropriately, including citations (URLs) of any code taken from external sources. We will also be looking for clearly indicated comments for the three patterns to be illustrated in the code. A penalty of -2 to -4 will be applied for instances of poor or missing comments, poor coding practices (e.g. duplicated code), or excessive procedural style code (for instance, executing significant program logic in main).

15 points for correctly structured output as evidence of correct execution: The output from a run captured in the text file mentioned per exercise should be present, as should be the set of Logger-n.txt files. A penalty of -1 to -3 will be applied per exercise for incomplete or missing output.

5 points for the README file: A README file with names of the team members, the Java version, and any other comments, assumptions, or issues about your implementation should be present in the GitHub repo. Incomplete/missing READMEs will be penalized -2 to -5 points.

10 points for the updated UML file showing changes from part 1 to part 2 as described. Incomplete or missing elements in the UML diagram will be penalized -2 to -4 points.

10 point possible bonus for clearly identified graph generation code AND a line graph generated by the Java code included in the repository in an easily readable format.

Please ensure all class staff are added as project collaborators to allow access to your private GitHub repository. Do not use public repositories.

Overall Project Guidelines

Be aware that the class midterm will run from noon Saturday 10/15 through the end of the day Thursday 10/20. The exam window will be three hours (longer for those with accommodation). Plan ahead to make sure you allocate time for the project work and the midterm. To help with this, I will waive the 5% late penalties for the first two days after the due date, but the 15% penalty for the last two days will still be in place.

Assignments will be accepted late for four days. There is no late penalty within 4 hours of the due date/time. In the next 48 hours, the penalty for a late submission is normally 5%, but is waived for this assignment. In the next 48 hours, the late penalty increases to 15% of the grade. After this point, assignments will not be accepted.

Use e-mail or Piazza to reach the class staff regarding homework/project questions, or if you have issues in completing the assignment for any reason.