This assignment is intended to be done by your team of two 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 3 is intended as a continuation of the Project 2 game simulation of the **Raiders of the Lost Arctangent** (or RotLA). You may reuse code and documentation elements from your Project 2 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 – 25 points

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

1) (15 points) For the existing RotLA Project 2, create either a detailed UML Activity diagram to describe the flow of actions and decision points in the simulation, or a detailed UML State diagram that shows program states and transitions that cause state changes. Which one you use likely depends on whether your code is more code flow or state/transition based.
2) (10 points) Draw a class diagram for extending the RotLA simulation described in Project 3 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: Strategy, Decorator, and Observer.

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

Using the Project 2 Java code developed previously as a starting point, your team will create an updated Java program to simulate extended the elements of the RotLA game. The simulation should perform all functions previously enabled in Project 2. Generally, Adventurers and Creatures will continue to perform all functions they performed in Project 2. The simulation code will be refactored as follows:

Change Summary

- Introduction of Treasure objects
- Strategy Pattern for search and combat
- Decorator Pattern for celebration after combat
- Observer Pattern with Logger and Tracker subscribers

Treasure Objects

- Treasure will now be represented by objects (via an inheritance hierarchy).
- 24 Treasure objects will now be randomly placed in Rooms at the start of the simulation (4 of each of 6 subtypes). There are six subtypes (subclasses) of Treasure: Sword, Gem, Armor, Portal, Trap, Potion
    - A Sword provides the Adventurer holding it with a +1 bonus to all combat rolls against Creatures
    - A Gem is a cursed item, and the Adventurer holding it makes Creatures even angrier which gives a +1 bonus to combat rolls by Creatures against the Adventurer
    - An Armor Treasure provides a -1 to Creature's combat rolls against the Adventurer holding it
    - A Portal can be used by an Adventurer as an alternative to moving, and will randomly teleport the Adventurer to another Room inside the four levels of the Facility. You can decide what will make the Adventurer use the Portal – a random chance, a circumstance, etc.

    - A Trap will immediately cause 1 point of damage to an Adventurer (as if a combat was lost)
    - A Potion will allow an Adventurer to take an additional point of damage before being eliminated (i.e. allow the Adventurer to take 4 damage vs. 3 damage before being eliminated)
- Adventurers will only find a Treasure in a Room through a successful Search. Rooms without Treasure do not need to be Searched.
- If an Adventurer finds a Treasure during a Search, the Treasure is removed from the room, and the Treasure is assigned to the Adventurer (as an inventory item or Adventurer property).
- Adventurers can only hold 1 of each type of Treasure, so if they find a second Sword or Potion, for instance, they will not remove it from the room.
    - There is an exception to this for Traps. The Trap Treasure type effect will impact any Adventurer who finds it – the Trap is removed from the Room and damages the Adventurer regardless of whether the Adventurer has found other Traps before.

Similar to Project 2, the Game will end if all placed Treasures are found, if all Adventurers are eliminated, or if all Creatures are eliminated.

Strategy for Search and Combat

Using the Strategy pattern, create, assign, and use subclassed custom search and combat algorithms for Adventurers. When an Adventurer is instantiated, assign concrete search and combat algorithms to the Adventurers by their type.

- Combat algorithms
    - Stealth – has a 50% chance of avoiding a combat for each Creature engaged; no bonus to 2 dice roll for fighting a Creature
    - Untrained – no bonus to 2 dice roll for fighting a Creature
    - Trained – gains +1 to 2 dice roll used for fighting a Creature
    - Expert – gains +2 to 2 dice roll used for fighting a Creature
- Search algorithms
    - Careful – will find a treasure on a roll of 7+ on 2 dice, no effect from Trap treasure 50% of the time
    - Quick – has a 33% chance of skipping the search, otherwise will find a treasure on a roll of 9+ on 2 dice
    - Careless – will only find a treasure on a roll of 10+ on 2 dice
- Assignments of algorithms
    - Brawlers: Expert Combat, Careless Search
    - Sneakers: Stealth Combat, Quick Search
    - Runners: Untrained Combat, Quick Search
    - Thief: Trained Combat, Careful Search

Note that these algorithms for search and combat replace the original description of differences in how Adventurers would find treasure or fight. Also remember that held Treasures impact combat in addition to the assigned algorithm.

Decorator for Celebration after Combat

Add a Decorator pattern to extend the functionality of Adventurer combat. The added abstract Decorator will be called Celebrate and will decorate the Combat algorithm class. The concrete Decorators subclasses of Celebrate will be Shout, Dance, Jump, and Spin. If an Adventurer defeats a Creature in Combat, there is a chance (your option) that they will perform 0 to 2 (randomly determined) of each of the Celebrate concrete actions (represented by a printed message) as a follow up to the Combat result. All celebration execution should be printed in a single message – for example: "Sneaker celebrates: shout, jump, jump, spin, spin."

Your implementation of this Decorator should clearly show that the Celebrate methods are additional extensions to the Combat class, and are not part of that class (i.e. the Combat algorithm class code should not be changed to represent any Celebrate functionality).

Observer for Logging Events

Add an Observer pattern implementation to allow subscription for and publication of events. You may create a custom Observer framework, or you may use a library (e.g. Java Flow) or tool. You may use a pull or push model for events. Event messages can be text, JSON, or specialized event objects of your design.

Use the Observer implementation to publish a summary of game simulation events.

- Publish the following events:
  - Adventurer/Creature enters room
  - Adventurer/Creature wins/loses combat
  - Adventurer celebrates
  - Adventurer damage points change
  - Adventurer/Creature is defeated/removed
  - Treasure is found by Adventurer (include type of treasure)
- Sufficient information should be provided in the published event that the subscriber does not have to query any game objects for additional information.
- Create an event consumer class called a Logger. The Logger object should be instantiated at the beginning of each full Adventurer/Creature turn and should close at the end of each turn. The Logger object should subscribe for the published events that occur during a turn and write each of them in a human readable form as they are received to a text file named "Logger-n.txt" where n is the turn number of the simulation.
- Create an event consumer class called a Tracker. The Tracker object will be instantiated at the beginning of the simulation run and stay active until the end. The Tracker will subscribe for the published events and maintain a data structure in memory for current game status. At the end of each day the Tracker should print a summary of the cumulative data like:

  Tracker: Turn 4

  Total Active Adventurers: 3

Adventurers	Room	Damage	Treasure
Brawler	1-1-1	0	Gem, Sword, Armor
Sneaker	4-0-1	2	Sword
Runner	2-2-2	3	Portal
Thief	2-1-0	1	Gem, Sword, Armor, Portal

Total Active Creatures: 4

Creatures	Room
Orbiter	2-0-1
Orbiter	4-1-0
Seeker	3-1-1
Blinker	2-1-2

Outputs, Comments, UML Class Diagram Update

Captured output 1: Run a single simulated game until complete, showing the board render report for each turn from Project 2, as well as the Tracker object output described above. 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 that run that are created from the Logger object.

Captured output 2: Without using the Tracker or Logger subscribers, run the game 30 times, printing out only the run number and the result (all Treasure found, all Adventurers eliminated, all Creatures eliminated). Print out a final summary of the counts of each type of ending from the 30 runs. Capture that output in a text file called MultipleGameRun.txt.

Identify OO Patterns: In commenting the code, clearly identify your implementations of Strategy, Observer, and Decorator.

Include a UML class diagram update: Also include in your repository an updated version of the RotLA UML class diagram from 3.1 that shows your actual class implementations in project 3.2. Note what changed between part 3.1 and part 3.2 (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 JUnit test example

There is a 10-point extra credit element available for this assignment. For extra credit, import a version of JUnit of your choice, and use at least ten JUnit test (assert) statements to verify some of your starting expected objects are instantiated or to perform other similar functionality tests. For full bonus points you must document how you run your JUnit tests (e.g. with a command line or in the IDE), and you must capture output that shows the results of your tests. You can decide how the test methods are integrated with your production code.

In practice, writing your tests before development is recommended, but for this academic example, I recommend you do not pursue this bonus work until you are sure the simulation itself is working well. If you need support on using JUnit, I mention several references in the TDD lecture, but here are key helpful ones:

- The JUnit sites for JUnit 5 (https://junit.org/junit5/) and JUnit 4 (https://junit.org/junit4/)
- The Jenkov JUnit tutorials (they are for JUnit 4, but are extremely clear and helpful regardless): http://tutorials.jenkov.com/java-unit-testing/index.html
- Organizing your JUnit elements in your code: https://livebook.manning.com/book/junit-recipes/chapter-3/1

Grading Rubric:

Homework/Project 3 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 9/28 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 based on your effort to provide a thorough UML activity or state diagram that shows the flow of your Project 2 simulation. Poorly defined or clearly missing elements will cost -1 to -3 points, missing the diagram is -15 points.

Question 2 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 Observer, Strategy, and Decorator 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 10/5 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, MultipleGameRun.txt, the Logger-n.txt files, the updated UML class diagram for Part 2, 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.

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

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 5%. 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.