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 2.1: OO and UML exercises – 20 points**

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

1. (5 points) What does "design by contract" mean in an OO program? What is the difference between implicit and explicit contracts? Provide a text, pseudo code, or code example of each contract type.

2. (5 points) What are three ways modern Java interfaces differ from a standard OO interface design that describes only abstract method signatures to be implemented? Provide a Java code example of each.

3. (5 points) Describe the differences and relationship between abstraction and encapsulation. Provide a text, pseudo code, or Java code example that illustrates the difference.

4. (5 points) Draw a UML class diagram for the RotLA simulation described in Project 2.2. The class diagram should contain any classes, abstract classes, or interfaces you plan to implement to make the system work. Classes should include any key methods or attributes (not including constructors). Delegation or inheritance links should be clear. Multiplicity and accessibility tags are optional. Design of this UML diagram should be a team activity if possible and should help with eventual code development.

**Project 2.2: The RotLA game simulation – 30 points**

The code you develop here for Project 2 will also be used for Projects 3 and 4. In most semesters we use Project 2, 3, and 4 to practice OO programming by simulating days of activities at different places, zoos, restaurants, garages, etc. This time, we are going to simulate a self-playing game I call…

**Raiders of the Lost Arctangent** (or RotLA): It is loosely based on an old silly board game called Temple of the Beastmen by Lester Smith from Game Designer's Workshop.

**Simulating the RotLA Game**

In RotLA, our Adventurers discover the entrance to a long-lost buried structure. At ground level is an entrance Room (0-1-1) to a four-level deep ancient Facility. The Facility's center Rooms are labeled 0-1-1, 1-1-1, 2-1-1, 3-1-1, and 4-1-1 and are always connected from floor to floor. The center Rooms are surrounded by eight Rooms on each level and they have connections vertically (North/South) and horizontally (East/West) to adjacent Rooms on the same level. Only the center Rooms allow movement to the next level Up or Down (as well as to adjacent Rooms).

Our brave Adventurers start in room 0-1-1. The four Adventurers include a Brawler, a Sneaker, a Runner, and a Thief.

- Brawlers get a +2 to dice rolls when fighting a Creature.
- Sneakers have a 50% chance not to have to fight Creatures found in their Room.
- Runners get to move twice in a turn (including moving to Rooms, Creature fighting, and treasure seeking activities).
- Thieves get a +1 to finding a treasure and a +1 to fighting a Creature.

Strange and dangerous Creatures roam the Facility. When the game starts, four of each type of Creature below will be placed in the Facility in random starting Rooms (as described below).

- Orbiters – Orbiter Creatures circle the outer Rooms of the level they are on, whether connected or not – they may go in clockwise or counterclockwise directions. They will always start in an outer Room on any of the four levels. They will not move if in a Room with an Adventurer.
- Seekers – Seeker Creatures move to join any adjacent Adventurer on their level. Seekers can start anywhere in the four levels. If no Adventurer is adjacent to them or an Adventurer is already in their Room, Seekers will not move.
- Blinkers – Blinker Creatures randomly move each turn to any Rooms across all four levels (not 0-1-1). They always start in a random Room on level 4. They will not move if in a Room with an Adventurer.

Adventurers move each turn into a connected Room in the Facility, randomly choosing a valid direction, moving one Room at a time. Adventurers will not leave the Facility (i.e. move back to 0-1-1 once they leave it).

On a given turn, first all Adventurers move, then fight or search for treasure. After all Adventurers have moved, all Creatures move and fight any Adventurers in their Rooms. When this is completed, the turn is over.

When Adventurers move into a random connected room, they may find one or more Creatures there. If there is one or more Creatures in a room, the Adventurer will spend the turn fighting each Creature present (note that a Sneaker has a 50% chance to avoid fighting Creatures).

When fighting each Creature, both the Creature and the Adventurer roll two 6 sided dice (two random integer numbers 1 to 6 added together). If the Adventurer's dice score is greater than the Creatures, the Creature is defeated and is removed from the game. If the Creature has a greater score than the Adventurer, the Creature does a point of damage to the Adventurer. Once an Adventurer has taken 3 points of damage, it is removed from the game. On a tie, no action is taken.

If the Adventurer moves into an empty Room, they may look for treasure. The Adventurer will find treasure by rolling a 10 or more on the sum of two 6-sided dice.

Once Adventurers are done, Creatures move per their special movement. If they enter a Room or begin in a Room with one or more Adventurers, the Creature will fight each Adventurer present.

Over all the turns of the game, if the Adventurers as a group find ten treasures or eliminate all Creatures, or if the Creatures eliminate all Adventurers, the game ends. Credit for finding treasure is maintained even if Adventurers are eliminated.

Drawing the facility: Rooms are designated by a triplet of level, N/S value, and E/W value. The following example is a drawing of the ground level and the first two levels down of the Facility in a typical turn. Room 1-0-0 has a Brawler as well as an Orbiter and a Blinker in the room. Room 1-2-1 has a Seeker. Room 1-2-2 has a Shooter. Room 2-1-1 has the Runner and Thief, Room 2-2-2 has a Blinker

Each turn draw the Facility as shown below (extended for all 4 levels). Include a summary report for each turn of the Adventurers and Creature status, how many treasures they have found, and how much damage they have. For example:

```
RotLA Turn 6:

0-1-1: - : -

1-0-0: B: OB      1-0-1: - : -      1-0-2: - : -

1-1-0: - : -      1-1-1: - : -      1-1-2: - : -

1-2-0: - : -      1-2-1: - : S      1-2-2: S : -

2-0-0: - : -      2-0-1: - : -      2-0-2: - : -

2-1-0: - : -      2-1-1: RT : -     2-1-2: - : -

2-2-0: - : -      2-2-1: - : -      2-2-2: : B

<levels 3 and 4 omitted>

Brawler – 1 Treasure(s) – 0 Damage
Sneaker – 2 Treasure(s) – 2 Damage
Runner – 0 Treasure(s) – 0 Damage
Thief – 2 Treasure(s) – 1 Damage

Orbiters – 1 Remaining
Seekers – 2 Remaining
Blinkers – 0 Remaining
```

**Building an OO version of the Simulated Game**

Create an inheritance hierarchy for Adventurers and the four subclasses of Brawler, Sneaker, Runner, Thief. Use an ArrayList to hold all active Adventurer object instances.

Create an inheritance hierarchy for Creatures and the three subclasses of Orbiter, Seeker, and Blinker. Use an ArrayList to hold all active Creature objects instances.

Use the subclasses to handle any special movement or actions on the part of the different types of Adventurers or Creatures.

Create a structure for holding the facility Room objects, including the ground Room 0-1-1, and nine Rooms per each of four levels as shown above. This could be a simple ArrayList for holding Room objects that know what their connected Rooms are, and what Creatures or Adventurers are presently in the Room. You could also use more advanced Java containers.

Create a board renderer that can use the collections of Rooms, Adventurers, and Creatures to draw the levels and occupants, as well as the status report, in the general format shown above.

Create a game engine that initializes the elements, and loops through turns, visiting all Adventurers and then all Creatures to process their turns (e.g. move, then fight or seek treasure), and watch for ending conditions. Make sure to end the simulation run by printing out the reason the game ends (all treasure found, all Adventurers eliminated, all Creatures eliminated).

**Output for and Thoughts on the Simulated Game**

The code should be in Java 8 or higher and should be object-oriented in design and implementation.

*Captured output:* Run the simulated game in two ways. First, run a single simulated game, showing the board render report for each turn, until the game ends. Capture this output in a text file called SingleGameRun.txt (can be from console cut and paste or direct write to a text file). Then without printing the turn by turn changes or board renders, 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 Elements:* In commenting the code, include comments for at least one example of: Inheritance, Polymorphism, Cohesion, Identity, Encapsulation, and Abstraction.

*Include a UML diagram update:* Also include in your repository an updated version of the RotLA UML diagram from 2.1 that shows your actual class implementations in project 2.2. Note what changed between part 2.1 and part 2.2 (if anything) in a comment paragraph.

Hints on construction – I suggest you start with a very simple version of the simulation that has small examples of all elements and actions (this is known as a “vertical slice”), then add variations and more instances once you have the basics working. You may be tempted to use advanced containers like Maps or Graphs to hold data, but you can get a lot done with simple ArrayLists of objects…

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.

**Grading Rubric**

**Project 2 is worth 50 points total.**

**Part 2.1 is worth 20 points and is due on Wednesday 9/14 at 8 PM.** The submission will be a single PDF per team. The PDF must contain the names of all team members.

Questions 1-3 will be graded on the quality (not quantity) of the answer. Questions 1-3 should include examples and supporting citations. Solid answers will get full points, poor-quality or missing elements for answers will cost -1 or -2 points each, missing answers completely will be -5 points.

Question 4 should provide a UML class diagram that could be followed to produce the RotLA simulation program in Java. This includes identifying major contributing or communicating classes (ex. Rooms, Adventurers, etc.) 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. 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 2.2 is worth 30 points and is due Wednesday 9/21 at 8 PM.** The submission will be a URL to a GitHub repository. The repository should contain well-structured OO Java code for the RotLA simulation, the two captured text files with program results requested, and a README file that has the names of the team members, the Java version, and any other comments on the work – including clearly documenting any assumptions or interpretations of the problem. Only one URL submission is required per team.

10 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 six OO terms to be illustrated in the code. A penalty of -1 to -2 will be applied for instances of poor or missing comments or excessive procedural style code (for instance, executing significant procedural logic in main).

5 points for correctly structured output: The output from the single and multiple runs captured in the text files mentioned for the exercise should be present. A penalty of -1 to -2 will be applied for each incomplete or missing output file.

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

10 points for the updated UML file from project 2.1 as described, including how the structure changed between projects 2.1 and 2.2. 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.