CSCI 3081W: Program Design and Development
Final Project: New Feature Extension
Fall 2023

Please read the whole thing!
(No deadline extensions will be provided)

Subject to minor changes until Wednesday, November 22nd at 11:59 pm.
Peer review got added (Wednesday, December 6th at 10:22 am.)

New Feature Checkoff due date: Wednesday, November 22nd at 11:59 pm.
Writing, Coding, and Presentation due date: Sunday, December 17th at 11:59 pm

Goal

Design and implement new feature(s) to add to the ongoing project.

Source Code:

1) Your lab10 solution
   OR
2) Provided base code https://github.umn.edu/umn-csci-3081-f23/public-hw4-basecode.
   a) This will release on Friday, November 17th
   OR
3) Provided Team homework 4 repository (i.e. Team-099-99-hw4)
   a) This will release on Friday, November 17th

Overview:

In this homework assignment, you will apply all the concepts you've learned so far from design patterns to sprints (workshop 6). Consider the lab project as an ongoing project in your workplace. As a team of agile coders, you all will have ~4 weeks to impress your product owners (the instructors and TAs). You have to decide on a feature(s) addition to this project, and come up with a good design pattern which leads to seamless integration. The report you submit will inform the reader about your feature (this refers to the checkoff - lab11). The implementation must use a pattern which creates a robust design and works well with any past and future features. The video should show a demo of this feature, before deployment to DockerHub.

What you'll learn this assignment:

1. Designing a new feature(s) for a project.
2. Applying agile skills by working in a sprint.
3. Creating a UML for a new feature.
4. Coding the new feature(s) for an existing codebase.
5. Presenting the implementation through a YouTube video (make sure we are able to see it).

Instructions:
Work with your team members but be sure that you understand the solutions you come up with and not one person is doing all the work. If you need help, you can talk with your TAs.

Where to store your source code?
We will create a github repo for you. Please do not make your own private repo.
Repo format: Team-001-01-homework4

Designing a New Feature:
The feature that you pick must be **significantly** interesting**!**
We recommend that you get checked off before coding your idea! If the idea is trivial and the idea was not checked off, you will have points deducted.
We also have a Canvas submission for getting your idea checked off (due Wednesday, November 22nd)!
The checkoff is required!

Teams of 2: 1 non-trivial feature with a design pattern(s) as the solution
Teams of 4: 2 non-trivial features each with a design pattern(s) as the solution

Examples:

1. Decorator & Factory - battery wrapped around the drone and add recharge stations; logic for factory pattern and business implementation must be updated too.
2. Singleton - data collection, then output to .csv and analyze the data that you collected
3. Observer - notifications
4. Mediator - stop signs
5. State - drone attributes
6. Builder & Decorator - drone specialization
7. If you come up with another idea and want to get it checked off as "valid", please talk to one of your TAs during Lab 11!

Checkoffs
Once you have your features and ideas, you **must** get them checked off with an instructor or graduate TA. Then submit to Canvas for final approval/official check off, so that it is officially documented and can be referenced back to.

Requirements (whole project):

If an individual has 0 GitHub commits AND the peer review(s) about them are poor, they will receive a 0 on Homework 4.

Coding

- Use design pattern(s) to implement your new feature(s)

    - Brainstorm different design patterns, your first choice may not always be the best one.
    - If mid-way through your coding process, you see a better way to implement things, by all means try it out
- Place all of the newly created classes inside the *libs/transit* folder

  Important: If your code does not compile on CSElabs machines, you won't receive credit for the coding section.

<u>Documentation</u>

- Doxygen
    - Doxygen for ALL of the classes (existing and newly created)
    - You are **NOT** required to do doxygen for the classes under these folder
        - dependencies/
        - libs/routing/
        - libs/transit/util/
        - apps/
    - Doxyfile
- Google Code Style
    - Code Style for ALL of the classes (existing and newly created)
    - You are not required to do code styling for the classes under these folder
        - dependencies/
        - libs/routing/
        - libs/transit/util/
        - apps/
- Write Up
    - File Name: README.md (should be located in the GitHub repository)
    - Team number, member names, and x500.
    - What is the project about (overview of the whole project, not just the hw4)?
    - How to run the simulation (overview of the whole project, not just the hw4)?
    - What does the simulation do specifically (individual features i.e. movement of entities etc) (overview of the whole project, not just the hw4)?
    - New Feature
        - What does it do?
        - Why is it significantly interesting?
        - How does it add to the existing work?
        - Which design pattern did you choose to implement it and why?
        - Instruction to use this new feature (if the new feature is not user interactable, please mention this as well)
    - Sprint **retrospective**
- UML
    - Create UML diagram depicting **ONLY** your new feature
    - Upload this to the github repo and also display this in the aforementioned README file

Docker

- Upload and push your final code to DockerHub
- Put this link inside the GitHub README (be sure to state what the link is about).

YouTube Video Presentation (~5 min)

- If you create a PowerPoint (or presentation material) for your Youtube Presentation Video, please include these with your submission as a PDF in your GitHub repository.
- Explain the new feature that your team has implemented
    - What is it?
    - Why is this new feature important?
    - Which design pattern did you use?
    - Demo
    - Optional: Source code if you have more time
    - Everyone in the team must talk with video cameras on
- You can upload this video to YouTube and put the link that will redirect toward your video inside the README (be sure to state what the link is about). Make sure the teaching staff has access to this video. Feel free to set it to private or delete it once you have received your final score.

Peer Review

- Peer reviews are individual. You will assign a value for each team member in your group. The total will add up to 100. For example, if you have 2 people, you could do 50-50. There will also be a portion where you list each member's responsibilities. If a peer review comes back and a member has a 0, we will check your team's GitHub commits. If they have 0 commits, they will receive a 0 on homework 4.
- Review your teammates by filling out the form below.
- Form: https://forms.gle/BtL7PCiz5huEn3nU8 (Please do not be vindictive.)
- This is due on Sunday, December 17th at 11:59 pm.

Submission (due December 17th):

- Canvas:
    - Github repository link
- Github:
    - Source Code
    - Doxyfile
    - Write up (README file)
        - Write up
        - Docker link
        - Video presentation link
        - UML
- Peer Review: Fill out this form: https://forms.gle/BtL7PCiz5huEn3nU8 (Please do not be vindictive.)

Grading Breakdown:

- Coding: 45%
- Documentation: 40%
- YouTube Video Presentation: 5%
- Docker: 5%
- Peer review: 5%

Observer Pattern - Notifications

The goal of this feature is to add notifications to the sidebar in the 3D simulation using the observer pattern.

If you are not familiar with the observer pattern, feel free to check the Refactoring Guru Observer Example.

In our Drone Simulation System, there are several potential places where observer patterns can apply.

- The ***droneObserver*** which will print information about drones in the frontend UI. For example, Drone1 is on the way to pick up Package1.
  Drone1 has picked up Package1.
  Drone1 is delivering the Package1 to Customer1.
  Drone1 has delivered the Package1 to Customer1.
  …etc.

  Here is the example screenshot from last semester (Uber simulation). As you can see, the corresponding information is printed out on the left side.

- The ***deliveryObserver*** which will print information about deliveries (packages and corresponding customers) in the frontend UI.
  In the above example, Robot1 is scheduled.
  Robot1 is picked up.
  Robot1 is delivered.
  …etc.

- Other observers based on what you want to observe. Feel free to incorporate anything relevant to drone simulation systems that you added into your simulation from lab10!

Decorator and Factory Pattern - Battery Wrap Around Drone and Recharge Stations

The goal of this feature is to add the battery on top of the drone in the 3D simulation using the decorator pattern. If you are not familiar with the decorator pattern, feel free to check the Refactoring Guru Decorator Example and also review the previous lab where we went through the decorator pattern. Additionally, you will need to extend your factory to include the creation of Recharge Stations.

By applying the decorator pattern on top of the Drone, you should be able to track the battery of the drone. For example, as time goes by, the battery of the drone decreases by 2% starting from 100%, then
if the battery is below some threshold or is <=0%, the drone should not be able to move/deliver the robot any more. In this case, there are some possible solutions but feel free to come up with other solutions based on your design.

1. Drone keeps track of its battery and goes to the recharge station when it detects a depleted battery.
2. Drone stops at where it is (0% battery) and waits for other entities to recharge it.
3. …etc.

You can also add other decorators based on what you want to decorate. Feel free to incorporate anything you added into your simulation from lab10!

Singleton Pattern - Data Collection

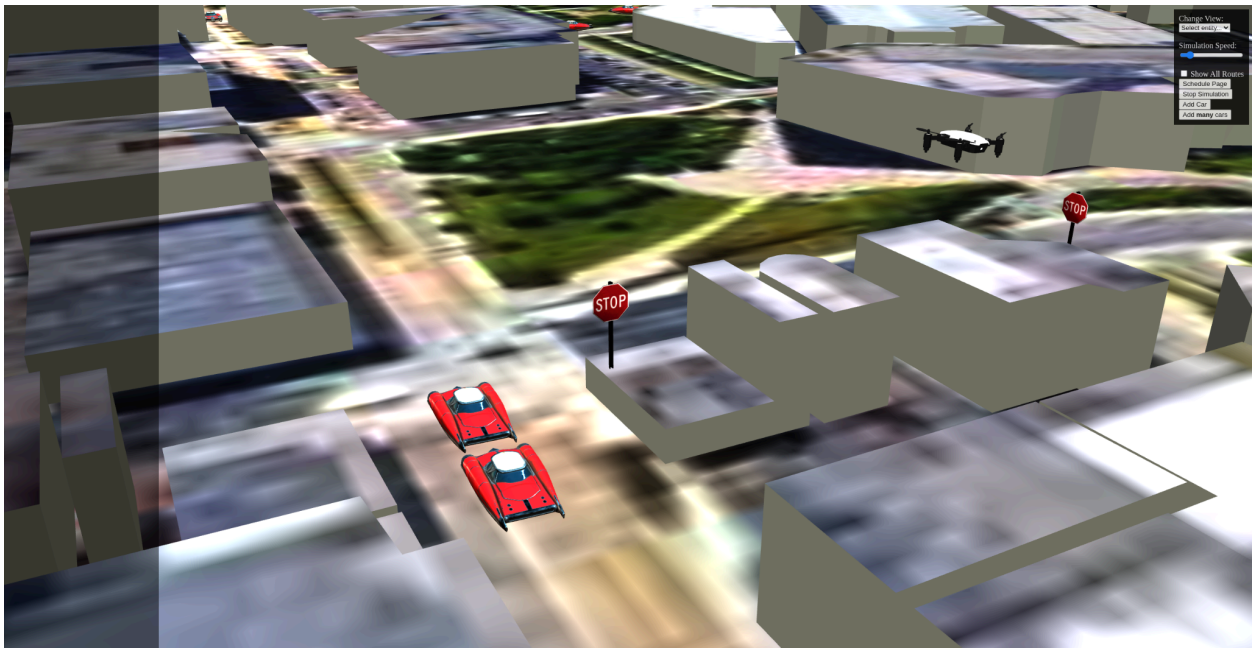
The goal of this feature is to add the ability to track data from the 3D simulation in a CSV file using the singleton pattern.

If you are not familiar with the singleton pattern, feel free to check the [Refactoring Guru Singleton Example].

In our Drone Simulation System, there are several potential places where singleton patterns can apply. For example, students can track a ***drone's*** or a ***robot's***

- Speed
- Amount of trips
- Fuel efficiency
- Directions
- Position
- …etc.
- Feel free to incorporate anything you added into your simulation from lab10!

Mediator - Stop Signs



The goal of this feature is to add intersections to the simulation using the mediator pattern to simulate traffic. Stop signs mark intersections and only allow one entity to cross through the intersection at a time. Additionally, cars and drones do not collide with each other at intersections, to more accurately simulate traffic. This functionality can be added using the decorator pattern to create 'Collidable' entities which must not collide with each other. The simulation starts with many Car entities and a handful of Intersection entities to create some traffic. Deliveries are scheduled normally, but now Drones must wait in traffic along their route.

Example implementation

Mediator design pattern

- Rather than having entities talk directly with each other to pass information and make decisions, a central mediator facilitates messages and decision making which can drastically simplify program logic and follow SOLID design principles.
- Multiple Intersection entities exist in the simulation and mediate Collidable entities within a certain radius of the intersection to only allow one entity to move at a time. Each intersection keeps track of the order that entities arrived at the intersection, then lets them through one by one.

Decorator design pattern

- Certain entities (Drones and Cars) are not allowed to collide with each other only if they are facing the same direction. The Collidable decorator adds additional functionality to these classes by adding to the update() function to prevent the entity from moving if the entity will collide.

- Collidable decorated entities will queue up at intersections, so the mediator Intersections only have to deal with nearby entities, not all of them!

State - Drone Attributes

The goal of this feature is to modify the drone's behavior at different states of the simulation.

If you are not familiar with the state pattern, feel free to check the Refactoring Guru State Example.

In the simulation, there are many possible places where the state pattern can be implemented.

Basic state implementation for Drone:

- Drone will have 3 states:
    + Available:- "Unassigned and Ready" - Indicates the drone is currently not tasked with a delivery and is ready for deployment.
    + In-transit:- "En Route to Pickup" - Describes the drone's state when it is actively moving towards the location to pick up a package.
    + Delivering:- "Destination Delivery in Progress" - This state signifies that the drone has picked up the package and is currently on its way to the delivery destination.
- The following attributes for Drone can be modified at different states:
    + Speed
    + Color
    + Altitude i.e. as high altitude when in-transit and low when delivering and pickup
- The drone should be able to modify itself through the state pattern. A simple example would be changing the drone's speed depending on the number of completed trips.

Possible state extension:

- Having a failure state where the chance of the drone breaking down and requiring replacement increases as the total distance traveled increases.
- Charging the delivery rate , the delivery rate is dynamically adjusted based on the time the drone spends in the 'Delivering' state such that the longer the drone takes to deliver a package, the lower the delivery rate or fee.

Builder & Decorator - Drone Specialization

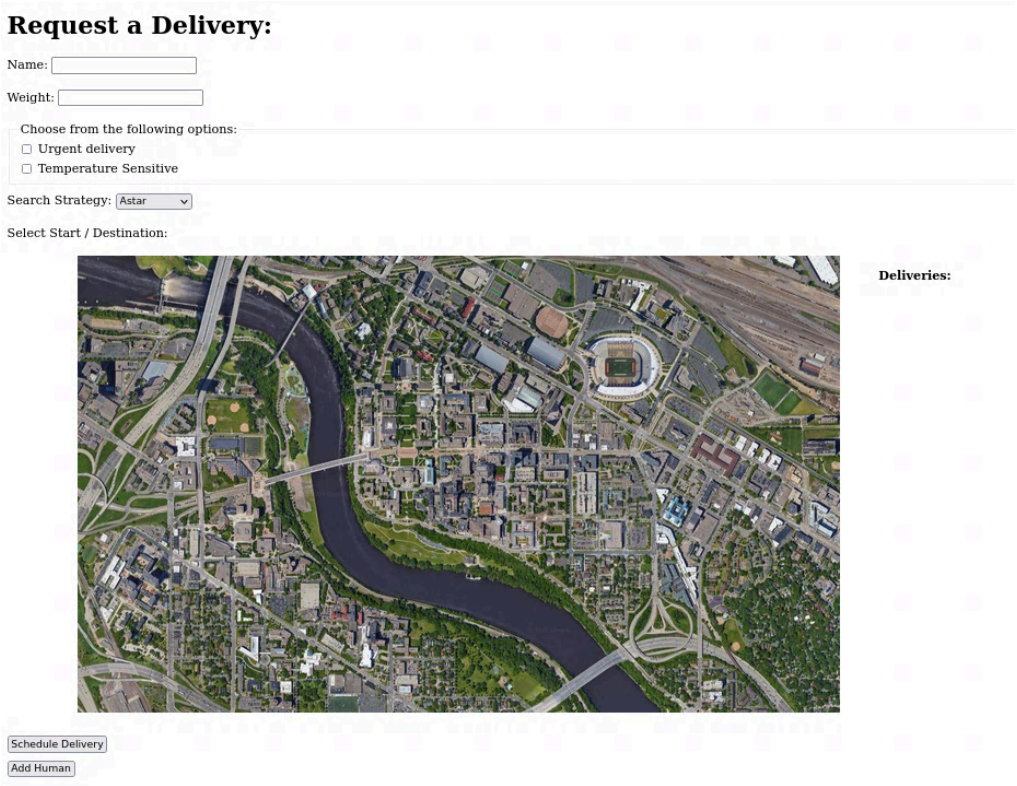
The goal of this feature is to allow drones to specialize their abilities for delivering special types of packages. You will use the decorator design pattern to give the packages new requirements and the builder design pattern to give the drone new abilities.

If you need reminders about these patterns, take a look at Refactoring Guru Builder Example and Refactoring Guru Decorator Example.

Based on the specific requirements of the package scheduled for delivery, only certain drones can pick them up. For example, a really heavy package will need a drone with wheels to deliver it over ground, a package containing sensitive medical samples and supplies will need a drone equipped with a cooler, and a package which is marked urgent (meaning the robot paid extra for expedited prime shipping) will need a drone with extra powerful propellers to deliver it using the beeline strategy over the tops of the buildings.

A couple of notes for implementation:

- You will need to allow the user to specify the weight of the packages (i.e. let the simulation choose the drone on the basis of the weight of the package). Add options such as Urgency and cooling requirements in the scheduling page. The above conditions should aid the simulation to choose the appropriate drone. See an example below.



- To begin, the system will only have 1 basic drone. When a package is scheduled, if no drones currently in the system are *able* to carry it, then a new drone is spawned with features to match the package's requirements. This new drone now exists in the system to take future deliveries.
    - You will need to indicate what type of drone is taking each package. You can implement this in a number of ways, including printing out the features of the drone taking the package or importing new assets.
- You will need to implement a new algorithm for deciding which package to pick up for each drone. Here is one example (Note: this is a naive solution for simple implementation, not an optimal one):

1) Check if there is a package whose requirements match perfectly with the drone's features. For example, if a drone has wheels and a cooler, then it will try to pick up a heavy package which needs temperature control.
2) Check if there is a package whose number of requirements is one less than the number of the drone's features. For example, if the drone has propellers and a cooler, and there's one single-attribute package scheduled which requires temperature control, then it can pick up this one.
3) Repeat checking packages with even fewer attributes until all the packages are checked or the drone has picked up a package.

You don't have to follow our example, but make sure there's no deadlock with your algorithm. In other words, every package should be delivered in the end as long as there's a capable drone.