HW2 – Decoupling with Observers and Extending with Decorators

Estimated time: 12-16 hours

Objectives

- Become more familiar with UML modeling
- Become familiar with the Observer pattern
- Become familiar with the Decorator pattern
- Improve unit testing skills
- Become broaden and strengthen programming skills in related areas, including network communications and user interfaces.

Overview

In this assignment, you will build a simply road-race tracking system for cycling that will help race officials, support teams, and spectators monitor where cyclists currently are on a race course. In the building this system, you will look for and take advantage of opportunities to apply the observer and decorator patterns.

Background

Cycling is exciting endurance sport, but traditional road races can be difficult for officials to monitor, support personal to be in the right place, and spectators to enjoy. To see into these the challenges, it's necessary to understand some of following facts, circumstances, and conditions of road races

- Race routes can cover long distances and on all kinds of roads. For example, a local race named Lotoja starts in center of Logan and finishes in Jackson Hole Wyoming, covering 206 miles of country roads, mountain passes, and state highways.
- Each racer has a number that uniquely identifies that person. This number is called a bib number.
- In amateur races, racers compete in groups based on race experience and ability. Race groups are capped with a limit, typically somewhere between 50 and 100. All bib numbers for a group come from the same block or range of numbers. For examples, all of the racers in a Men Cat-1 group might have numbers between 1-99.
- Race groups start at different times, typically at least 5 minutes apart from each other.
- Racers from groups are not allowed to work with (ride close behind or in front of) racers from other groups for an extended time. Doing so is considered cheating and is a penalized.

- Officials need to monitor racers to make sure that they do not work with different groups. This is difficult do because official have to make mental notes about who is riding with who and then watch to see if that situation has doesn't change.
- Each racer is allowed to have a support team that provides food and water along the way, but only from designated areas called feed zones. A support team has to drive to a feed zone via an alternate route, hopefully arriving before its racers. If the team misses its racers, it a) needs to know this and b) get the next feed zone ahead of the racer. This is difficult to do if the support team doesn't know where its racer is on the course.
- To follow what's happening in a race, a spectator would like to know where certain racers are relative to other racers. Since vehicle traffic is either blocked or discouraged on the course, it is typically impossible to physical follow a favor rider. In general, a spectator would like to monitor a handful of racers for the whole race and follow others on and off on, depending what's happening.

Many amateur races know require racers to wear a RFI device that helps official record times at the finish line. The same technology could be used to track their position along the course.

Functional Requirements

Below is a list of functional requirements for a tracking system that would help official, support teams, and spectators follow road races. Assume that every racer is wearing an RFI device and that there are many sensors along the course that can detect when each racer passes a sensor's location. Also, assume that the sensor data can be transmitted via a wireless or cellular network connection to a server. Finally, assume that all the senor's clocks are synchronized to within 1/1000th of a second.

In general, we would like the server to be able to send out information to officials, support teams, and spectators according to their preferences and needs.

1. Race Groups
   1.1. A race event could involve between 1 and 50 race groups
   1.2. Each race group has a label, like "Mens Cat 1", "Mens Cat 2", "Womens Cat 1", "Juniors", etc.
   1.3. Each race group can have between 1-100 racers
   1.4. Each race group has an assigned block of numbers
   1.5. The racers in a race group must have a number from the assigned block
   1.6. Each race group has a designated start time
2. Racers
   2.1. Each racer number has a name and birth date
   2.2. Each racer must belong to one race group
   2.3. Each racer must have a bib number that corresponds to the race group
   2.4. A racer's official start time is the designated start time of the racer's group
   2.5. A racer will eventually have an end time or DNF, which means did not finish.
3. Sensors
   3.1. Each sensor has a unique identifier

3.2. Each sensor has a location on the race course, measured in miles with three decimal points of accuracy

4. Data Collection
   4.1. During a race, whenever a racer passes a sensor, that sensor will send it number, racer's bib number and time (accurate to the 100^{th} of a second, but expressed in milliseconds) to a server
   4.2. Multiple racers may pass the sensor at the same time
5. Cheating detection
   5.1. The system needs to automatically detect when two racers from different groups pass two consecutive sensors within 3 seconds of each other.
   5.2. To this end, the system should include a "cheating computer" that observer and encapsulates a list of detected cheaters.
      5.2.1. The cheating computer should receive updates about racer state (times at sensors) and look for pairs of racers from different groups with similar times at two consecutive sensors.
      5.2.2. When a cheating computer detects a pair of racers that match this criteria, then the cheating computer should add them to a list of cheaters (part of the "cheating computers") state.
      5.2.3. The "cheating computer" should be observable so racer official can receive notifications about the situation. See 6.3.
6. Subscribing to and receiving results
   6.1. Via a staff person (the one who is running the program), spectators and support teams should be able to subscribe to receiving data for any number of racers
      6.1.1. Spectators should be able to receive this information via email every 5 minutes or whenever there is a significant change in the relative positions of the racers
      6.1.2. To this end, the system needs to include an observer capable of observing one or more racers and sending emails when the states of those racers changes.
         6.1.2.1. This observer will need to know the real person's name and email address
      6.1.3. The staff person should be able customize the emails that get send to spectator or support team. Below are a list of things that should be available in the customization:
         6.1.3.1. Automatically add a header (like Hi <person name>)
         6.1.3.2. Automatically add a footer (like the racer organization's names)
         6.1.3.3. Automatically add a p.s. that contains a motivation or humorous quotes.
   6.2. Via a staff person (the one who is running the program), race prompters should be able to setup giant screens that display that should position data of race leaders and crowd favorites for groups of spectators.
      6.2.1. The staff person should be able to change what racers the giant screens are highlighting
      6.2.2. To this end, the system needs to include a big-screen observer capable of monitoring and display the state of 1-30 racers.
   6.3. Via a staff person (the one who is running the program), race officials should be able to subscribe to

For testing purposes, the customer (instructor) will provide a program that will simulate the sensors.

- The simulator will generate a CSV file containing a list race groups, where each line in the file containing the following fields: group number, label, start time (milliseconds after overall race start time), min bid number, max bib number.
- The simulator will generate a CSV file containing a list of racers, where each line in the file containing the following fields: first name, last name, bib number, group number
- The sensor simulator will send "Racer Status" messages to a server.
- Each Racer Status message will contain the following fields:
    - Sensor Id: integer
    - Racer Bib Number: integer
    - Time: integer representing the time milliseconds since the start of the first group
- The Racer State message will be formatted in JSON as follows: { "SensorId": "<value>", "RacerBibNumber" : "<value>", "Timestamp" : "value" }, where the <value>'s are character string representations of the field values.
- The Time in the messages generated by the simulators will be realistic, but the messages will be sent at an accelerated rate. Real races can take 6-12 hours; simulations will run in 2-4 minutes.

Design and Implementation Suggestions

- Study the Bouncing Ball system for examples of a) a control GUI and observer creation form, b) implementations of subject and observer classes, c) a concrete obverse that has a GUI, and d) decoupling the subject and observer in time by putting the refreshing of the observer's GUI on a timer.
- Study the sample server. It contains sample code for receiving the datagrams (messages) from the simulator.
- In your program, encapsulate the receiving and process of racer status messages in a class with that responsibility. Have one method be the main "receive data" method and run that method on its own thread.
- Think about what data structures you need to manage racer data, and perhaps race groups and sensors. Design structures that are good encapsulations and abstractions.
- Consider using a decorator to extend the capabilities of the email observers

Instructions

To build this system, you will need to do the following:

1. Design, implement, and test the core server functionality, which should include appropriate subjects whose state will be observed by subscribers
2. Design, implement, and test subscribing and unsubscribing functionality for spectators, support teams, and race officials. This includes:
    2.1. Deciding what data needs to be sent to each kind of subscriber
    2.2. An user interface that allows a staff person to
        2.2.1. Create observers of the following kinds

2.2.1.1. email observers for race officials that observer the cheating computer
2.2.1.2. email observers for support teams and spectators
2.2.1.3. big-screen observers

2.2.2. Subscribe or unsubscribe 2.2.1.2 and 2.2.1.3 observers with specific racers
2.2.3. Allow a staff person to customize email observers with headers, footers, and quotes.

3. Capture your designs in appropriate UML class, interaction, and state-chart diagrams.
4. Test your software at the unit level with meaningful and relatively thorough executable test cases.
5. Test your software at the system level using the sensor simulators.

Related Technologies

To complete this assignment you will have to use work with network communications and user interfaces. You should have some experience with the later from early CS classes. For network communications, you may need to take an hour or so and learn about how to use UDP in your selected development environment.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw2_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	Max Points
A clear and concise designs consisting of UML class, interaction, and state diagrams	20
A working implement, with good encapsulation, abstractions, inheritance, and aggregation, and appropriate use of the observer, decorator, and other pattern	40
Meaningful, executable unit test cases	20
Reasonable systems testing using the simulator	20