Module 11
Design Patterns

CS W169A: Software Engineering

1 Overview

The first part of this worksheet is meant to introduce the reader to some practical code examples of some of the design patterns we've discussed in class. This tutorial is largely based off the Design Patterns in Ruby documentation. You are definitely encouraged to read about the other design patterns with code examples, but we will go over the patterns we deem more relevant to the class in this worksheet. The reference code is available here.

The second section of this worksheet will ask you to fill in some design pattern definitions and think about what kind of problems or contexts warrant what kinds of design patterns.

2 Design Pattern Examples

2.1 Observer

Let's consider an `Employee` object that has a `salary` property. We'd like to be able to change their salary and keep the payroll system informed about any modifications. The simplest way to achieve this is passing a reference to payroll and inform it whenever we modify the employee `salary`:

```
class Employee
  attr_reader :name, :title
  attr_reader :salary

  def initialize(name, title, salary, payroll)
    @name = name
    @title = title
    @salary = salary
    @payroll = payroll
  end

  def salary(new_salary)
    @salary = new_salary
    @payroll.update(self)
  end
end
```

2.2 Decorator

Here is an implementation of an object that simply writes a text line to a file.

At some point, we might need to print the line number before each one, or a timestamp or a checksum. We could achieve this by adding new methods to the class that performs exactly what we want, or by creating a new subclass for each use case. However, none of these solutions is optimal.

```
class SimpleWriter
  def initialize(path)
    @file = File.open(path, 'w')
  end

  def write_line(line)
    @file.print(line)
    @file.print("\n")
  end

  def close
    @file.close
  end
end
```

2.3 Factory

Imagine that you are asked to build a simulation of life in a pond that has plenty of ducks. But how would we model our `Pond` if we wanted to have frogs instead of ducks? In the implementation above, we are specifying in the `Pond`'s initializer that it should be filled up with ducks.

```
class Pond
  def initialize(number_ducks)
    @ducks = number_ducks.times.inject([]) do |ducks, i|
      ducks << Duck.new("Duck#{i}")
      ducks
    end
  end

  def simulate_one_day
    @ducks.each {|duck| duck.speak}
    @ducks.each {|duck| duck.eat}
    @ducks.each {|duck| duck.sleep}
  end
end
```

2.4 Singleton

Let's consider the implementation of a logger class. Logging is a feature used across the whole application, so it makes sense that there should only be a single instance of the logger.

```
class SimpleLogger
  attr_accessor :level
  ERROR,WARNING,INFO = 1,2,3

  def initialize
    @log = File.open("log.txt", "w")
```

```
    @level = WARNING
  end

  def error(msg)
    ..
  end

  def warning(msg)
    ..
  end

  def info(msg)
    ..
  end
end
```

3 Problems

3.1 Definitions

Fill in the definitions for the following design patterns.

- Observer:
- Decorator:
- Factory:
- Composite:
- Iterator:
- Visitor:
- Singleton:

3.2 Name that Pattern

For each problem below, you're presented with a situation or system that we'd like to implement. Decide which design pattern is best suited to each context and justify why.

- Armando's theater management site wants to send emails to every patron announcing the new season. However, donors (a type of patron) should get a special email welcoming them to the opening gala.

- Sometimes a theater has to reschedule a show. When that occurs, all tickets should be updated to reflect the new show time.

- The site allows patrons to purchase tickets through Visa. For security reasons, however, the site can only maintain one connection to Visa's purchasing service.

- Sometimes people buy a single ticket, but most of the time people buy tickets in pairs, or for the whole family. The theater wants to be able to print these tickets in the same way.

- The theater recently decided that VIP ticket should, in addition to showing the show name, now also show the director's name to make it more informative for the audience.

- The theater used to write each ticket on a post-it note and give that to the recipient. Having decided that their employees have better things to do than write on post-its, the theater bought a printer for tickets to hide the complexity of actually printing tickets.