CSE 443 Object Oriented Design, Fall 2023

Homework

Last submission date: 25 December 2023

In this homework, you're going to answer the following questions with a software design document. The expected homework has the following requirements:

- The document shall have four chapters for each question in this homework.
- Each answer chapter shall have the following requirements
    - The chapter shall include an introduction section, that will include the following:
        - Your understanding of the problem
        - Required explanations in the question (for example, the problem you should figure out in question 1)
    - The chapter shall include a class diagram including all classes of your solution.
    - The class diagram should include hierarchical boundaries, just like the ones in our reference book
    - The class diagram should include notes containing C++ code, that is necessary for the corresponding design pattern used in the answer. An example C++ note on a class diagram is shown in Appendix A.
    - The chapter shall include a number of sequence diagrams. The number of these diagrams depend on the question. The aim of these diagrams is to show the relevant call sequences in your solution. Mostly, there has to be an initialization (for example, class registrations) and an action sequence (for example, calls to the registered classes)

Questions

1. Here, you can see the class hierarchy of media files. The first design uses polymorphic behavior for the play function so that, when multiple media files need to be iterated through a Media pointer container to play them, this solution is sufficient. Then, you are asked to add new operations to these media files: filter() and export(). You also realize that other operations may need to be added to these two media types.
   Explain the problem with adding these two (and possible future) operations on this design.
   (a) Determine and explain the solution with the correct design pattern (the original pattern from GOF book) and refactor this design, adding the required operations.
   (b) Then refactor your solution with value-based approach.

```
class Media {
public:
  // Constructors, destructor,
  virtual void play() = 0;
private:
  // Private members
};
```

```
class Audio : public Media {
public:
  // Constructors, destructor,
  void play() override {
    // Audio play code
  };
private:
  // Private members
};
class Video : public Media {
public:
  // Constructors, destructor,
  void play() override {
    // Video play code
  };
private:
  // Private members
};
```

2. Consider the Media class hierarchy given in question 1. This time, you are <u>not</u> requested to add new operations on these classes, “play” operation is sufficient.
   However, the play functionality is now expected to be performed on various output devices: For now, these devices are “Speakers” and “Headphones” but may be extended in the future.
   Use bridge design pattern to make this design. Just like the media files from question 1, the output devices should have their own class hierarchy, and the required connection between these two hierarchies must be done using the bridge pattern.

   Hint: You should design Speaker and Headphone classes with output operations.

3. You are asked to implement operations on shapes using an existing class library. Consider only two classes from this library: “LibCircle” and “LibSquare”, as shown below.
   Use external polymorphism design pattern to implement draw and serialize operations on these classes.
   Hint: You should design your own shape hierarchy with required polymorphic operations. Each of your own shape classes should have separate strategies for each polymorphic operation.
   Hint: Your own hierarchy can be easily designed with an abstract base class and a class template derived from this class, whose template parameter is the actual library shape.

```
class LibCircle {
public:
 // Constructors, destructor,
 int32_t getRadius();
 void setRadius(const int32_t);
private:
 // Private members
};
```

```
class LibSquare {
public:
 // Constructors, destructor,
 int32_t getSide();
 void setSide(const int32_t);
private:
 // Private members
};
```

4. You have a weather monitoring system that is fed by several sensors around the region. The class model of the system is shown below. Each of these sensors provide its data through the corresponding set method.
   Now, you are required to add functionality that need the sensor readings. The first one is the DisplayStation as shown below, and another is LogStation that logs the readings when they change. You make a design decision that further stations may need to be added in the future. Modify/extend the original design so that when the WMSystem is updated, corresponding stations are fed with the updated data. Use the correct design pattern from the GOF book so that the updated version of the WMSystem will not need to change when a third station is added to the system.

```
class WMSystem {
public:
 // Constructors, destructor,
 void setTemperature(float temp);
 void setAirPressur(float pres);
 void setWind(float dir, float speed);
private:
 // Private members
};
```

```
class DisplayStation {
public:
 // Constructors, destructor,
 void displayTemperature(float temp);
 void displayPressure(float pres);
 void displayWind(float dir, float speed);
private:
 // Private members
};
```

Appendix: Guidelines for Presenting Solutions with UML Class Diagrams

You can see an example UML representation of adapter design pattern below. Showing your solutions with UML class diagram and notes containing C++ code should be done similarly: