МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: логирование, перегрузка операций

Студент гр.0382 _____ Литягин С.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы.

Изучить применение логгеров, изучение перегрузки оператора вывода в поток; написание логгеров нескольких типов, для отслеживания изменений в состоянии объекта.

Задание.

Необходимо проводить логирование того, что происходит во время игры.

Требования:

- Реализован класс логгера, который будет получать объект, который необходимо отслеживать, и при изменении его состоянии записывать данную информацию.
- Должна быть возможность записывания логов в файл, в консоль или одновременно в файл и консоль.
- Должна быть возможность выбрать типа вывода логов
- Все объекты должны логироваться через перегруженный оператор вывода в поток.
- Должна соблюдаться идиома RAII

Потенциальные паттерны проектирования, которые можно использовать:

- *Адаптер (Adapter) - преобразование данных к нужному формату логирования*
- *Декоратор (Decorator) - форматирование текста для логирования*
- *Мост (Bridge) - переключение между логированием в файл/консоль*
- *Наблюдатель (Observer) - отслеживание объектов, которые необходимо логировать*
- *Синглтон (Singleton) - гарантия логирования в одно место через одну сущность*
- *Заместитель (Proxy) - подстановка и выбор необходимого логирования*

Выполнение работы.

В ходе работы были использованы паттерны: Наблюдатель, Одиночка и мост. Паттерн Наблюдатель был нужен, чтобы создать механизм, благодаря которому класс логгера сможешь следить и реагировать на события в классах объектов. Паттерн одиночка был применен, чтобы гарантировать логгирование через одну сущность в одно место. Паттерн мост был применен, чтобы дать возможность переключать разные типы логгеров.

Были созданы или изменены следующие классы:

1. Был создан класс-интерфейс Observer, имеющий метод:

- виртуальный метод Update() для вывода информации об изменениях состояния объекта (метод реализуется в классе-наследнике)

2. Был создан класс субъекта Subject (субъекты и будут отслеживаться логгером). Имеет поле Observer* observer (т.е. субъект может хранить указатель на наблюдателя). Объявлены следующие методы:

- метод SetObs(Observer* obs) для установки указателя наблюдателя в поле субъекта observer
- метод Notify() вызывает метод Update() наблюдателя для обновления информации о его состоянии, если у субъекта установлен указатель на наблюдателя, иначе в консоль выведется сообщение, что наблюдатель у объекта не установлен
- виртуальный метод GetLog() для получения строки с информацией о состоянии субъекта (метод реализуется в классах-наследниках)
- дружественная функция-перегрузка оператора вывода в поток operator<<(std::ostream &out, Subject* sub); теперь при выводе в поток субъекта будет вызван вывод в поток строки с состоянием субъекта, т.е. результат метода GetLog()

3. Чтобы логгер мог работать с субъектами, они должны быть. В нашей игре мы будем отслеживать состояния игрока, противников и вещей. Все они являются наследниками интерфейса Object. Делаем так, чтобы

класс Object наследовался от класса субъектов Subject. Соответственно, в классах Hero, Ent, Eye, Spider, Axe, Coin, Candy был добавлен метод:

- переопределенный метод GetLog(), который возвращает строку с информацией о состоянии объекта

4. Был создан класс-интерфейс LoggerImplication, имеющий метод:

- виртуальный метод Out(Subject* sub) для вывода объекта в поток (метод реализуется в классах-наследниках)

5. Был создан класс FileLogger, наследуемый от LoggerImplication. Имеет поле std::ofstream output для хранения потока для файлового вывода. Имеет метод:

- переопределенный метод Out(Subject* sub) для вывода в поток output объект (поскольку оператор вывода в потом перегружен, то будет выведена информация об объекте)

6. Был создан класс ConsoleLogger, наследуемый от LoggerImplication. Имеет метод:

- переопределенный метод Out(Subject* sub) для вывода в поток стандартный поток (т.е. в консоль) объект (поскольку оператор вывода в потом перегружен, то будет выведена информация об объекте)

7. Был создан класс ConsFileLog, наследуемый от LoggerImplication. Имеет поле std::ofstream output для хранение потока для файлового вывода. Имеет метод:

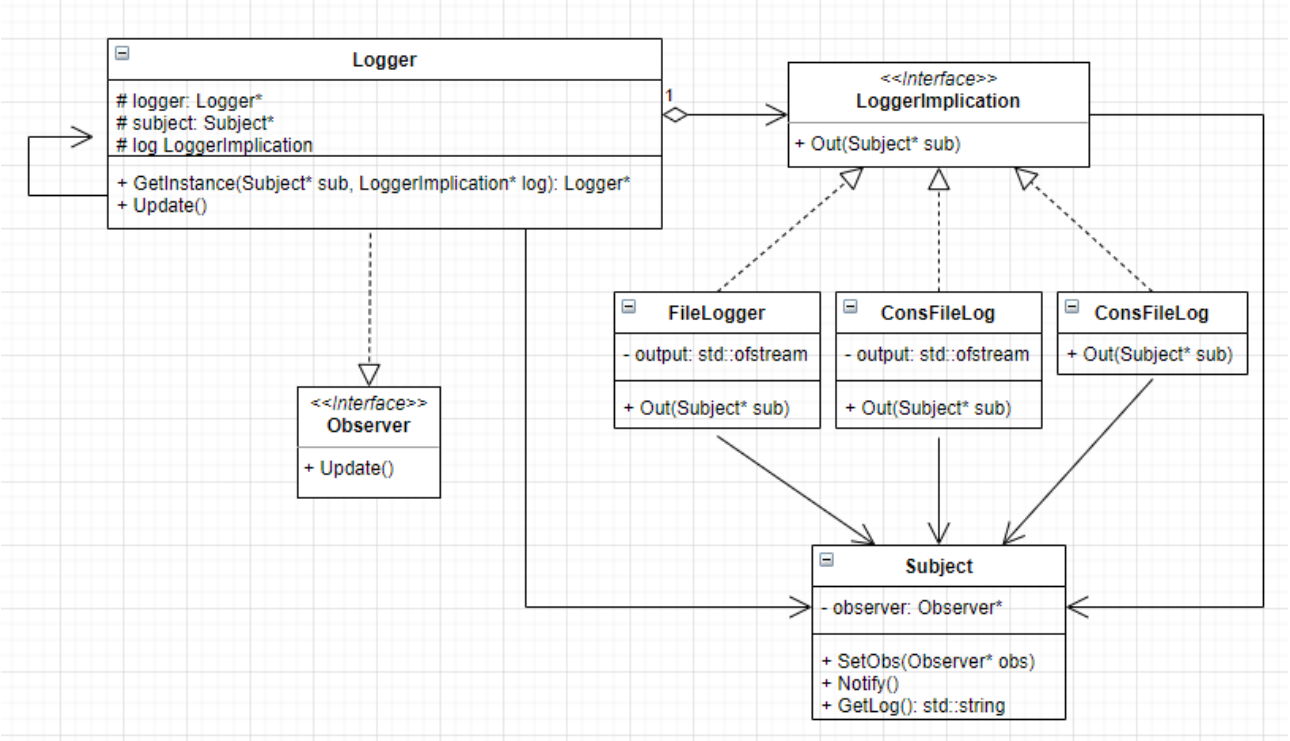
- переопределенный метод Out(Subject* sub) для вывода в поток output и стандартный поток (т.е. в консоль) объект (поскольку оператор вывода в потом перегружен, то будет выведена информация об объекте)

8. Был создан класс-одиночка Logger, наследуемый от Observer. Имеет поля static Logger* logger, Subject* subject, LoggerImplication* log. Имеет следующие методы:

- переопределенный метод Update() для вывода информации об изменениях состояния объекта; для этого вызывается метод Out(subject) у объекта log (т.е. происходит вывод информации через определенный тип логгера)
- метод GetInstance(Subject* sub, LoggerImplication* log), который возвращает единственный экземпляр своего класса. Это нужно, поскольку конструктор класса-одиночки скрыт от пользователя, а экземпляр нужен. Собственно, проверяем, если logger имеет нулевой указатель, то передаем в поле logger новый объект Logger(sub, log), а затем устанавливаем субъекту наблюдателя с помощью метода SetObs(logger)

UML-диаграмма классов представлена на рис. 1.

Рисунок 1 – UML-диаграмма классов.



Выводы.

В ходе работы было изучено применение логгеров и изучение перегрузки оператора вывода в поток; был написан логгер с возможностью вывода в

консоль/в файл/в консоль и файл для отслеживания изменений в состоянии объекта.