Лабораторная работа №5
Разработка системы со связью вида «интерфейс/реализация»

Теоретические сведения

Виртуальная функция – это функция, обозначенная ключевым словом *virtual*, для которой предполагается переопределение в производных классов, но не гарантируется. В случае, если в производном классе отсутствует реализация виртуальной функции, будет использовано определение функции родительского класса.

Пример:

```
class Example{
public:
    virtual void func();
}
```

Чистая виртуальная функция – это виртуальная функция, для которой отсутствует определение в базовом классе. Синтаксически это выражается следующим образом:

```
class Example{
public:
    virtual void func() = 0;
}
```

Класс, содержащий чистую виртуальную функцию, называется *абстрактным*. Абстрактный класс не может иметь своих экземпляров (объектов). Однако, можно создавать указатели на это класс, но хранить под ними объекты его наследников. Нужно учитывать, что в таком случае будут доступны только те функции производного класса, которые были объявлены в базовом абстрактном классе.

Класс «интерфейс» – абстрактный класс, не содержащий никаких полей, или реализованных функций. Такой класс работает исключительно как «договор» о том, какие функции необходимы в реализациях предков, позволяя абстрагироваться от конечных классов. Используя такие классы как прослойки понижается связность системы, что улучшает ее расширяемость и модульность. (Пример выше подходит под описание класса «интерфейса»)

Парадигма вида «интерфейс/реализация» отражается путем использования классов «интерфейсов» для прослойки между «исполнителем» и «заказчиком». Синтаксически это выражено следующим образом:

- Описывается некоторый интерфейсный класс, который отражает определенное действие, например, «дает молоко».
- Описывается некоторый потребитель, которому требуется результат этого действия, например, «веселый молочник». Внутри себя он содержит указатель на интерфейс «дает молоко» (или на массив указателей).

- Каким-либо образом (через конструктор/через сеттер) в этот указатель записывается наследник класса «дает молоко», например, «корова», «коза», что позволяет классу «веселый молочник» использовать функции «дает молоко» у классов «корова» или «коза» в отрыве от того, как именно они реализованы.

Такая система позволяет избегать дублирования функционала «веселого молочника» и создания классов «козий веселый молочник» и «коровий веселый молочник» благодаря слабому связыванию классов.

Некоторые сложности с синтезом таких классов можно избежать, осмыслив, что чаще всего класс «интерфейс» – это *абстракция действий*, а не *абстракция сущностей*.

Интерфейс предоставляет услуги для конечных пользователей. В наилучшем случае конечным пользователям предоставляются только те услуги, которые им необходимы (*см. Interface Segregation*)

Детали реализации скрыты от пользователей. Нужно помнить: изменения в реализации не должны требовать внесения изменений в пользовательский код. Если интерфейс не терпит изменений, то пользователям все равно, изменится ли реализация. Главное, чтобы реализация корректно выполняла и возвращала требуемые и объявленные интерфейсом значения.

Абстрактное электронное письмо от «заказчика»:

«Необходимо описать следующие классы электроприборов:

- фонарик,
- лампочка,
- электроутюг.

Необходимо описать следующие классы светящихся предметов:

- керосиновая лампа,
- свечка,
- фонарик.

Также нужно описать несколько контейнеров с дополнительным функционалом:

- сетевой фильтр – в него можно подключать электроприборы и «заряжать» их,
- гирлянда – в нее можно подключать все, что светится и по нажатию кнопочки на гирлянде все подключенные в нее светящиеся объекты должны попытаться выдавать свет.

Важно учесть, что электроприборы могут светиться, только если заряжены.»

Дополнительное письмо заказчика

«Вот еще что: розетки выдают ограниченное число ватт, а электроприборы их потребляют. В случае, если число ватт потребления превышает число ватт питания – фильтр выпадает в защиту и перестает подавать электричество»

Задание на лабораторную работу

1. Прочитать «сообщение заказчика» и определить необходимые для решения задачи классы. Можно выполнить без дополнительной части, но тогда конечная оценка будет ниже.
2. Нарисовать UML диаграмму классов (Class Diagram), с отношением между ними.
3. Реализовать заданную систему на языке C++. Классы описать при помощи разделения объявления и реализации на hpp и cpp файлы.
4. Отразить возможности модели в main.
5. Отобразить работу функции main на UML диаграмме последовательности действий (Sequence Diagram)
6. Написать CMakeLists.txt для сборки исполняемого файла.

Источники к ознакомлению

1. https://itproger.com/spravka/cpp/abstract#:~:text=C%2B%2B%20abstract-,%D0%A7%D1%82%D0%BE%20%D1%82%D0%B0%D0%BA%D0%BE%D0%B5%20abstract%20%D0%B2%20C%2B%2B%3F,%D0%B5%D1%91%20%D0%BD%D0%B0%20%D1%83%D1%81%D0%BC%D0%BE%D1%82%D1%80%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%BD%D1%8B%D1%85%20%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2.
2. https://metanit.com/cpp/tutorial/5.12.php
3. https://www.c-cpp.ru/books/chisto-virtualnye-funkcii-i-abstraktnye-tipy
4. Вайсфельд М. Объектно-ориентированное мышление. — СПб.: Питер, 2014. — 304 с.