Конфигурирование класса Window с помощью WindowImp

Важнейший вопрос, который мы еще не рассмотрели, – как сконфигурировать окно с помощью подходящего подкласса `WindowImp`. Другими словами, когда инициализируется переменная `_imp` и как узнать, какая оконная система (следовательно, и подкласс `WindowImp`) используется? Ведь окну необходим объект `WindowImp`.

Тут есть несколько возможностей, но мы остановимся на той, где используется паттерн **абстрактная фабрика**. Можно определить абстрактный фабричный класс `WindowSystemFactory`, предоставляющий интерфейс для создания различных видов системно-зависимых объектов:

```
class WindowSystemFactory {
public:
    virtual WindowImp* CreateWindowImp() = 0;
    virtual ColorImp* CreateColorImp() = 0;
    virtual FontImp* CreateFontImp() = 0;

    // операции "Create..." для всех видов ресурсов оконной системы
};
```

Далее разумно определить конкретную фабрику для каждой оконной системы:

```
class PMWindowSystemFactory : public WindowSystemFactory {
    virtual WindowImp* CreateWindowImp()
        { return new PMWindowImp; }
    // ...
};

class XWindowSystemFactory : public WindowSystemFactory {
    virtual WindowImp* CreateWindowImp()
        { return new XWindowImp; }
    // ...
};
```

Чтобы инициализировать член `_imp` указателем на объект `WindowImp`, соответствующий данной оконной системе, конструктор базового класса `Window` может использовать интерфейс `WindowSystemFactory`:

```
Window::Window () {
    _imp = windowSystemFactory->CreateWindowImp();
}
```

Переменная `windowSystemFactory` – это известный программе экземпляр подкласса `WindowSystemFactory`. Она, аналогично переменной `guiFactory`, определяет внешний облик. И инициализировать `windowSystemFactory` можно точно так же.

Паттерн мост

Класс `WindowImp` определяет интерфейс к общим средствам оконной системы, но на его дизайн накладываются иные ограничения, нежели на интерфейс

класса Window. Прикладной программист не обращается к интерфейсу WindowImp непосредственно, он имеет дело только с объектами класса Window. Поэтому интерфейс WindowImp необязательно должен соответствовать представлению программиста о мире, как то было в случае с иерархией и интерфейсом класса Window. Интерфейс WindowImp может более точно отражать сущности, которые в действительности предоставляют оконные системы, со всеми их особенностями. Он может быть ближе к идее пересечения или объединения функциональности – в зависимости от требований к целевой оконной системе.

Важно понимать, что интерфейс класса Window призван обслуживать интересы прикладного программиста, тогда как интерфейс класса WindowImp в большей степени ориентирован на оконные системы. Разделение функциональности окон между иерархиями Window и WindowImp позволяет нам независимо реализовывать и специализировать их интерфейсы. Объекты из этих иерархий взаимодействуют, позволяя Lexi работать без изменений в нескольких оконных системах.

Отношение иерархий Window и WindowImp являет собой пример паттерна мост. Идея его создания заключалась в том, чтобы предоставить возможность совместной работы отдельным иерархиям классов, даже в случае их раздельного эволюционирования. Критерии разработки, которыми мы руководствовались, заставили нас создать две различные иерархии классов: одну, поддерживающую логическое понятие окон, и другую для хранения промежуточных вариантов окон. Паттерн мост позволяет нам сохранять и совершенствовать наши логические абстракции управления окнами без необходимости привлечения программно-зависимого кода и наоборот.

2.7. Операции пользователя

Часть функциональности Lexi доступна через WYSIWYG-представление документа. Вы вводите и удаляете текст, перемещаете точку вставки и выбираете участки текста, просто указывая и щелкая мышью или нажимая клавиши. Другая часть функциональности доступна через выпадающие меню, кнопки и клавиши-ускорители. К этой категории относятся такие операции:

- создание нового документа;
- открытие, сохранение и печать существующего документа;
- вырезание выбранной части документа и вставка ее в другое место;
- изменение шрифта и стиля выбранного текста;
- изменение форматирования текста, например, установка режима выравнивания;
- завершение приложения и др.

Lexi предоставляет для этих операций различные пользовательские интерфейсы. Но мы не хотим ассоциировать конкретную операцию с определенным пользовательским интерфейсом, поскольку для выполнения одной и той же операции желательно иметь несколько интерфейсов (например, листать страницы можно с помощью кнопки или выбора из меню). Кроме того, в будущем может понадобиться изменить интерфейс.