Lexi, создать унифицированную, интуитивно понятную абстракцию окна, не зависящую от оконной системы конкретного поставщика.

Итак, мы определили оконный интерфейс, с которым будет работать Lexi. Но где же в нем место для реальной платформенно-зависимой оконной системы? Если мы не собираемся реализовывать собственную оконную систему, то в каком-то месте наша абстракция окна должна быть выражена в терминах целевой системы. Но где именно?

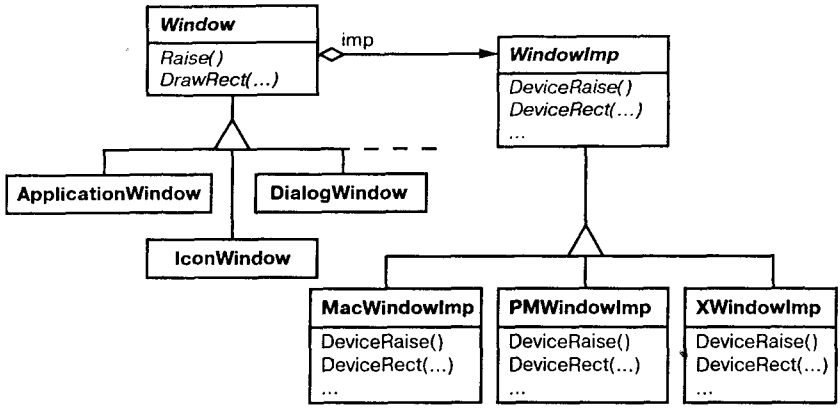
Можно было бы реализовать несколько вариантов класса `Window` и его подклассов – по одному для каждой оконной среды. Выбор нужного варианта производится при сборке Lexi для данной платформы. Но представьте себе, с чем вы столкнетесь при сопровождении, если придется отслеживать множество разных классов с одним и тем же именем `Window`, но реализованных для разных оконных систем. Вместо этого мы могли бы создать зависящие от реализации подклассы каждого класса в иерархии `Window`, но закончилось бы это тем же самым комбинаторным ростом числа классов, о котором уже говорилось при попытке добавить элементы оформления. Кроме того, оба решения недостаточно гибки, чтобы можно было перейти на другую оконную систему уже после компиляции программы. Поэтому придется поддерживать несколько разных исполняемых файлов.

Ни тот, ни другой вариант не выглядят привлекательно, но что еще можно сделать? То же самое, что мы сделали для форматирования и декорирования, – *инкапсулировать изменяющуюся сущность*. В этом случае изменчивой частью является реализация оконной системы. Если инкапсулировать функциональность оконной системы в объекте, то удастся реализовать свой класс `Window` и его подклассы в терминах интерфейса этого объекта. Более того, если такой интерфейс сможет поддержать все интересующие нас оконные системы, то не придется изменять ни `Window`, ни его подклассы при переходе на другую систему. Мы сконфигурируем оконные объекты в соответствии с требованиями нужной оконной системы, просто передав им подходящий объект, инкапсулирующий оконную систему. Это можно сделать даже во время выполнения.

Классы Window и WindowImp

Мы определим отдельную иерархию классов `WindowImp`, в которой скроем знание о различных реализациях оконных систем. `WindowImp` – это абстрактный класс для объектов, инкапсулирующих системно-зависимый код. Чтобы заставить Lexi работать в конкретной оконной системе, каждый оконный объект будем конфигурировать экземпляром того подкласса `WindowImp`, который предназначен для этой системы. На диаграмме ниже представлены отношения между иерархиями `Window` и `WindowImp`:

Скрыв реализацию в классах `WindowImp`, мы сумели избежать «засорения» классов `Window` зависимостями от оконной системы. В результате иерархия `Window` получается сравнительно компактной и стабильной. В то же время мы можем расширить иерархию реализаций, если будет нужно поддержать новую оконную систему.



Подклассы WindowImp

Подклассы `WindowImp` преобразуют запросы в операции, характерные для конкретной оконной системы. Рассмотрим пример из раздела 2.2. Мы определили `Rectangle::Draw` в терминах операции `DrawRect` над экземпляром класса `Window`:

```
void Rectangle::Draw (Window* w) {
    w->DrawRect(_x0, _y0, _x1, _y1);
}
```

В реализации `DrawRect` по умолчанию используется абстрактная операция рисования прямоугольников, объявленная в `WindowImp`:

```
void Window::DrawRect (
    Coord x0, Coord y0, Coord x1, Coord y1
) {
    _imp->DeviceRect(x0, y0, x1, y1);
}
```

где `_imp` – переменная-член класса `Window`, в которой хранится указатель на объект `WindowImp`, использованный при конфигурировании `Window`. Реализация окна определяется тем экземпляром подкласса `WindowImp`, на который указывает `_imp`. Для `XWindowImp` (то есть подкласса `WindowImp` для оконной системы X Window System) реализация `DeviceRect` могла бы выглядеть так:

```
void XWindowImp::DeviceRect (
    Coord x0, Coord y0, Coord x1, Coord y1
) {
    int x = round(min(x0, x1));
    int y = round(min(y0, y1));
    int w = round(abs(x0 - x1));
    int h = round(abs(y0 - y1));
    XDrawRectangle(_dpy, _winid, _gc, x, y, w, h);
}
```