Есть и другие способы выбора фабрики во время выполнения. Например, можно было бы вести реестр, в котором символьные строки отображаются на объекты фабрик. Это позволяет зарегистрировать экземпляр новой фабрики, не меняя существующий код, как то требуется при предыдущем подходе. И нет нужды связывать с приложением код фабрик для всех платформ. Это существенно, поскольку связать код для `MotifFactory` с приложением, работающим на платформе, где Motif не поддерживается, может оказаться невозможным.

Важно, впрочем, лишь то, что после конфигурации приложения для работы с конкретной фабрикой объектов, мы получаем нужный внешний облик. Если впоследствии мы изменим решение, то сможем инициализировать `guiFactory` по-другому, чтобы изменить внешний облик, а затем динамически перестроим интерфейс.

Паттерн абстрактная фабрика

Фабрики и их продукция – вот ключевые участники паттерна абстрактная фабрика. Этот паттерн может создавать семейства объектов, не инстанцируя классы явно. Применять его лучше всего, когда число и общий вид изготавливаемых объектов остаются постоянными, но между конкретными семействами продуктов имеются различия. Выбор того или иного семейства осуществляется путем инстанцирования конкретной фабрики, после чего она используется для создания всех объектов. Подставив вместо одной фабрики другую, мы можем заменить все семейство объектов целиком. В паттерне абстрактная фабрика акцент делается на создании *семейств* объектов, и это отличает его от других порождающих паттернов, создающих только один какой-то вид объектов.

2.6. Поддержка нескольких оконных систем

Как должно выглядеть приложение – это лишь один из многих вопросов, встающих при переносе приложения на новую платформу. Еще одна проблема из той же серии – оконная среда, в которой работает Lexi. Данная среда создает иллюзию наличия нескольких перекрывающихся окон на одном растровом дисплее. Она распределяет между окнами площадь экрана и направляет им события клавиатуры и мыши. Сегодня существует несколько широко распространенных и во многом не совместимых между собой оконных систем (например, Macintosh, Presentation Manager, Windows, X). Мы хотели бы, чтобы Lexi работал в любой оконной среде по тем же причинам, по которым мы поддерживаем несколько стандартов внешнего облика.

Можно ли воспользоваться абстрактной фабрикой?

На первый взгляд представляется, что перед нами еще одна возможность применить паттерн абстрактная фабрика. Но ограничения, связанные с переносом на другие оконные системы, существенно отличаются от тех, что накладывают независимость от внешнего облика.

Применяя паттерн абстрактная фабрика, мы предполагали, что удастся определить конкретный класс глифов-виджетов для каждого стандарта внешнего облика. Это означало, что можно будет произвести конкретный класс для данного

стандарта (например, MotifScrollBar и MacScrollBar) от абстрактного класса (допустим, ScrollBar). Предположим, однако, что у нас уже есть несколько иерархий классов, полученных от разных поставщиков, – по одной для каждого стандарта. Маловероятно, что данные иерархии будут совместимы между собой. Поэтому отсутствуют общие абстрактные изготавливаемые классы для каждого вида виджетов (ScrollBar, Button, Menu и т.д.). А без них фабрика классов работать не может. Необходимо, чтобы иерархии виджетов имели единый набор абстрактных интерфейсов. Только тогда удастся правильно объявить операции Create... в интерфейсе абстрактной фабрики.

Для виджетов мы решили эту проблему, разработав собственные абстрактные и конкретные изготавливаемые классы. Теперь сталкиваемся с аналогичной трудностью при попытке заставить Lexi работать во всех существующих оконных средах. Именно разные среды имеют несовместимые интерфейсы программирования. Но на этот раз все сложнее, поскольку мы не можем себе позволить реализовать собственную нестандартную оконную систему.

Однако спасительный выход все же есть. Как и стандарты на внешний облик, интерфейсы оконных систем не так уж радикально отличаются друг от друга, ибо все они предназначены примерно для одних и тех же целей. Нам нужен унифицированный набор оконных абстракций, которым можно закрыть любую конкретную реализацию оконной системы.

Инкапсуляция зависимостей от реализации

В разделе 2.2 мы ввели класс Window для отображения на экране глифа или структуры, состоящей из глифов. Ничего не говорилось о том, с какой оконной системой работает этот объект, поскольку в действительности он вообще не связан ни с одной системой. Класс Window инкапсулирует понятие окна в любой оконной системе:

- операции для отрисовки базовых геометрических фигур;
- возможность свернуть и развернуть окно;

Таблица 2.3. Интерфейс класса Window

Обязанность	Операции

управление окнами	`virtual void Redraw()`
	`virtual void Raise()`
	`virtual void Lower()`
	`virtual void Iconify()`
	`virtual void Deiconify()`
	`...`

графика	`virtual void DrawLine(...)`
	`virtual void DrawRect(...)`
	`virtual void DrawPolygon(...)`
	`virtual void DrawText(...)`
	`...`