Инкапсуляция алгоритма форматирования

С учетом всех ограничений и деталей процесс форматирования с трудом поддается автоматизации. К этой проблеме есть много подходов, и у разных алгоритмов форматирования имеются свои сильные и слабые стороны. Поскольку Lexi – это WYSIWYG-редактор, важно соблюдать компромисс между качеством и скоростью форматирования. В общем случае желательно, чтобы редактор реагировал достаточно быстро и при этом внешний вид документа оставался приемлемым. На достижение этого компромисса влияет много факторов, и не все из них удастся установить на этапе компиляции. Например, можно предположить, что пользователь готов смириться с замедленной реакцией в обмен на лучшее качество форматирования. При таком предположении нужно было бы применять совершенно другой алгоритм форматирования. Есть также компромисс между временем и памятью, скорее, имеющий отношение к реализации: время форматирования можно уменьшить, если хранить в памяти больше информации.

Поскольку алгоритмы форматирования обычно оказываются весьма сложными, желательно, чтобы они были достаточно замкнутыми, а еще лучше – полностью независимыми от структуры документа. Оптимальный вариант – добавление нового вида глифа вовсе не затрагивает алгоритм форматирования. С другой стороны, при добавлении нового алгоритма форматирования не должно возникать необходимости в модификации существующих глифов.

Учитывая все вышесказанное, мы должны постараться спроектировать Lexi так, чтобы алгоритм форматирования легко можно было заменить, по крайней мере, на этапе компиляции, если уж не во время выполнения. Допустимо изолировать алгоритм и одновременно сделать его легко замещаемым путем инкапсулирования в объекте. Точнее, мы определим отдельную иерархию классов для объектов, инкапсулирующих алгоритмы форматирования. Для корневого класса иерархии определим интерфейс, который поддерживает широкий спектр алгоритмов, а каждый подкласс будет реализовывать этот интерфейс в виде конкретного алгоритма форматирования. Тогда удастся ввести подкласс класса `Glyph`, который будет автоматически структурировать своих потомков с помощью переданного ему объекта-алгоритма.

Классы Compositor и Composition

Мы определим класс `Compositor` (композитор) для объектов, которые могут инкапсулировать алгоритм форматирования. Интерфейс (см. табл. 2.2) позволяет объекту этого класса узнать, *какие* глифы надо форматировать и *когда*. Форматируемые композитором глифы являются потомками специального подкласса класса `Glyph`, который называется `Composition` (композиция). Композиция при создании получает объект некоторого подкласса `Compositor` (специализированный для конкретного алгоритма разбиения на строки) и в нужные моменты предписывает композитору форматировать глифы, по мере того как пользователь изменяет документ. На рис. 2.5 изображены отношения между классами `Composition` и `Compositor`.