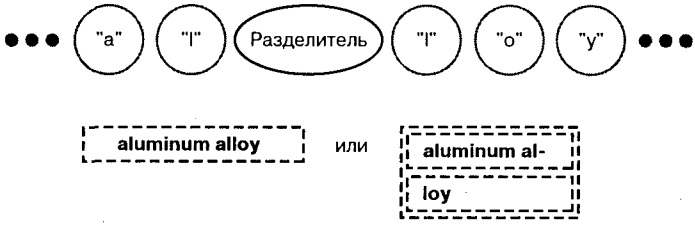
(discretionary) глиф. Разделяющие глифы являются экземплярами подкласса `Glyph` – класса `Discretionary`.

Разделяющий глиф может выглядеть по-разному в зависимости от того, является он последним символом в строке или нет. Если это последний символ, глиф выглядит как дефис, в противном случае не отображается вообще. Разделяющий глиф запрашивает у своего родителя (объекта `Row`), является ли он последним потомком, и делает это всякий раз, когда от него требуют отобразить себя или вычислить свои размеры. Стратегия форматирования трактует разделяющие глифы точно так же, как пропуски, считая их «кандидатами» на завершающий символ строки. На диаграмме ниже показано, как может выглядеть встроенный разделитель.



Паттерн посетитель

Вышеописанная процедура – пример применения паттерна посетитель. Его главными участниками являются класс `Visitor` и его подклассы. Паттерн посетитель абстрагирует метод, позволяющий иметь заранее неопределенное число видов анализа структур глифов без изменения самих классов глифов. Еще одна полезная особенность посетителей состоит в том, что их можно применять не только к таким агрегатам, как наши структуры глифов, но и к любым структурам, состоящим из объектов. Сюда входят множества, списки и даже направленные ациклические графы. Более того, классы, которые обходит посетитель, необязательно должны быть связаны друг с другом через общий родительский класс. А это значит, что посетители могут пересекать границы иерархий классов.

Важный вопрос, который надо задать себе перед применением паттерна посетитель, звучит так: «Какие иерархии классов наиболее часто будут изменяться?» Этот паттерн особенно удобен, если необходимо выполнять действия над объектами, принадлежащими классу со стабильной структурой. Добавление нового вида посетителя не требует изменять структуру класса, что особенно важно, когда класс большой. Но при каждом добавлении нового подкласса вы будете вынуждены обновить все интерфейсы посетителя с целью включить операцию `Visit...` для этого подкласса. В нашем примере это означает, что добавление подкласса `Foo` класса `Glyph` потребует изменить класс `Visitor` и все его подклассы, чтобы добавить операцию `VisitFoo`. Однако при наших проектных условиях гораздо более вероятно добавление к Lexi нового вида анализа, а не нового вида глифов. Поэтому для наших целей паттерн посетитель вполне подходит.