Нам не хватает механизма параметризации пунктов меню запросами, которые они должны выполнять. Таким способом удалось бы избежать разрастания числа подклассов и обеспечить большую гибкость во время выполнения. Параметризовать `MenuItem` можно вызываемой функцией, но это решение неполно по трем причинам:

- в нем не учитывается проблема отмены/повтора;
- с функцией трудно ассоциировать состояние. Например, функция, изменяющая шрифт, должна «знать», какой именно это шрифт;
- функции с трудом поддаются расширению, а использовать их части тоже затруднительно.

Поэтому лучше параметризовать пункты меню *объектом*, а не функцией. Тогда мы сможем прибегнуть к механизму наследования для расширения и повторного использования реализации запроса. Кроме того, у нас появляется место для сохранения состояния и возможность реализовать отмену и повтор. Вот еще один пример инкапсуляции изменяющейся сущности, в данном случае – запроса. Каждый запрос мы инкапсулируем в объект-команду.

## Класс Command и его подклассы

Сначала определим абстрактный класс `Command`, который будет предоставлять интерфейс для выдачи запроса. Базовый интерфейс включает всего одну абстрактную операцию `Execute`. Подклассы `Command` по-разному реализуют эту операцию для выполнения запросов. Некоторые подклассы могут частично или полностью делегировать работу другим объектам, а остальные выполняют запрос сами (см. рис. 2.11). Однако для запрашивающего объект `Command` – это всего лишь объект `Command`, все они рассматриваются одинаково.

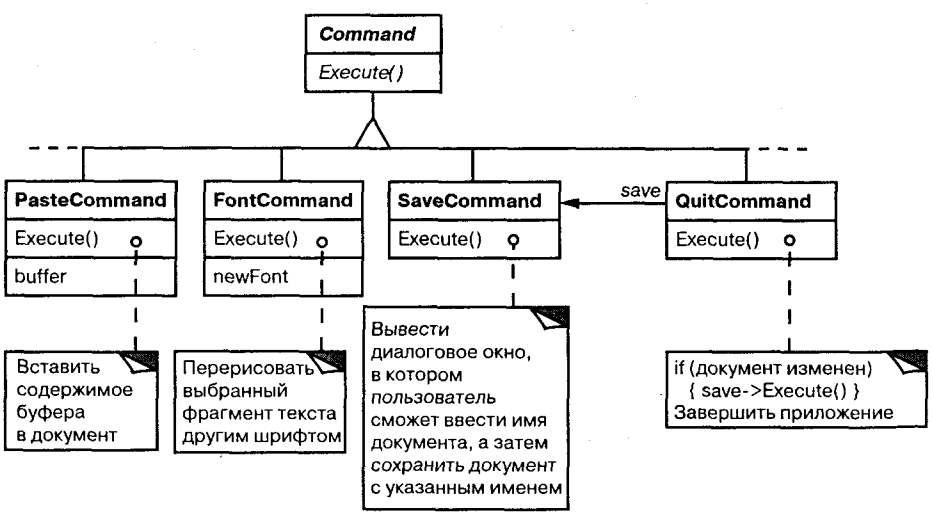


*Рис. 2.11. Часть иерархии класса Command*

Теперь в классе MenuItem может храниться объект, инкапсулирующий запрос (рис. 2.12). Каждому объекту, представляющему пункт меню, мы передаем экземпляр того из подклассов Command, который соответствует этому пункту, точно так же, как мы задаем текст, отображаемый в пункте меню. Когда пользователь выбирает некоторый пункт меню, объект MenuItem просто вызывает операцию Execute для своего объекта Command, тем самым предлагая ему выполнить запрос. Заметим, что кнопки и другие виджеты могут пользоваться объектами Command точно так же, как и пункты меню.

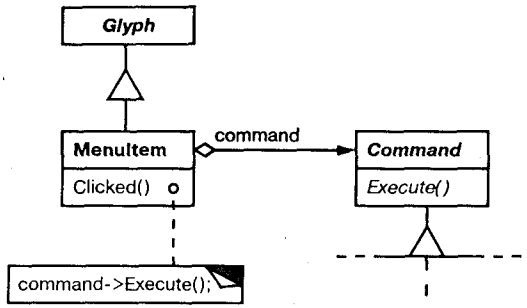


*Рис. 2.12. Отношение между классами MenuItem и Command*

## Отмена операций

Для того чтобы отменить или повторить команду, нужна операция Unexecute в интерфейсе класса Command. Ее выполнение отменяет все, что было сделано предыдущей командой Execute. При этом используется информация, сохраненная операцией Execute. Так, при команде FontCommand операция Execute была бы должна сохранить координаты участка текста, шрифт которого изменялся, а равно и первоначальный шрифт (или шрифты). Операция Unexecute класса FontCommand должна была бы восстановить старый шрифт (или шрифты) для этого участка текста.

Иногда определять, можно ли осуществить отмену, необходимо во время выполнения. Скажем, запрос на изменение шрифта выделенного участка текста не производит никаких действий, если текст уже отображен требуемым шрифтом. Предположим, что пользователь выбрал некий текст и решил изменить его шрифт на случайно выбранный. Что произойдет в результате последующего запроса на отмену? Должно ли бессмысленное изменение приводить к столь же бессмысленной отмене? Наверное, нет. Если пользователь повторит случайное изменение шрифта несколько раз, то не следует заставлять его выполнять точно такое же число отмен, чтобы вернуться к последнему осмысленному состоянию. Если суммарный эффект выполнения последовательности команд нулевой, то нет необходимости вообще делать что-либо при запросе на отмену.

Для определения того, можно ли отменить действие команды, мы добавим к интерфейсу класса Command абстрактную операцию Reversible (обратимая), которая возвращает булево значение. Подклассы могут переопределить эту операцию и возвращать true или false в зависимости от критерия, вычисляемого во время выполнения.