# Часть 2. Порождающие паттерны

В большинстве объектно-ориентированных языков программирования за конструирование объекта отвечает конструктор. В самом простом случае, клиент знает, какого типа объект ему требуется и создает его путем вызова соответствующего конструктора. Но в некоторых случаях, тип объекта может быть неизвестен вызывающему коду, или процесс конструирования может быть настолько сложным, что использование конструктора будет неудобным или невозможным. Порождающие паттерны предназначены для решения типовых проблем создания объектов.

В некоторых случаях, разработчик хочет гарантировать, что будет создан лишь один экземпляр некоторого класса. Иногда, такая потребность диктуется спецификацией системы, в которой сказано, что в системе должен быть лишь один считыватель мыслей пользователя. Но гораздо чаще такой подход применяется за счет того, что он обеспечивает глобальную точку доступа к некоторому объекту. Это позволяет любому объекту получить доступ к любой точке системы, что избавляет разработчика от необходимости продумывать обязанности классов и выделять их зависимости.

Класс, который гарантирует создание лишь одного экземпляра и предоставляет к нему глобальную точку, известен как паттерн "Синглтон". Это самый знаменитый паттерн проектирования, недостатки которого ставят под сомнение его полезность. Использование глобальных объектов с изменяемым состоянием приводит к "эффекту бабочки", когда поведение системы начинает зависеть от порядка вызова методов в разных модулях, и практически не поддается тестированию и сопровождению.

Но не все порождающие поведения столь сомнительны.

Возрастающая сложность системы часто приводит к необходимости изоляции процесса создания объектов. Бывает полезным скрыть от вызывающего кода конкретный тип создаваемого объекта, чтобы иметь возможность изменить его в будущем. В других случаях, тип создаваемого объекта зависит от аргументов метода, что также делает невозможным использование конструкторов. А иногда процесс конструирования должен контролироваться наследником, что приводит к появлению "стратегии" создания. Так мы приходим к второму по популярности паттерну проектирования - "фабрикам".

Существует несколько разновидностей фабрик. *Абстрактная фабрика* предназначена для создания семейства объектов и позволяет заменять это семейство путем использования нужного подкласса фабрики. Статический *Фабричный метод* представляет собой статический метод, который возвращает экземпляр конкретного или полиморфного класса, в зависимости от аргументов метода или конфигурации. Классический фабричный метод является стратегией конструирования объектов и обеспечивает гибкость за счет полиморфного использования.

Фабрики прекрасно справляются с инкапсуляцией процесса создания и обеспечивают гибкость за счет наследования. Но иногда, клиентам требуется гибкость иного рода. Процесс создания некоторых объектов состоит из множества этапов, и лишь потребителю известно, какие этапы обязательны, а какие нет. Паттерн *Строитель* позволяет клиентам собирать сложный объект по кусочкам, не вдаваясь в подробности того, как из этих кусочков получается окончательный результат. Строитель отлично подходит для создания тестируемых классов и тестовых данных, но активно применяется и в логике приложения.

В этой части книги мы очень подробно рассмотрим порождающие паттерны. Эти паттерны весьма просты по своей природе, но обладают множеством нюансов, с точки зрения реализации и влияния на дизайн приложения.