Что представляют собой паттерны проектирования? Паттерн представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования. В данном случае предполагается, что есть некоторый набор общих формализованных проблем, которые довольно часто встречаются, и паттерны предоставляют ряд принципов для решения этих проблем.

Хотя идея паттернов как способ описания решения распространенных проблем в области проектирования появилась довольно давно, но их популярность стала расти во многом благодаря известной работе четырех авторов Эриха Гаммы, Ричарда Хелма, Ральфа Джонсона, Джона Влиссидеса, которая называлась "Design Patterns: Elements of Reusable Object-Oriented Software" (на русском языке известна как "Приемы объектно-ориентированного проектирования. Паттерны проектирования") и которая вышла в свет в 1994 году. А сам коллектив авторов нередко называют "Банда четырёх" или Gang of Four или сокращенно GoF. Данная книга по сути являлась первой масштабной попыткой описать распространенные способы проектирования программ. И со временем применение паттернов стало считаться хорошей практикой программирования.

Что же дает нам применение паттернов? При написании программ мы можем формализовать проблему в виде классов и объектов и связей между ними. И применить один из существующих паттернов для ее решения. В итоге нам не надо ничего придумывать. У нас уже есть готовый шаблон, и нам только надо его применить в конкретной программе.

Причем паттерны, как правило, не зависят от языка программирования. Их принципы применения будут аналогичны и в C#, и в Jave, и в других языках. Хотя в рамках данного руководства мы будем говорить о паттернах в контексте языка C#.

Также мышление паттернами упрощает групповую разработку программ. Зная применяемый паттерн проектирования и его основные принципы другому программисту будет проще понять его реализацию и использовать ее.

В то же время не стоит применять паттерны ради самих паттернов. Хорошая программа предполагает использование паттернов. Однако не всегда паттерны упрощают и улучшают программу. Неоправданное их использование может привести к усложнению программного кода, уменьшению его качества. Паттерн должен быть оправданным и эффективным способом решения проблемы.

Существует множество различных паттернов, которые решают разные проблемы и выполняют различные задачи. Но по своему действию их можно объединить в ряд групп. Рассмотрим некоторые группы паттернов. В основу классификации основных паттернов положена цель или задачи, которые определенный паттерн выполняет.

**Порождающие паттерны**

Порождающие паттерны — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие:

* **Абстрактная фабрика (Abstract Factory)**
* **Строитель (Builder)**
* **Фабричный метод (Factory Method)**
* **Прототип (Prototype)**
* **Одиночка (Singleton)**

Другая группа паттернов - **структурные паттерны** - рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся:

* **Адаптер (Adapter)**
* **Мост (Bridge)**
* **Компоновщик (Composite)**
* **Декоратор (Decorator)**
* **Фасад (Facade)**
* **Приспособленец (Flyweight)**
* **Заместитель (Proxy)**

Третья группа паттернов называются **поведенческими** - они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие:

* **Цепочка обязанностей (Chain of responsibility)**
* **Команда (Command)**
* **Интерпретатор (Interpreter)**
* **Итератор (Iterator)**
* **Посредник (Mediator)**
* **Хранитель (Memento)**
* **Наблюдатель (Observer)**
* **Состояние (State)**
* **Стратегия (Strategy)**
* **Шаблонный метод (Template method)**
* **Посетитель (Visitor)**

Существуют и другие классификации паттернов в зависимости от того, относится паттерн к классам или объектам.

Паттерны классов описывают отношения между классами посредством наследования. Отношения между классами определяются на стадии компиляции. К таким паттернам относятся:

* **Фабричный метод (Factory Method)**
* **Интерпретатор (Interpreter)**
* **Шаблонный метод (Template Method)**
* **Адаптер (Adapter)**

Другая часть паттернов - **паттерны объектов** описывают отношения между объектами. Эти отношения возникают на этапе выполнения, поэтому обладают большей гибкостью. К паттернам объектов относят следующие:

* **Абстрактная фабрика (Abstract Factory)**
* **Строитель (Builder)**
* **Прототип (Prototype)**
* **Одиночка (Singleton)**
* **Мост (Bridge)**
* **Компоновщик (Composite)**
* **Декоратор (Decorator)**
* **Фасад (Facade)**
* **Приспособленец (Flyweight)**
* **Заместитель (Proxy)**
* **Цепочка обязанностей (Chain of responsibility)**
* **Команда (Command)**
* **Итератор (Iterator)**
* **Посредник (Mediator)**
* **Хранитель (Memento)**
* **Наблюдатель (Observer)**
* **Состояние (State)**
* **Стратегия (Strategy)**
* **Посетитель (Visitor)**

И это только некоторые основные паттерны. А вообще различных шаблонов проектирования гораздо больше. Одни из них только начинают применяться, другие являются популярными на текущий момент, а некоторые уже менее распространены, чем раньше.

И в данном руководстве мы рассмотрим наиболее основные и распространенные паттерны и принципы их использования применительно к языку C#.

**Как выбрать нужный паттерн?**

Прежде всего при решении какой-нибудь проблемы надо выделить все используемые сущности и связи между ними и абстрагировать их от конкретной ситуации. Затем надо посмотреть, вписывается ли абстрактная форма решения задачи в определенный паттерн. Например, суть решаемой задачи может состоять в создании новых объектов. В этом случае, возможно, стоит посмотреть на порождающие паттерны. Причем лучше не сразу взять какой-то определенный паттерн - первый, который показался нужным, а посмотреть на несколько родственных паттернов из одной группы, которые решают одну и ту же задачу.

При этом важно понимать смысл и назначение паттерна, явно представлять его абстрактную организацию и его возможные конкретные реализации. Один паттерн может иметь различные реализации, и чем чаще вы будете сталкиваться с этими реализациями, тем лучше вы будете понимать смысл паттерна. Но не стоит использовать паттерн, если вы его не понимаете, даже если он на первый взгляд поможет вам в решении задачи.

И в конечном счете надо придерживаться принципа KISS (Keep It Simple, Stupid) - сохранять код программы по возможности простым и ясным. Ведь смысл паттернов не в усложнении кода программы, а наоборот в его упрощении.