**ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ**

|  |  |
| --- | --- |
| **Название** | **Singleton** |
| **Тип** | Порождающий |
| **Краткое описание** | Создает единственный экземпляр некоторого класса и предоставляет к нему доступ. |
| **Применение** | Singleton.DoSomething();  Данный паттерн возлагает контроль над созданием единственного объекта на сам класс. Доступ к этому объекту осуществляется через статическую функцию-член класса, которая возвращает указатель или ссылку на него. Этот объект будет создан только при первом обращении к методу, а все последующие вызовы просто возвращают его адрес. Для обеспечения уникальности объекта, конструкторы и оператор присваивания объявляются закрытыми. |
| **Достоинства** | * Класс сам контролирует процесс создания единственного экземпляра. * Паттерн легко адаптировать для создания нужного числа экземпляров. * Возможность создания объектов классов, производных от Singleton. |
| **Недостатки** | * В случае использования нескольких взаимозависимых одиночек их реализация может резко усложниться. |

|  |  |
| --- | --- |
| **Название** | **Abstract Factory** |
| **Тип** | Порождающий |
| **Краткое описание** | Предоставляет метод фабрики для создания объектов или классов, которые связаны или зависимы, не уточняя конкретный класс.  Абстрактная фабрика представляет собой некоторый полиморфный базовый класс, назначением которого является объявление интерфейсов фабричных методов, служащих для создания продуктов всех основных типов (один фабричный метод на каждый тип продукта). Производные от него классы, реализующие эти интерфейсы, предназначены для создания продуктов всех типов внутри семейства или группы. |
| **Применение** | Factory.CreateProductA();  Factory.CreateProductB();  Используется если:   * Система должна оставаться независимой как от процесса создания новых объектов, так и от типов порождаемых объектов. * Необходимо создавать группы или семейства взаимосвязанных объектов, исключая возможность одновременного использования объектов из разных семейств в одном контексте. |
| **Достоинства** | * Скрывает сам процесс порождения объектов, а также делает систему независимой от типов создаваемых объектов, специфичных для различных семейств или групп (пользователи оперируют этими объектами через соответствующие абстрактные интерфейсы). * Позволяет быстро настраивать систему на нужное семейство создаваемых объектов |
| **Недостатки** | * Трудно добавлять новые типы создаваемых продуктов или заменять существующие, так как интерфейс базового класса абстрактной фабрики фиксирован. |

|  |  |
| --- | --- |
| **Название** | **Builder** |
| **Тип** | Порождающий |
| **Краткое описание** | Отделяет создание объекта от его представления, так что объект может поддерживать различные представления. Поэтапное создание сложного объекта.  Отделяет алгоритм поэтапного конструирования сложного продукта (объекта) от его внешнего представления так, что с помощью одного и того же алгоритма можно получать разные представления этого продукта. Поэтапное создание продукта означает его построение по частям. После того как построена последняя часть, продукт можно использовать. Для этого определяется алгоритм поэтапного создания продукта в специальном классе Director (распорядитель), а ответственность за координацию процесса сборки отдельных частей продукта возлагается на иерархию классов Builder. В этой иерархии базовый класс Builder объявляет интерфейсы для построения отдельных частей продукта, а соответствующие подклассы ConcreteBuilder их реализуют подходящим образом, например, создают или получают нужные ресурсы, сохраняют промежуточные результаты, контролируют результаты выполнения операций. |
| **Применение** | Builder.BuildPartA();  Builder.BuildPartB();  Build.GetFinalProduct();  Может помочь в решении следующих задач:   * В системе могут существовать сложные объекты, создание которых за одну операцию затруднительно или невозможно. Требуется поэтапное построение объектов с контролем результатов выполнения каждого этапа. * Данные должны иметь несколько представлений. |
| **Достоинства** | * Возможность контролировать процесс создания сложного продукта. * Возможность получения разных представлений некоторых данных. |
| **Недостатки** | * ConcreteBuilder и создаваемый им продукт жестко связаны между собой, поэтому при внесении изменений в класс продукта скорее всего придется соотвествующим образом изменять и класс ConcreteBuilder. |

|  |  |
| --- | --- |
| **Название** | **Strategy** |
| **Тип** | Поведения |
| **Краткое описание** | Определяет несколько алгоритмов для работы с наборами объектов. Паттерн переносит в отдельную иерархию классов все детали, связанные с реализацией алгоритмов. Для замены одного алгоритма другим достаточно перенастроить этот указатель на объект нужного типа. |
| **Применение** | List.Add(ObjA); List.Add(ObjB); List.SortStrategy(Ascending); List.SortStrategy(Descending);  Применяется в системах, поведение которых может определяться согласно одному алгоритму из некоторого семейства. Все алгоритмы этого семейства являются родственными: предназначены для решения общих задач, имеют одинаковый интерфейс для использования и отличаются только реализацией (поведением). Пользователь, предварительно настроив программу на нужный алгоритм (выбрав стратегию), получает ожидаемый результат |
| **Достоинства** | * Систему проще поддерживать и модифицировать, так как семейство алгоритмов перенесено в отдельную иерархию классов. * Паттерн предоставляет возможность замены одного алгоритма другим в процессе выполнения программы. * Паттерн позволяет скрыть детали реализации алгоритмов от клиента. |
| **Недостатки** | * Для правильной настройки системы пользователь должен знать об особенностях всех алгоритмов. * Число классов в системе, построенной с применением данного паттерна, возрастает. |