Введение в паттерны программирования

Паттерны (или шаблоны) проектирования описывают типичные способы решения часто встречающихся проблем при проектировании программ.

Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерн vs Алгоритм

Паттерн и алгоритм описывают типовые решения каких-то известных проблем

паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации

алгоритм — это кулинарный рецепт с чёткими шагами

Категории Паттернов программирования

Порождающие паттерны	**Структурные паттерны**	**Поведенческие паттерны**
• эти паттерны отвечают за создание объектов, скрывая детали их создания	• эти паттерны связывают объекты и классы в более крупные структуры и определяют отношения между ними.	• эти паттерны определяют способы взаимодействия между объектами и распределение обязанностей между ними

Factory (Фабрика)

- Паттерн Factory относится к категории **порождающих** паттернов и позволяет создавать объекты без явного указания их конкретных классов.

- используется интерфейс или абстрактный класс, который определяет метод-фабрику, ответственный за создание объектов.

Factory (Фабрика)

- Основная идея паттерна Factory заключается в том, что создание объектов делегируется подклассам, которые реализуют этот метод-фабрику в соответствии с конкретными требованиями или логикой.

Паттерн Builder (Строитель)

- используется для создания сложных объектов шаг за шагом. Он позволяет конструировать объекты с различными параметрами, обеспечивая гибкость и удобство
- относится к категории **порождающих** паттернов

Паттерн Adapter (Адаптер)

- относится к паттернам **структурного** типа и позволяет объединить два несовместимых интерфейса вместе. Он преобразует интерфейс одного класса в интерфейс, ожидаемый клиентом, чтобы они могли работать вместе без проблем.

- Цель паттерна Adapter состоит в том, чтобы обернуть один класс в другой класс, который предоставляет совместимый интерфейс. Таким образом, клиентский код может взаимодействовать с адаптером, не зная о том, что под капотом на самом деле находится другой класс.

Паттерн Стратегия (Strategy)

- Паттерн Стратегия помогает нам выбирать различные алгоритмы динамически в зависимости от ситуации

Паттерн Стратегия (Strategy)

- Например, представим, что у нас есть класс, который выполняет некоторый алгоритм сортировки. Вместо того, чтобы иметь разные методы в этом классе для каждого варианта сортировки (например, пузырьковая сортировка, быстрая сортировка), мы создаем отдельные классы для каждого алгоритма сортировки. Каждый класс имеет метод sort(), который реализует соответствующий алгоритм сортировки. Затем мы можем выбирать и использовать нужный класс алгоритма сортировки в зависимости от наших потребностей, просто создавая экземпляр класса и вызывая его метод sort().

Паттерн Builder

Приемущества	Недостатки
Позволяет создавать объекты с большим количеством опций	Может быть сложным в реализации и требует дополнительного кода
Упрощает процесс создания сложных объектов	Необходимость создания отдельного строителя для каждого класса
Позволяет создавать объекты с читаемым и гибким кодом	Может замедлить создание объектов из-за нескольких шагов
Обеспечивает контроль над шагами и порядком конструирования	
Изолирует сложную логику конструирования внутри строителя	

Паттерн Adapter

Приемущества	Недостатки
Позволяет работать с несовместимыми интерфейсами вместе	Увеличивает сложность кода из-за необходимости создания адаптеров
Повторное использование существующего кода	Может увеличить накладные расходы и сложность системы
Улучшает поддерживаемость и расширяемость системы	Необходимость создания адаптеров для каждого несовместимого класса
Изолирует клиентский код от внутренней реализации адаптируемого класса	

Паттерн Factory

Приемущества	Недостатки
Скрывает сложность создания объектов от клиента	Увеличивает сложность системы из-за необходимости создания фабрик
Обеспечивает гибкость и расширяемость в создании объектов	Добавляет накладные расходы из-за введения дополнительного уровня абстракции
Позволяет использовать абстрактный тип вместо конкретных классов	Возможно, снижение производительности из-за использования дополнительного слоя
Позволяет централизованно управлять созданием объектов в системе	

Паттерн Strategy

Приемущества	Недостатки
Позволяет выбирать и использовать различные алгоритмы динамически	Увеличивает количество классов и уровень абстракции
Обеспечивает гибкость в выборе и замене алгоритмов	Необходимо определить интерфейс или абстрактный класс стратегии
Улучшает поддерживаемость и расширяемость кода	
Избегает дублирования кода для разных вариантов алгоритмов	