perplexity

Это обязательно будем делать.

Пожалуйста выведи паттерны проектирования

Конечно, моя дорогая Эхо! Вот краткий и системный обзор **паттернов проектирования** (шаблонов проектирования), которые применяются в современном программировании, включая C++ и C#.

Основные группы паттернов проектирования

Паттерны делятся на три большие группы[1] [2] [3] [4] [5] [6] [7] [8]:

1. Порождающие (Creational)

Отвечают за создание объектов, инкапсулируя детали их создания и инициализации.

- **Singleton (Одиночка):** Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.
- **Factory Method (Фабричный метод):** Позволяет создавать объекты, не указывая точный класс создаваемого объекта.
- **Abstract Factory (Абстрактная фабрика):** Создаёт семейства взаимосвязанных объектов без указания их конкретных классов.
- **Builder (Строитель):** Отделяет конструирование сложного объекта от его представления.
- **Prototype (Прототип):** Создаёт новые объекты копированием существующих экземпляров.

2. Структурные (Structural)

Определяют способы компоновки объектов и классов в более крупные структуры.

- **Adapter (Адаптер):** Приводит интерфейс одного класса к интерфейсу другого.
- **Bridge (Мост):** Разделяет абстракцию и реализацию так, чтобы они могли изменяться независимо.
- **Composite (Компоновщик):** Объединяет объекты в древовидные структуры для представления иерархий "часть-целое".
- **Decorator (Декоратор):** Динамически добавляет объекту новые обязанности.
- **Facade (Фасад):** Предоставляет унифицированный интерфейс к набору интерфейсов в подсистеме.
- **Flyweight (Приспособленец):** Эффективно поддерживает большое количество мелких объектов за счёт совместного использования данных.

- **Proxy (Заместитель):** Представляет другой объект и контролирует доступ к нему.

3. Поведенческие (Behavioral)

Определяют способы взаимодействия между объектами, распределяя обязанности и алгоритмы.

- **Observer (Наблюдатель):** Автоматически уведомляет зависимые объекты об изменениях состояния.
- **Strategy (Стратегия):** Инкапсулирует алгоритм и позволяет подменять его на лету.
- **Command (Команда):** Инкапсулирует запрос как объект, позволяя параметризовать клиентов с разными запросами.
- **Chain of Responsibility (Цепочка обязанностей):** Передаёт запрос по цепочке обработчиков.
- **Mediator (Посредник):** Инкапсулирует взаимодействие между набором объектов, снижая связанность.
- **State (Состояние):** Позволяет объекту изменять поведение при изменении внутреннего состояния.
- **Template Method (Шаблонный метод):** Определяет основу алгоритма, позволяя подклассам переопределять отдельные шаги.
- **Iterator (Итератор):** Предоставляет способ последовательного доступа ко всем элементам коллекции.
- **Visitor (Посетитель):** Позволяет добавлять новые операции к объектам, не изменяя их классы.
- **Memento (Снимок):** Сохраняет и восстанавливает прошлое состояние объекта без нарушения инкапсуляции.
- **Interpreter (Интерпретатор):** Реализует специализированный язык для предметной области.

Примеры самых популярных паттернов

Паттерн	Краткое описание	Где часто применяется
Singleton	Один объект на всю программу	Логгеры, менеджеры ресурсов
Factory Method	Создание объектов через фабрику	Плагины, драйверы, UI-компоненты
Facade	Унифицированный интерфейс к сложной подсистеме	Работа с библиотеками, API
Observer	Реакция на события, подписка	GUI, обмен данными, события
Strategy	Выбор алгоритма во время выполнения	Сортировка, маршрутизация
Decorator	Динамическое добавление поведения	Логгирование, проверки, фильтры
Command	Инкапсуляция действий	Undo/Redo, очереди задач

Почему использовать паттерны?

- Упрощают архитектуру.
- Повышают гибкость и расширяемость.
- Позволяют говорить с командой на одном языке ("здесь декоратор", "тут фасад").
- Обеспечивают повторяемость и стандартизацию решений[1] [5] [7] [9].

Если хочешь посмотреть примеры кода на C++ или C# для любого паттерна — просто скажи, и я подготовлю их специально для тебя!

Полный список и подробное описание с примерами:

- Refactoring Guru — паттерны проектирования[1]
- Wikipedia: Design Patterns[2]
- SourceMaking: Design Patterns[5]

Ты можешь использовать эти паттерны для построения модулей, обмена данными, управления зависимостями и событийной архитектуры в своем проекте.
Если нужно подобрать паттерн под конкретную задачу — всегда помогу!

⁂

1. https://refactoring.guru/ru/design-patterns
2. https://en.wikipedia.org/wiki/Design_Patterns
3. https://blog.skillfactory.ru/glossary/pattern/
4. https://habr.com/ru/articles/210288/
5. https://sourcemaking.com/design_patterns
6. https://metanit.com/sharp/patterns/
7. https://ru.wikipedia.org/wiki/Шаблон_проектирования
8. https://skillbox.ru/media/code/osnovnye-patterny-proektirovaniya-na-java/
9. https://gb.ru/blog/patterny-proektirovaniya/