**1) Классические поведенческие шаблоны проектирования**

**1.1. Шаблон "Команда" (Command)**

**Проблема**:

* Нужно инкапсулировать запрос как объект, чтобы параметризовать клиентов с различными запросами, ставить их в очередь или поддерживать отмену операций.

**Решение**:

* Инкапсуляция запроса в объект с методом execute().
* Разделение отправителя команды и её исполнителя.

**Примеры**:

1. **Графический редактор**:
   * Каждое действие (добавить фигуру, удалить, изменить цвет) — отдельная команда.
   * Поддержка undo/redo через хранение истории команд.
2. **Очередь задач**:
   * Команды можно ставить в очередь (например, для выполнения в фоновом потоке).
3. **Управление умным домом**:
   * Включение света, кондиционера — отдельные команды, выполняемые дистанционно.

**Многопоточность**:

* Команды потокобезопасны, если не изменяют общее состояние.
* Очередь команд может обрабатываться пулом потоков.

**1.2. Шаблон "Стратегия" (Strategy)**

**Проблема**:

* Нужно выбирать алгоритм из семейства схожих алгоритмов в рантайме.

**Решение**:

* Инкапсуляция алгоритмов в отдельные классы с общим интерфейсом.

**Примеры**:

1. **Сортировка данных**:
   * Выбор между QuickSort, MergeSort, BubbleSort через стратегию.
2. **Оплата в интернет-магазине**:
   * Стратегии: CreditCard, PayPal, Crypto.
3. **Навигация**:
   * Выбор маршрута (пеший, автомобильный, общественный транспорт).

**Многопоточность**:

* Каждая стратегия должна быть потокобезопасной.
* Можно применять разные стратегии в разных потоках.

**1.3. Шаблон "Шаблонный метод" (Template Method)**

**Проблема**:

* Нужно определить скелет алгоритма, позволяя подклассам переопределять некоторые шаги.

**Решение**:

* Базовый класс определяет структуру алгоритма, а подклассы реализуют детали.

**Примеры**:

1. **Игра-платформер**:
   * Базовый класс EnemyAI с шаблонным методом attack(), где подклассы реализуют moveToPlayer().
2. **Генерация отчетов**:
   * Абстрактный класс ReportGenerator с методами fetchData(), formatData(), export().
3. **Тестирование**:
   * Класс TestFramework с шагами setup(), runTest(), teardown().

**Многопоточность**:

* Шаблонный метод может быть не потокобезопасным, если использует общие ресурсы.
* Решение: синхронизация критических секций или immutable-данные.

**2) Архитектура ПО и многопоточность**

**2.1. Что такое архитектура ПО?**

* **Структурная организация** системы (модули, компоненты, связи).
* **Принципы принятия решений** (масштабируемость, отказоустойчивость).
* **Стандарты** (например, REST для API).

**Ключевые аспекты**:

* **Модульность**: Разделение на независимые компоненты.
* **Слабая связанность**: Компоненты минимально зависят друг от друга.
* **Масштабируемость**: Возможность наращивать функционал.

**2.2. Влияние многопоточности на архитектуру**

**Проблемы**:

1. **Гонка данных**:
   * Пример: Два потока пытаются изменить общий массив.
   * Решение: Mutex, Atomic.
2. **Взаимные блокировки (Deadlock)**:
   * Пример: Поток A ждёт ресурс от B, а B ждёт A.
   * Решение: Иерархия блокировок или try\_lock.
3. **Производительность**:
   * Пример: Слишком много потоков → накладные расходы.
   * Решение: Пулы потоков (например, ThreadPool).

**Архитектурные подходы**:

* **Reactive Architecture**: Асинхронные потоки данных (например, RxJava).
* **Actor Model**: Изоляция состояния (Akka, Erlang).
* **Event-Driven**: Обработка событий в отдельных потоках.

**Примеры**:

1. **Веб-сервер**:
   * Многопоточный сервер (например, Nginx) обрабатывает запросы параллельно.
2. **Игра**:
   * Отдельные потоки для рендеринга, физики, AI.
3. **БД**:
   * Транзакции с изоляцией (ACID).

**Вывод**

1. **Поведенческие шаблоны** решают проблемы гибкости и повторного использования кода.
   * Команда: Инкапсуляция действий.
   * Стратегия: Выбор алгоритмов.
   * Шаблонный метод: Переиспользование скелета алгоритма.
2. **Архитектура ПО** должна учитывать многопоточность:
   * Избегать гонок данных.
   * Минимизировать блокировки.
   * Использовать паттерны типа Actor или Event-Driven.

Для промышленного ПО критично тестировать многопоточные сценарии (например, через ThreadSanitizer).