1. Поведенческие шаблоны проектирования

Классические поведенческие шаблоны проектирования — это шаблоны, описывающие способы организации взаимодействия между объектами и распределения обязанностей таким образом, чтобы обеспечить гибкость, расширяемость и минимальную связанность компонентов системы.

Их основная цель — инкапсулировать алгоритмы, варианты поведения и взаимодействие между объектами, позволяя изменять поведение программы без изменения самих объектов.

2. Классические поведенческие шаблоны проектирования

2.1 Состояние (State)

Проблема:

Нужно реализовать объект, поведение которого зависит от его состояния, при этом избегая большого количества `if-else` или `switch-case`.

Решение:

Инкапсуляция поведения в отдельные классы-состояния. Объект делегирует поведение текущему состоянию, которое можно динамически изменять.

Инженерный анализ:

- Инкапсуляция: Поведение связано с отдельными объектами состояния.
- Разделяй и властвуй: Поведение разнесено по различным классам, легко поддерживать и расширять.
- Ортогональность: Можно менять состояния независимо от основного класса.

Пример (Qt):

В Qt реализован похожий подход в QStateMachine. Поведение интерфейса зависит от состояния. Например, кнопка может быть в состояниях `enabled`, `disabled`, `loading`, и каждый из них определяет набор действий.

Многопоточность:

Если состояние меняется часто, требуется синхронизация (например, AtomicReference в Java).

2.2 Посредник (Mediator)

Проблема:

Множество объектов взаимодействуют напрямую, что приводит к сильной связанности и затруднённой модификации/расширению.

Решение:

Создаётся объект-посредник, который инкапсулирует взаимодействие между участниками.

Инженерный анализ:

- Инкапсуляция: Взаимодействие инкапсулировано в одном месте.
- Разделяй и властвуй: Объекты не знают друг о друге, только о посреднике.
- Ортогональность: Участников можно заменять без изменения других.

Пример (Qt):

QDialog часто выступает как посредник между кнопками, полями и логикой. Вместо того чтобы каждое поле знало о других, диалоговая форма управляет логикой взаимодействия.

Многопоточность:

Посредник может стать узким местом (требует thread-safe реализации, например, synchronized методов).

2.3 Наблюдатель (Observer)

Проблема:

Нужно, чтобы несколько объектов реагировали на изменение состояния другого объекта без сильной связанности.

Решение:

Используется подписка на уведомления: субъект сообщает наблюдателям об изменениях.

Инженерный анализ:

- Инкапсуляция: Логика уведомлений и реакции разделены.
- Разделяй и властвуй: Наблюдатели не знают о внутренностях субъекта.
- Ортогональность: Добавление новых наблюдателей не влияет на субъект.

Пример (Qt):

Сигналы и слоты — это реализация шаблона "наблюдатель". Объекты могут подписываться на события (например, нажатие кнопки), не зная деталей друг друга.

Многопоточность:

Уведомления должны быть потокобезопасными (например, через SynchronizedList или механизм событий в Qt).

3. Влияние многопоточности на дизайн ПО

Многопоточность — мощный инструмент повышения производительности, но она сильно влияет на дизайн системы, предъявляя дополнительные требования:

- Безопасность данных: требуется синхронизация доступа к разделяемым ресурсам.
- Изоляция и ортогональность: проектирование модулей, устойчивых к параллельному исполнению.
- Реактивный и событийный подход: шаблоны типа "Наблюдатель" и "Состояние" часто используются вместе с потоками (например, обработка событий в UI-потоке и рабочем потоке).
- Посредник облегчает коммуникацию между потоками, часто выступая в роли очереди или контроллера.

В Qt реализация многопоточности завязана на `QThread`, а взаимодействие между потоками организовано через сигналы и слоты (автоматически обеспечивается потокобезопасность при `QueuedConnection`), что позволяет эффективно использовать поведенческие шаблоны в многопоточной среде.