**Отчёт по лабораторной работе: Применение паттерна "Мост" для управления темами интерфейса**

**1. Описание проблемы предметной области**

Программа предназначена для управления темами оформления графического интерфейса, включая поддержку светлой, тёмной и пользовательских тем. Без использования паттернов проектирования реализация такой системы столкнулась бы со следующими проблемами:

* **Жёсткая связь между виджетами и стилями**: Каждый виджет (кнопка, метка, поле ввода и т.д.) содержал бы логику применения стилей для всех тем, что привело бы к дублированию кода.
* **Сложность добавления новых тем**: для каждой новой темы пришлось бы изменять все классы виджетов, нарушая принцип открытости/закрытости.
* **Нарушение принципа единственной ответственности**: Классы виджетов совмещали бы логику отображения и стилизации, усложняя поддержку кода.

Целью работы было создание гибкой архитектуры, позволяющей добавлять темы без изменения существующего кода виджетов.

**2. Решение: применение паттерна "Мост"**

Паттерн "Мост" был применён для разделения **абстракции** (виджеты интерфейса) и **реализации** (стили тем).

**Структура решения**:

* **Абстракция**:
  + Абстрактный класс Widget, определяющий интерфейс для рендеринга.
  + Конкретные реализации: Button, Label, LineEdit, CheckBox, ComboBox, Slider, TextEdit.  
    Каждый виджет содержит ссылку на объект Theme и делегирует ему применение стилей через метод render().
* **Реализация**:
  + Интерфейс Theme с методом applyStyle(QWidget\*).
  + Конкретные реализации: LightTheme, DarkTheme, CustomTheme.  
    Классы тем инкапсулируют логику стилизации для различных виджетов.

**Взаимодействие**:

* Виджеты принимают объект Theme через конструктор.
* Метод render() вызывает theme->applyStyle(), передавая Qt-виджет для стилизации.
* ThemeManager управляет созданием и хранением тем, обеспечивая их повторное использование.

**3. Диаграмма классов**

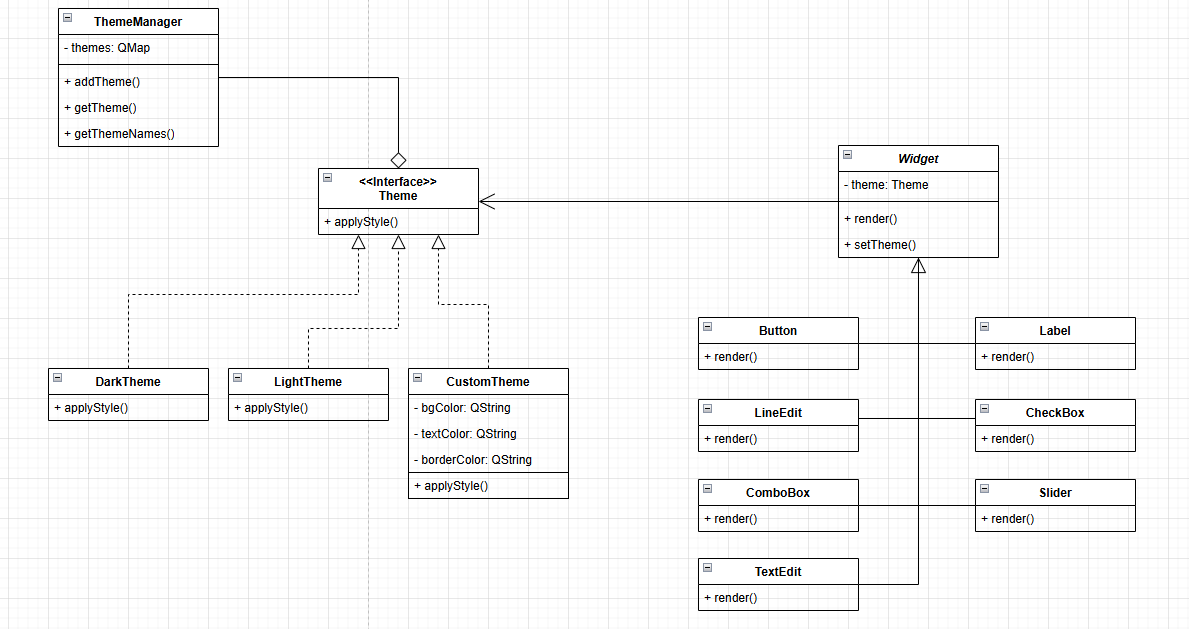
****

Рисунок 1 – Архитектура приложения с паттерном “Мост”

На рисунке 1 изображена структура классов приложения с использованием паттерна “Мост”, где сам паттерн используется для разделения абстракции виджетов (отображаемых элементов интерфейса) и реализации тем (стилей внешнего вида отображаемых элементов).  
**Элементы и связи:**

* **Абстракция (виджеты):**
  + **Базовый класс Widget:**
    - **Атрибуты: theme: Theme\* (ссылка на объект темы).**
    - **Методы:**
      * **render(QWidget\*): абстрактный метод для применения стиля.**
      * **setTheme(Theme\*): изменяет тему виджета.**
  + **Конкретные классы:**
    - **Button, Label, LineEdit, CheckBox, ComboBox, Slider, TextEdit.**
    - **Каждый переопределяет render(), вызывая theme->applyStyle().**
* **Реализация (темы):**
  + **Интерфейс Theme:**
    - **Виртуальный метод applyStyle(QWidget\*).**
  + **Конкретные реализации:**
    - **LightTheme, DarkTheme: применяют предустановленные стили.**
    - **CustomTheme: принимает параметры цвета через конструктор и генерирует стили динамически.**
* **Менеджер тем:**
  + **Класс ThemeManager:**
    - **Атрибуты: themes: QMap<QString, Theme\*>.**
    - **Методы:**
      * **addTheme(): регистрирует новую тему.**
      * **getTheme(): возвращает тему по имени.**
      * **getThemeNames(): возвращает список доступных тем.**

**Типы связей:**

1. **Композиция:**
   * **Widget содержит Theme\* (виджеты не владеют темой, но используют её).**
2. **Агрегация:**
   * **ThemeManager хранит темы в QMap (темы существуют независимо от менеджера).**
3. **Наследование:**
   * **Все виджеты наследуют Widget.**
   * **Все темы реализуют интерфейс Theme.**

**4. Преимущества использования паттерна**

1. **Гибкость и расширяемость**:
   * Добавление новой темы (например, HighContrastTheme) требует только реализации интерфейса Theme.
   * Виджеты остаются неизменными.
2. **Устранение дублирования кода**:
   * Логика стилизации вынесена в отдельные классы, а не дублируется в каждом виджете.
3. **Соблюдение принципов SOLID**:
   * Принцип единственной ответственности: виджеты отвечают за отображение, темы — за стилизацию.
   * Принцип открытости/закрытости: система открыта для расширения (новые темы), но закрыта для модификации.
4. **Упрощение тестирования**:
   * Виджеты и темы тестируются независимо.

**5. Сравнение с версией без паттерна (подробно)**

**Архитектура без паттерна "Мост"**:

* **Проблемы**:
  + Каждый виджет сразу является виджетом с темой:
  + При добавлении новой темы (например, BlueTheme) нужно:
    - Добавить новые классы BlueLabel() и т.д.
    - Изменить логику выбора темы в коде.
* **Недостатки**:
  + **Дублирование кода**: Одни и те же виджеты с разными стилями просто дублируются с минимальными изменениями.
  + **Сложность поддержки**: Изменение стиля границы потребует создания нового класса.
  + **Невозможность динамического изменения**: Темы "зашиты" в код, пользовательские темы нереализуемы.

**Архитектура с паттерном "Мост"**:

* **Решенные проблемы**:
  + Темы вынесены в отдельную иерархию.
  + Добавление CustomTheme через GUI:
  + Код виджетов остаётся неизменным.

**6. Вывод**  
Применение паттерна "Мост" кардинально улучшило архитектуру программы:

1. **Динамическое управление темами**: Пользователи могут создавать кастомные темы через GUI, что было бы невозможно в жёстко связанной системе.
2. **Сокращение затрат на поддержку**: Добавление новой темы требует написания всего одного класса (например, HighContrastTheme), а не правок во всех виджетах.
3. **Чистота кода**: Логика стилей изолирована в классах Theme, что упрощает тестирование и отладку.
4. **Масштабируемость**: Система легко адаптируется для новых виджетов (например, ProgressBar) — достаточно унаследовать их от Widget и вызвать applyStyle().

Таким образом, паттерн "Мост" не только решил исходные проблемы, но и заложил основу для будущих улучшений, соответствуя принципам современного проектирования ПО.