ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Паттерн проектирования — это часто встречающееся решение определённой **проблемы** при проектировании архитектуры программ.

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

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

Составляющие паттерна

Паттерны, как правило описываются формально и чаще всего состоят из пунктов:
Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

1. проблема, которую решает паттерн;

2. структуры классов, составляющих решение;

3. особенностей реализации в различных контекстах;

4. связей с другими паттернами.

Классификация паттернов

Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы.

Низкоуровневые и простые паттерны — идиомы. Они ***не универсальны***, поскольку применимы только в рамках одного языка программирования.

Идиома программирования *— устойчивый способ выражения некоторой составной конструкции в одном или нескольких языках программирования. Идиома является шаблоном решения задачи, записи алгоритма или структуры данных путём комбинирования встроенных элементов языка. Идиому можно считать самым низкоуровневым шаблоном проектирования.*

Пример простой идиомы:

Инкремент

```
i += 1; /* i = i + 1; */
++i; /* тот же результат */
i++; /* тот же результат */
```

Самые **универсальные** — **архитектурные паттерны**, которые можно реализовать практически на любом языке.

Существует три основные группы паттернов:

- **Порождающие паттерны** (*для создания объектов без внесения в программу лишних зависимостей*).
- **Структурные паттерны** (*показывают различные способы построения связей между объектами.*)
- **Поведенческие паттерны** (*эффективная коммуникации между объектами*).

Зачем знать паттерны?

1. Проверенные решения. Тратится меньше времени при использовании готовых решений.
2. Стандартизация кода. Делается меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.

3. Общий программистский словарь. В предлагаемом решении достаточно указать название паттерна, что бы всем остальным стало понятно о чем идет речь.

ПОРОЖДАЮЩИЕ ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Порождающие паттерны - отвечают за ***удобное*** и ***безопасное*** создание новых объектов или даже целых семейств объектов.

1. **Одиночка**
2. **Фабричный метод**
3. **Абстрактная фабрика**
4. **Строитель**
5. **Прототип**

Паттерн Одиночка (Singleton)

Одиночка — это порождающий паттерн проектирования, который гарантирует, что у класса есть только **один экземпляр**, и предоставляет к нему **глобальную точку доступа**.

Применяется когда

1. В программе должен быть единственный экземпляр какого-то класса, доступный всем клиентам, например, общий доступ к базе данных из разных частей программы.

Одиночка скрывает от клиентов все способы создания нового объекта, кроме ***специального метода***. Этот метод либо ***создаёт объект***, либо ***отдаёт существующий объект***, если он уже был создан.

2. Необходимо иметь больше контроля над глобальными переменными.

В отличие от глобальных переменных, паттерн Одиночка гарантирует, что никакой другой код не заменит созданный экземпляр класса, поэтому вы всегда уверены в наличии лишь одного объекта-одиночки.

Паттерн Одиночка (Singleton)

Шаги реализации

1. Добавьте в класс приватное статическое поле, которое будет содержать одиночный объект.

2. Объявите статический создающий метод, который будет использоваться для получения одиночки.

3. Добавьте «ленивую инициализацию» (создание объекта при первом вызове метода) в создающий метод одиночки.

4. Сделайте конструктор класса приватным.

Паттерн Одиночка (Singleton) - Пример

```
class Singleton
{
public:
static Singleton& Instance()
{
// согласно стандарту, этот код ленивый и потокобезопасный
static Singleton s;
return s;
}
private:
Singleton() { }  // конструктор недоступен
~Singleton() {

} // и деструктор
// необходимо также запретить копирование
Singleton(Singleton const&); // реализация не нужна
Singleton& operator= (Singleton const&);  // и тут
};

int main(int argc, char** argv) {
//new Singleton(); // Won't work
Singleton& instance = Singleton::Instance();
return 0;
}
```

Паттерн Одиночка (Singleton) – пример класса

```
class Singleton
{public:
 static Singleton& Instance()
 { static Singleton s;
   return s;
 }
 void SomeInformation()
 {
  for (int i = 0; i < 10; i++) std::cout << mas[i] << ", ";
  std::cout <<std::endl;
 }
 private:В private части реализуем конструктор для инициализации массива
  Singleton() {
  for (int i = 0; i < 10; i++){mas[i] = 111; }
 };
 ~Singleton() { std::cout << "Private Destructor is called"<<std::endl;
};
 Singleton(Singleton const&); // реализация не нужна
 Singleton& operator= (Singleton const&);
 protected:
 int mas[10];}; В части prodected объявляем массив
```

Паттерн Одиночка (Singleton)

```
int main()
{
        //new Singleton(); // Создание объекта невозможно
        Singleton& instance = Singleton::Instance();
        Singleton::Instance().SomeInformation();
}
```