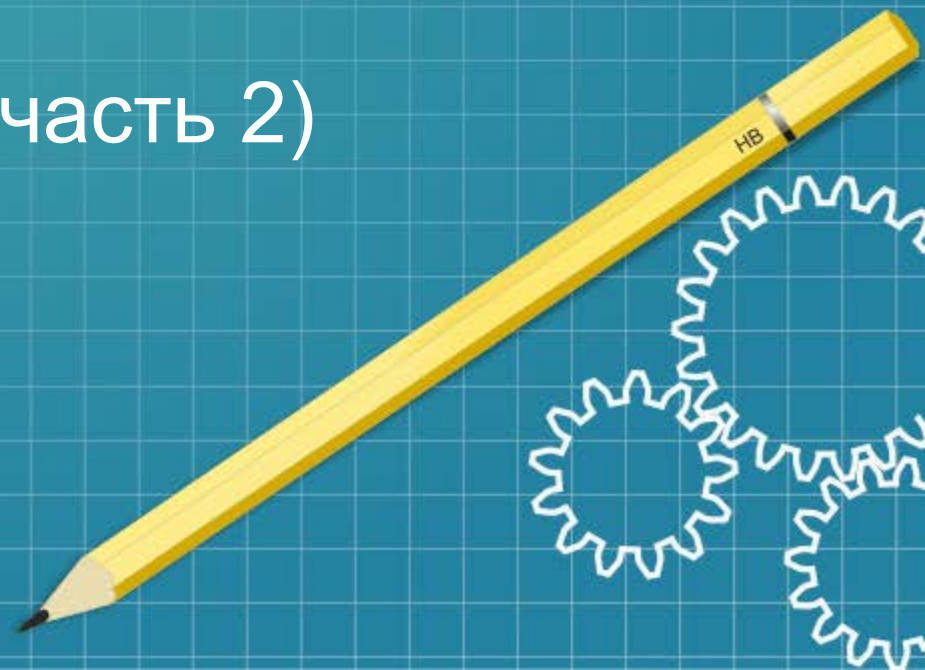
Модуль 2

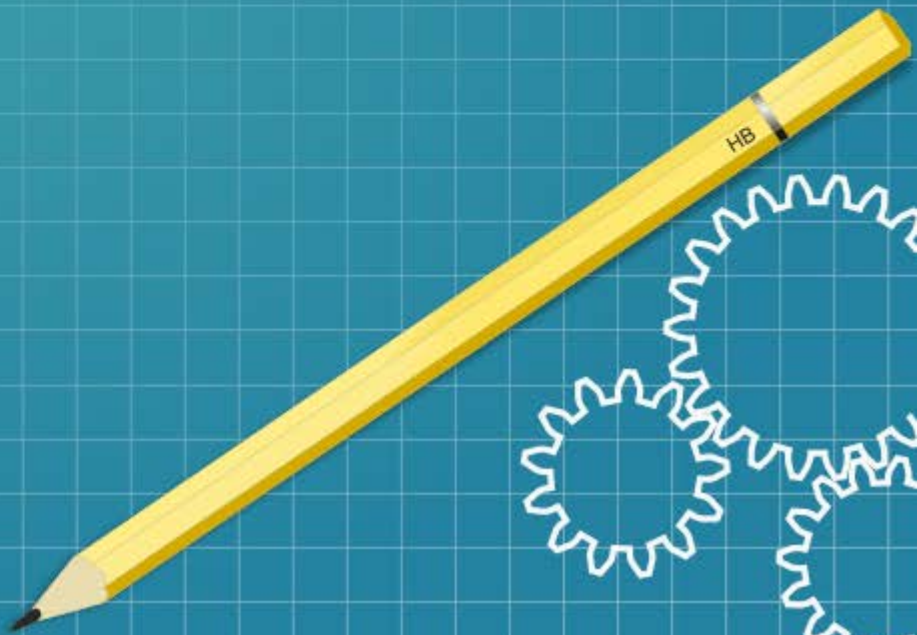
- Порождающие паттерны (часть 2)

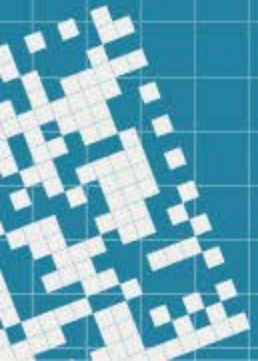


Содержание

- **Модуль 2 – Порождающие паттерны (часть 2)**
  - Builder
  - Prototype
  - Singleton
  - Практические примеры использования порождающих паттернов

Builder





Builder

Название паттерна - Builder/Строитель.
Описан в работе [GoF95].

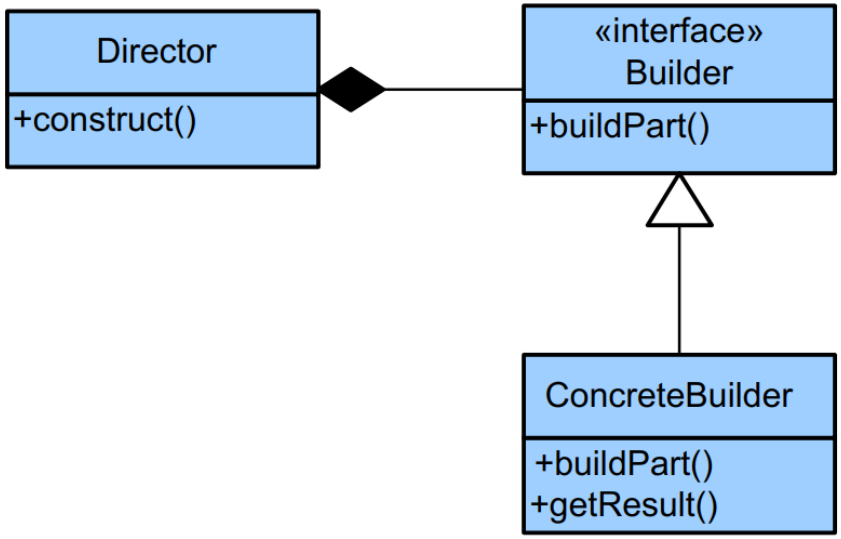
Цель паттерна
Отделяет процесс конструирования сложного объекта от его представления так, что в результате одного и того же процесса конструирования получаются разные представления [GoF95]. Иными словами, клиентский код может создавать сложный объект, определяя для него не только тип, но и содержимое. При этом клиент не должен знать о деталях конструирования объекта. [Grand2004]

Паттерн следует использовать когда…
- Общий алгоритм построения сложного объекта не должен зависеть от специфики каждого из его шагов.
- В результате одного и того же алгоритма конструирования надо получить различные продукты.

Результаты использования паттерна
- Есть возможность изменять внутреннюю структуру создаваемого продукта (или создать новый продукт)
- Повышение модульности за счет разделения распорядителя и строителя
- Пошаговое построение продукта

Builder

Структура паттерна




```
class Program
{
    static void Main(string[] args)
    {
        Director director = new Director();
        ConcreteBuilder builder = new ConcreteBuilder();
        director.Builder = builder;

        Console.WriteLine("Standard basic product:");
        director.buildMinimalViableProduct();
        Console.WriteLine(builder.GetProduct().ListParts());

        Console.WriteLine("Standard full featured product:");
        director.buildFullFeaturedProduct();
        Console.WriteLine(builder.GetProduct().ListParts());

        Console.WriteLine("Custom product:");
        builder.BuildPartA();
        builder.BuildPartC();
        Console.Write(builder.GetProduct().ListParts());
    }
}
```

Builder

Участники

- **Builder** – строитель:
- Обеспечивает интерфейс для пошагового конструирования сложного объекта (продукта) из частей.
- Пример: ТехнологияМодели

- **ConcreteBuilder** - конкретный строитель:
  - Реализует шаги построения сложного объекта, определенные в базовом классе Builder.
  - Создает результат построения (Product) и следит за пошаговым конструированием.
  - Определят интерфейс для доступа к результату конструирования.
  - Пример: ТехнологияМиниАвто и др

- **Director** – распорядитель:
- Определяет общий алгоритм конструирования, используя для реализации отдельных шагов возможности класса Builder.
- Пример: Конвейер

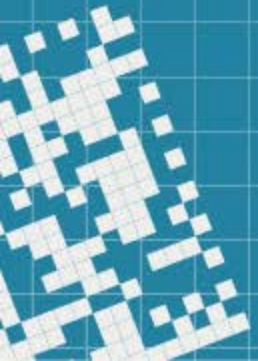
- **Product –** продукт:
- Сложный объект, который получается в результате конструирования.
- Пример: МиниАвто и др.

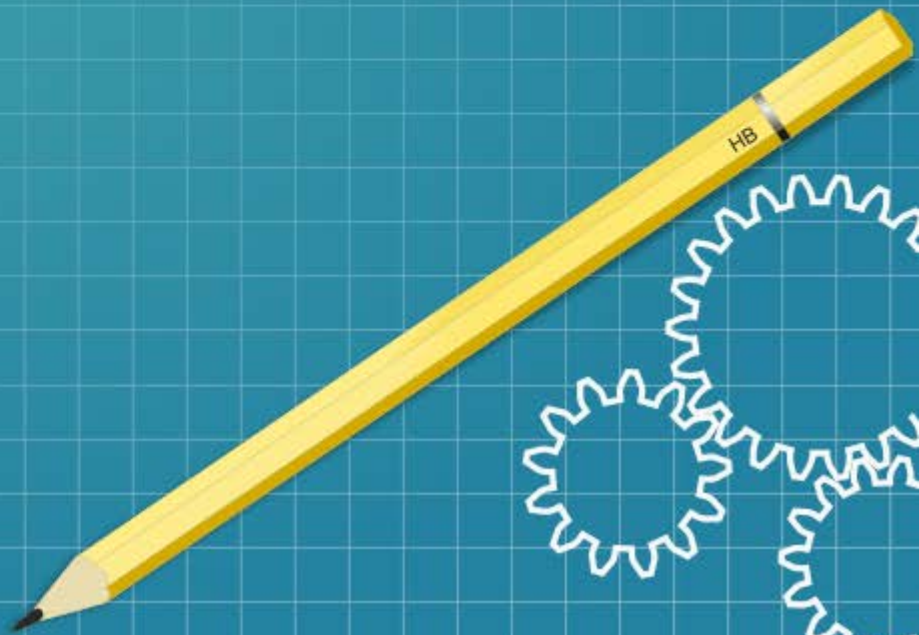
Builder

Практические задания

- Telephone
  - Производитель: Samsung
  - Комплектующие: Body, BatteryPlus, Cover
  - Создать комплектации: Basic (Body), Standard (Body + BatteryPlus), Lux (Body + BatteryPlus + Cover)
- Car
  - Производитель: Ford
  - Комплектующие: Body, TankPlus, Conditioner, Nitro
  - Создать комплектации: Basic, Standard, Lux
- Computer
  - Производитель: Dell
  - Комплектующие: Mainboard, Processor, VideoCard, SoundCard, Tuner
  - Создать комплектации: Basic, Standard, StandardPlus, Lux

Prototype





Prototype

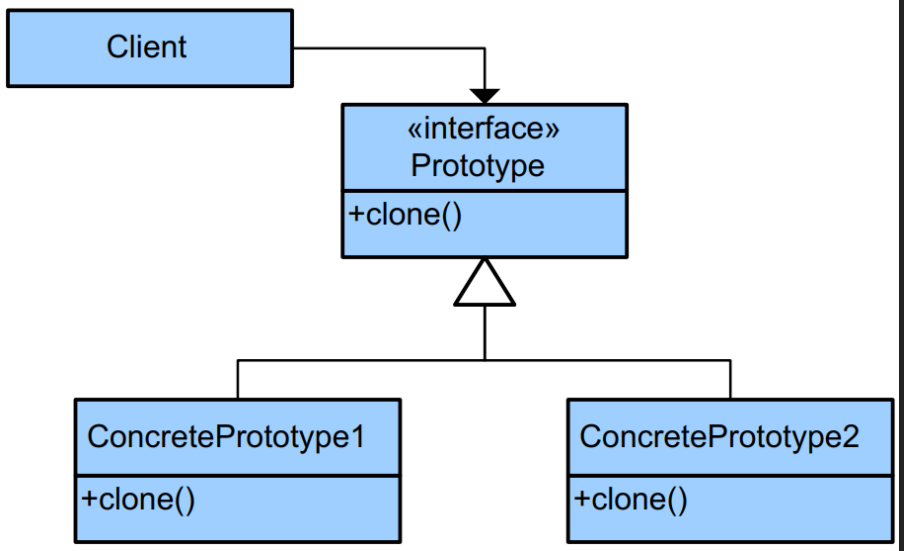
Название паттерна - Prototype/Прототип.
Описан в работе [GoF95].

Цель паттерна
Определяет виды создаваемых объектов с помощью экземпляра-прототипа и создает новые объекты путем копирования этого прототипа [GoF95]

Паттерн следует использовать когда…
- Клиентский код должен создавать объекты, ничего не зная об их классе, или о том, какие данные они содержат
- Классы создаваемых объектов определяются во время выполнения (например, при динамической загрузке)
- Экземпляры класса могут пребывать в не очень большом количестве состояний, поэтому может оказаться значительно удобнее создать несколько прототипов и клонировать их вместо прямого создания экземпляра класса.

Результаты использования паттерна
- Добавление/удаление новых типов продуктов во время выполнения
- Определение новых типов продуктов без необходимости наследования
- Использование диспетчера прототипов
- Отсутствие параметров в методе Clone()
- Реализация метода Clone() — наиболее трудная часть при использовании паттерна

Prototype




```
static void Main(string[] args)
{
    Person p1 = new Person();
    p1.Age = 42;
    p1.BirthDate = Convert.ToDateTime("1977-01-01");
    p1.Name = "Jack Daniels";
    p1.IdInfo = new IdInfo(666);

    Person p2 = p1.ShallowCopy();
    Person p3 = p1.DeepCopy();

    Console.WriteLine("Original values of p1, p2, p3:");
    Console.WriteLine("   p1 instance values: ");
    DisplayValues(p1);
    Console.WriteLine("   p2 instance values:");
    DisplayValues(p2);
    Console.WriteLine("   p3 instance values:");
    DisplayValues(p3);

    p1.Age = 32;
    p1.BirthDate = Convert.ToDateTime("1900-01-01");
    p1.Name = "Frank";
    p1.IdInfo.IdNumber = 7878;
    Console.WriteLine("\nValues of p1, p2 and p3 after changes to p1:");
    Console.WriteLine("   p1 instance values: ");
    DisplayValues(p1);
    Console.WriteLine("   p2 instance values (reference values have chang
    DisplayValues(p2);
    Console.WriteLine("   p3 instance values (everything was kept the sam
    DisplayValues(p3);
}

public static void DisplayValues(Person p)
{
    Console.WriteLine("      Name: {0:s}, Age: {1:d}, BirthDate: {2:MM/dd
```

Prototype

Участники

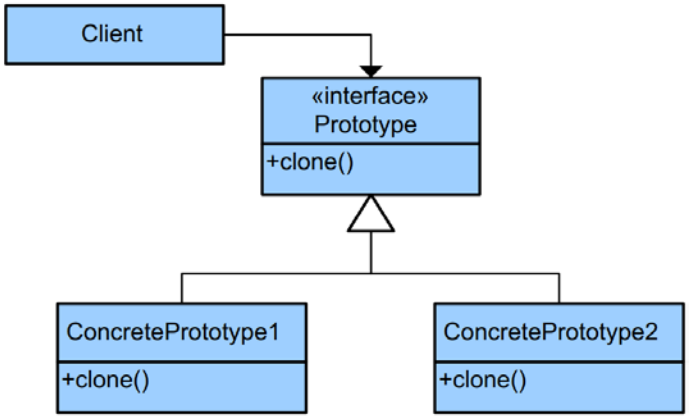
- **Prototype** – прототип:
- Определяет интерфейс для клонирования самого себя

- **ConcretePrototype** – конкретный прототип:
- Реализует операцию клонирования самого себя.

- **Client** – клиент:
- Создает новый объект, посылая запрос прототипу копировать самого себя.
- Пример: АбстрактнаяПуля

Лабораторная работа

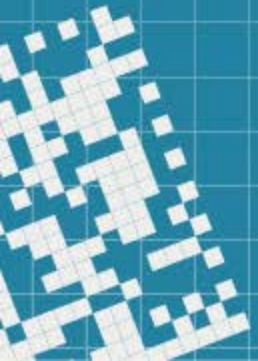
Цель работы: создать работающий программный код используя данные с этого слайда

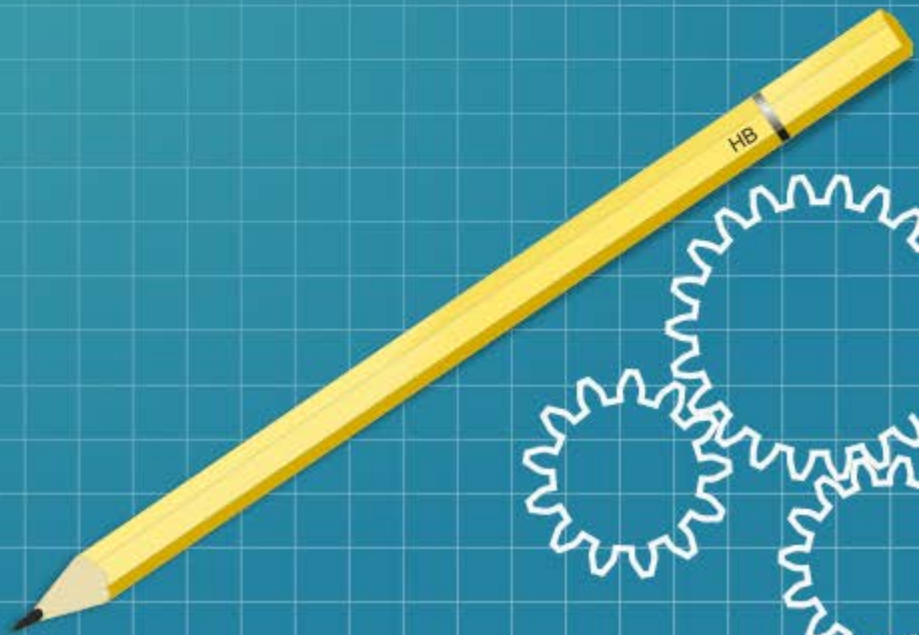



```
C:\Windows\system32\cmd.exe
Original values of p1, p2, p3:
   p1 instance values:
      Name: Jack, Second Name: Daniels, Age: 42, BirthDate: 01/01/77
      ID#: 666
      Full Address: 15 Kings Road
   p2 instance values:
      Name: Jack, Second Name: Daniels, Age: 42, BirthDate: 01/01/77
      ID#: 666
      Full Address: 15 Kings Road
   p3 instance values:
      Name: Jack, Second Name: Daniels, Age: 42, BirthDate: 01/01/77
      ID#: 666
      Full Address: 15 Kings Road

Values of p1, p2 and p3 after changes to p1:
   p1 instance values:
      Name: Frank, Second Name: Simpson, Age: 32, BirthDate: 01/01/00
      ID#: 7878
      Full Address: 10 Wall street
   p2 instance values (reference values have changed):
      Name: Jack, Second Name: Daniels, Age: 42, BirthDate: 01/01/77
      ID#: 7878
      Full Address: 10 Wall street
   p3 instance values (everything was kept the same):
      Name: Jack, Second Name: Daniels, Age: 42, BirthDate: 01/01/77
      ID#: 666
      Full Address: 15 Kings Road
```

```
class Program
{
    static void Main(string[] args)
    {
        Person p1 = new Person();
        p1.Age = 42;
        p1.BirthDate = Convert.ToDateTime("1977-01-01");
        p1.Name = "Jack";
        // p1.SecondName = "Daniels";
        p1.IdInfo = new IdInfo(666);
        // p1.Address = new Address("Kings Road", 15);

        Person p2 = p1.ShallowCopy();
        Person p3 = p1.DeepCopy();

        Console.WriteLine("Original values of p1, p2, p3:");
        Console.WriteLine("   p1 instance values: ");
        DisplayValues(p1);
        Console.WriteLine("   p2 instance values:");
        DisplayValues(p2);
        Console.WriteLine("   p3 instance values:");
        DisplayValues(p3);

        p1.Age = 32;
        p1.BirthDate = Convert.ToDateTime("1900-01-01");
        p1.Name = "Frank";
        // p1.SecondName = "Simson"
        p1.IdInfo.IdNumber = 7878;
        // p1.Address = new Address("Wall street", 10);
        Console.WriteLine("\nValues of p1, p2 and p3 after changes to p1:");
        Console.WriteLine("   p1 instance values: ");
        DisplayValues(p1);
        Console.WriteLine("   p2 instance values (reference values have changed):");
        DisplayValues(p2);
        Console.WriteLine("   p3 instance values (everything was kept the same):");
        DisplayValues(p3);
    }
```

Singleton





Singleton

Название паттерна - Singleton/Одиночка.
Описан в работе [GoF95].

Цель паттерна

Гарантирует, что у класса есть только один экземпляр, и предоставляет единую точку доступа к нему [GoF95].

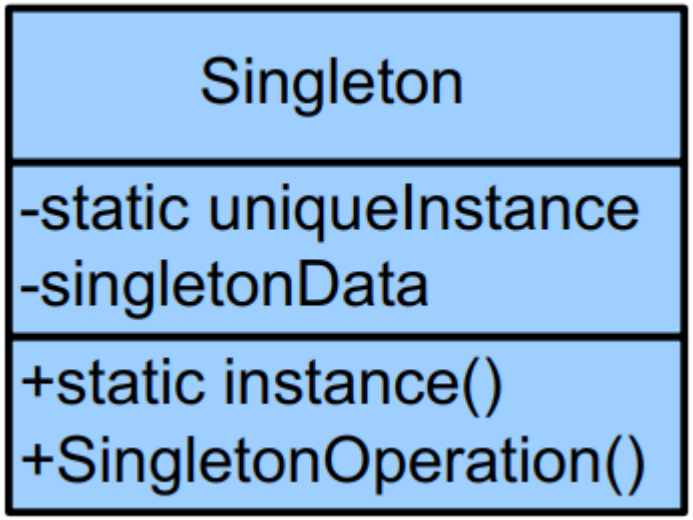
Паттерн следует использовать когда…

- Должен существовать только один экземпляр заданного класса, доступный всему клиентскому коду [GoF95].

Результаты использования паттерна

- Есть возможность полного контроля доступа клиента к единственному экземпляру. Что, например, может дать возможность подсчитать количество клиентских обращений или запрещать доступ в случае отсутствия соответствующих прав.
- Использование одиночки представляет более гибкие возможности, нежели статические функции классов или статические классы
- При реализации одиночки в многопоточной среде следует также учитывать тот факт, что два параллельных потока могут одновременно получать доступ к методу создания единственного экземпляра, что может привести к созданию более одного экземпляра одиночки. Во избежание ситуации «борьбы за ресурсы», следует код, ответственный за создание объекта, поместить в критическую секцию, предоставив тем самым доступ к нему только одному потоку.

Singleton




```
class Singleton
{
    private Singleton() { }
    private static Singleton _instance;
    public static Singleton GetInstance()
    {
        if (_instance == null)
        {
            _instance = new Singleton();
        }
        return _instance;
    }
    public static void someBusinessLogic()
    {
        // ...
    }
}

class Program
{
    static void Main(string[] args)
    {
        Singleton s1 = Singleton.GetInstance();
        Singleton s2 = Singleton.GetInstance();

        if (s1 == s2)
        {
            Console.WriteLine("Singleton works, bo
        }
        else
        {
            Console.WriteLine("Singleton failed, v
        }
    }
}
```

Singleton

Участники

Singleton – одиночка:

- Обеспечивает создание только одного экземпляра самого себя, ссылка на который сохраняется в статической переменной uniqueInstance, и глобальный доступ к нему через статический метод Instance()
- Запрещает клиентскому коду создавать собственные экземпляры, запретив ему доступ к своему конструктору (конструктор одиночки определяется как защищенный или приватный).

Singleton

Практические задания

- Реализовать пример, приведенный на слайде с определением Singleton