Современные средства разработки ПО

Шаблоны проектирования: введение, основные ШП

МГТУ им. Н.Э. Баумана, ИУ-6, Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru

Шаблоны проектирования

- Повторяемая структура взаимодействия объектов программы, представляющая собой решение задачи проектирования в рамках некоторого часто возникающего контекста
- Шаблоны проектирования связаны с ООП

Виды ШП

- Низкоуровневые — идиомы программирования. Связаны с устойчивыми решениями для одного или нескольких ЯП. Идиомы C++:
  - Pimpl — «pointer to implementation»,
  - swap — обмен значениями между переменными,
  - RAII — «Resource acquisition is initialization».
- **Шаблоны проектирования** — уровень взаимодействия объектов программы
- Архитектурные шаблоны — охватывают архитектуру ПО, т. е. взаимодействие между модулями и процессами

История

- Архитектор Кристофер Александр в 60х-70х годах издаёт ряд книг в области строительства зданий, оказавших сильное влияние на концепции построения ПО:
  - Notes on the Synthesis of Form (1964),
  - A Pattern Language (1968, 1977),
  - Houses Generated by Patterns (1969),
  - и др..
- Кент Бэк и Вард Каннингем использовали идеи шаблонов в строительстве для разработки ПО в 1988 году
- Джеймс Коплин опубликовал в 1991 году книгу Advanced C++ Idioms
- Эрих Гамма в 1994 году на базе своей докторской в соавторстве с Ричардом Хелмом, Ральфом Джонсоном и Джоном Влиссидесом пуликует книгу Design Patterns — Elements of Reusable Object-Oriented Software

Роль шаблонов проектирования

- Универсальный язык для обозначения приёмов проектирования ПО:
  - упрощает коммуникацию среди разработчиков;
  - упрощает обучение;
  - классифицирует приёмы проектирования, что упрощает их повторное использование.

Шаблоны проектирования — всего лишь идеи

- **ШП нужно использовать только после проектирования модели предметной области**
- ШП предназначены для решения вспомогательных задач организации внутренней структуры кода
- ШП не являются кубиками или пазлами — не нужно пытаться применять их буквально, скорее всего потребуется адаптация
- ШП не решения на все случаи — могут существовать более простые / оптимальные структуры

Шаблоны проектирования (основные)

- **Delegation pattern** – Объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту.
- **Functional design** – Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы (см. SRP).
- **Immutable interface** – Создание неизменяемого объекта.
- **Interface** – Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять (см. ISP).
- **Property container** – Позволяет добавлять дополнительные свойства для класса в контейнер (внутри класса), вместо расширения класса новыми свойствами.

Шаблоны проектирования (порождающие)

- **Abstract factory** – Класс, который представляет собой интерфейс для создания компонентов системы.
- **Builder** – Класс, который представляет собой интерфейс для создания сложного объекта.
- **Factory method** – Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс создавать.
- **Lazy initialization** – Объект, инициализируемый во время первого обращения к нему.
- **Prototype** – Определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор.
- **Singleton** – Класс, который может иметь только один экземпляр.
- **Multiton** – Гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним.

Шаблоны проектирования (структурные)

- **Adapter / Wrapper –** Объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс.
- **Composite (компоновщик) –** Объект, который объединяет в себе объекты, подобные ему самому.
- **Decorator –** Класс, расширяющий функциональность другого класса без использования наследования.
- **Facade –** Объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое.
- **Flyweight (приспособленец) –** Это объект, представляющий себя как уникальный экземпляр в разных местах программы, но фактически не являющийся таковым.
- **Proxy –** Объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него

Шаблоны проектирования (поведенческие) 1

- **Chain of responsibility (цепочка обязанностей)** – Предназначен для организации в системе уровней ответственности.
- **Command** – Представляет действие. Объект команды заключает в себе само действие и его параметры.
- **Interpreter** – Решает часто встречающуюся, но подверженную изменениям, задачу.
- **Iterator** – Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящих в состав агрегации.
- **Mediator (посредник)** – Обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга

Шаблоны проектирования (поведенческие) 2

- **Memento (хранитель)** – Позволяет не нарушая инкапсуляцию зафиксировать и сохранить внутренние состояния объекта так, чтобы позднее восстановить его в этих состояниях.
- **Null Object** – Предотвращает нулевые указатели, предоставляя объект «по умолчанию».
- **Observer (наблюдатель)** – Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.
- **Servant** – Используется для обеспечения общей функциональности группе классов.
- **Specification** – Служит для связывания бизнес-логики

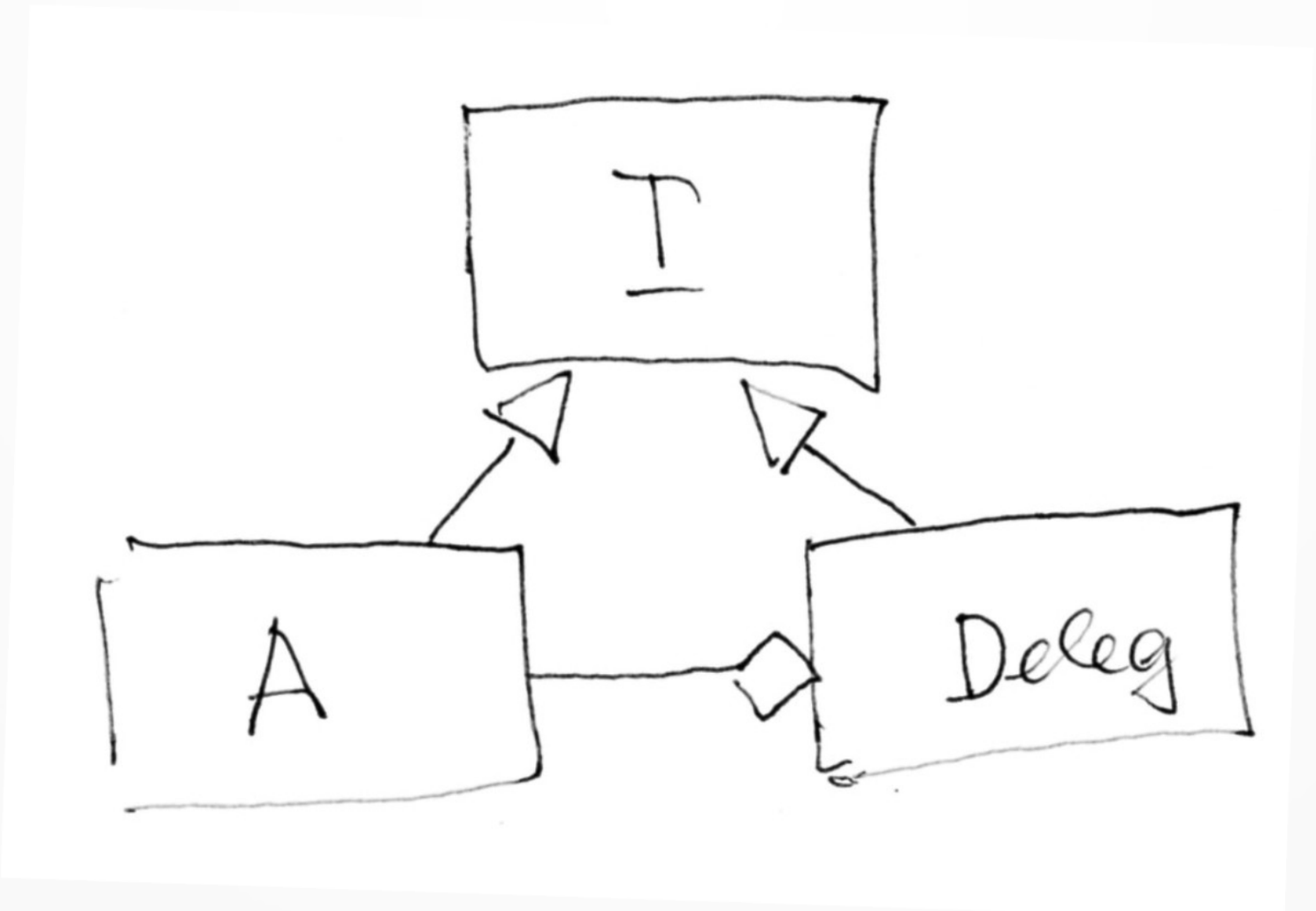
Шаблоны проектирования (поведенческие) 3

- **State** – Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.
- **Strategy** – Предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости.
- **Template method** – Определяет основу алгоритма и позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.
- **Visitor** – Описывает операцию, которая выполняется над объектами других классов. При изменении класса Visitor нет необходимости изменять обслуживаемые классы.

ШП Delegation (делегирование)

- Объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту
- Позволяет изменить поведение конкретного экземпляра объекта вместо создания нового класса путём наследования
- Недостаток: затрудняет оптимизацию по скорости в пользу чистоты абстракции

Диаграмма классов ШП Delegation (делегирование)



Пример реализации ШП Delegation (делегирование) 1

```
#include <iostream>

class I {
    public:
        virtual void f() = 0;
        virtual void g() = 0;
};

class A : public I {
    public:
        void f() { std::cout << "A: вызываем метод f()" << std::endl; }
        void g() { std::cout << "A: вызываем метод g()" << std::endl; }
};

class B : public I {
    public:
        void f() { std::cout << "B: вызываем метод f()" << std::endl; }
        void g() { std::cout << "B: вызываем метод g()" << std::endl; }
};
```

Пример реализации ШП Delegation (делегирование) 2

```
class C : public I {
   public:
    // Конструктор
     C() : m_i ( new A() ) { }
    // Деструктор
     virtual ~C() {
        delete m_i;
     }
     void f() { m_i->f(); }
     void g() { m_i->g(); }
    // Этими методами меняем поле-объект, чьи методы будем делегировать
     void toA() {
        delete m_i;
        m_i = new A();
     }
     void toB() {
        delete m_i;
        m_i = new B();
     }
   private:
    // Объявляем объект методы которого будем делегировать
     I * m_i;
};
```

Пример реализации ШП Delegation (делегирование) 3

```
int main() {
    C c;

    c.f();
    c.g();
    c.toB();
    c.f();
    c.g();

    return 0;
}

/* Output:
A: вызываем метод f()
A: вызываем метод g()
B: вызываем метод f()
B: вызываем метод g()
*/
```

Варианты ШП Delegation (делегирование)

- Шаблон делегирования является фундаментальной абстракцией, на основе которой реализованы другие шаблоны.

ШП Functional design (функциональный проект)

- ШП функциональный проект гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы.
- Функционально разработанные модули имеют низкое зацепление.
- Преимущества

Методика определения чистоты функционального проекта

- Если описание модуля (класса) включает связи, такие, как «и» либо «или», тогда проект имеет более чем одно предназначение, и соответственно возможно будет иметь побочные эффекты

ШП Immutable interface (неизменяемый интерфейс)

- Определяет интерфейсный класс, в котором определены методы, не изменяющие его состояние

Пример реализации ШП Immutable interface (неизменяемый интерфейс)

```cpp
#include <iostream>

class ImmutablePoint2D
{
public:
    virtual ~ImmutablePoint2D() = default;

    virtual int getX() const = 0;
    virtual int getY() const = 0;
};

class Point2D : public ImmutablePoint2D
{
public:
    Point2D(int x, int y) : _x(x), _y(y) {}

    virtual int getX() const override { return _x; }
    virtual int getY() const override { return _y; }

    void setX(int x) { _x = x; }
    void setY(int y) { _y = y; }

private:
    int _x, _y;
};

int main()
{
    ImmutablePoint2D * point = new Point2D(2,5);

    std::cout << point->getX() << " " << point->getY() << std::endl;
}
```

Особенности ШП Immutable interface (неизменяемый интерфейс)

- Плюс:
  - Четко передает намерения о неизменяемости класса
- Минус:
  - Для библиотечных классов необходимо предусматривать заранее
- Альтернатива:
  - Неизменяемая обёртка (immutable wrapper)

ШП Interface (интерфейс)

- Обеспечивает простой или более программно-специфический способ доступа к другим классам
- Интерфейс может:
  - содержать набор объектов и обеспечивать простую, высокоуровневую функциональность для программиста (например, ШП Фасад);
  - обеспечивать более чистый или более специфический способ использования сложных классов («класс-обёртка»);
  - использоваться в качестве «клея» между двумя различными API (ШП Адаптер);
  - и так далее.

ШП Property container (контейнер атрибутов/свойств)

- Обеспечивает возможность динамически расширять атрибутивный состав класса
- Это достигается путем добавления дополнительных атрибутов непосредственно самому объекту в ассоциативный контейнер, вместо расширения класса объекта новыми атрибутами

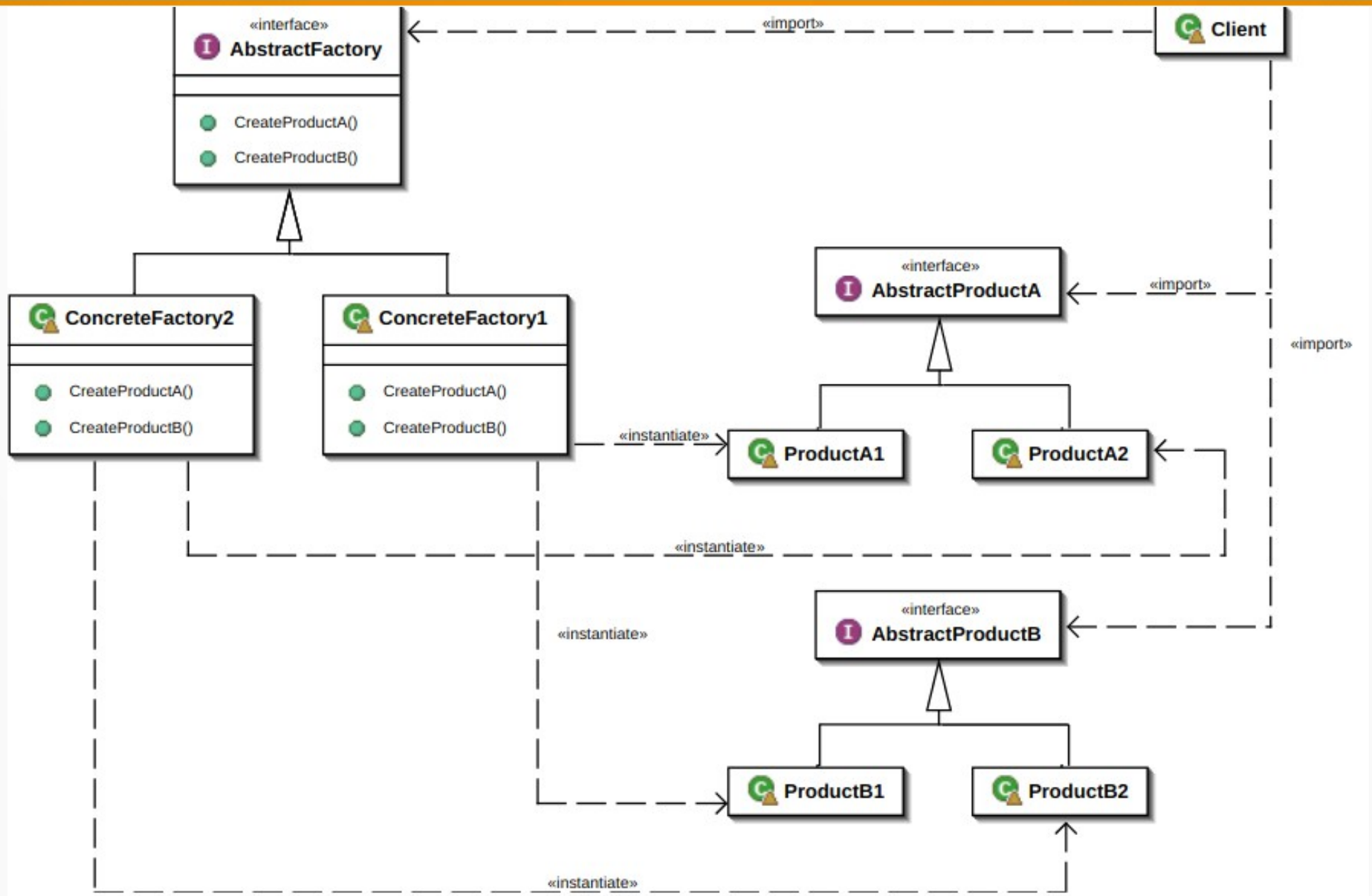
Особенности ШП Property container (контейнер атрибутов)

- Плюс:
  - Позволяет классу легко и быстро изменяться без изменения кода класса
- Минус:
  - Теряется строгая типизация
- В некоторых скриптовых ЯП на базе этого ШП построена инкапсуляция и даже весь ООП

ШП Abstract factory (абстрактная фабрика)

- Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов
- ШП реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки)
- Затем пишутся классы, реализующие этот интерфейс

ШП Abstract factory (абстрактная фабрика) Диаграмма классов



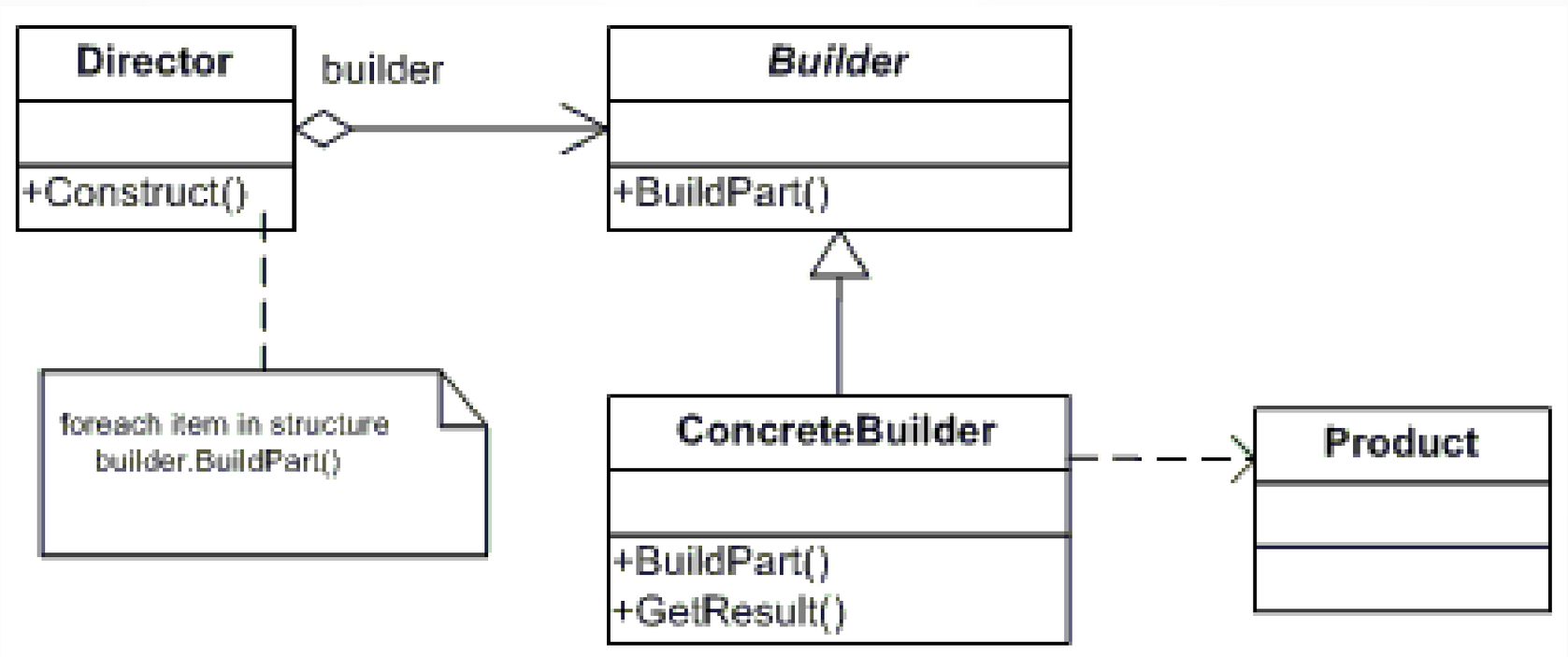
ШП Abstract factory (абстрактная фабрика) Особенности

- Плюсы:
  - изолирует конкретные классы;
  - упрощает замену семейств продуктов;
  - гарантирует сочетаемость продуктов.
- Минус:
  - сложно добавить поддержку нового вида продуктов.
- Ограничения:
  - Система не должна зависеть от того, как создаются, компонуются и представляются входящие в неё объекты.
  - Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения.
  - Система должна конфигурироваться одним из семейств составляющих её объектов.
  - Требуется предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.

ШП Builder (строитель)

- Порождающий шаблон проектирования предоставляет способ создания составного объекта
- Отделяет конструирование сложного объекта от его представления так, что в результате одного и того же процесса конструирования могут получаться разные представления

ШП Builder (строитель)
Диаграмма классов



ШП Builder (строитель)
Особенности

- Плюсы:
  - позволяет изменять внутреннее представление продукта;
  - изолирует код, реализующий конструирование и представление;
  - дает более тонкий контроль над процессом конструирования.
- Ограничения:
  - алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой;
  - процесс конструирования должен обеспечивать различные представления конструируемого объекта.

ШП Builder (строитель)
Примеры

- Переводчик с разных языков
- Сохранение текста графического редактора в различных форматах

Вопросы?