Паттерны

Впервые

Гамма, Хелм, Джонсон и Влиссидес

изд. «Питер», 2003

«Паттерны проектирования»

Банда четырех

Основной принцип паттерна

Введение новых уровней абстракции

Изменения ≠ побочные эффекты

Выявление вектора изменений

Инкапсуляция изменений

Примеры
(реализация на уровне компилятора)

- Наследование
- Композиция

Примеры

- Итератор

Фундаментальный принцип БЧ

Аксиома:

«Отдавайте предпочтение композиции объектов перед наследованием классов»

Канон экстремального программирования

- Выберите самое простое решение, которое может работать

Классификация паттернов

23 паттерна БЧ

- Паттерны создания объектов
- Структурные паттерны
- Поведенческие паттерны

Паттерны создания объектов

- Синглет
- Фабрика
- Строитель

Структурные паттерны

- Посредник
- Адаптер

Поведенческие паттерны

- Команда
- Шаблонный метод
- Состояние
- Стратегия
- Цепочка ответственности
- Наблюдатель
- Множественная диспетчеризация
- Посетитель

Design Patterns Library

http://hillside.net/patterns

Новые паттерны

- Трудности идентификации и описания
- Нет единого определения

Неудачные примеры

- Конструкторы, деструкторы:
  - Паттерн - Гарантированная инициализация и зачистки
  - Не Паттерн – Недостаточно содержательны
- Геттеры, сеттеры в JavaBeans

Паттерн Посыльный (Messenger)

```
class Point {
public: int x, y, z;
Point(int xi. int yi. int zi) : x(xi). y(yi). z(zi) {}
Point(const Points p) : x(p.x). y(p.y). z(p.z) {}
Point & operator=(const Points rhs) {
x = rhs.x:
y = rhs.y:
z = rhs.z:
return *this;

friend ostream & operator (ostreamS os. const Points p) {
return os<< “x=“<< p.x<<“y =“<<p.y<<“z=“<< p.z;
}
};
```

Паттерн Посыльный (Messenger)

```
class Vector { //Математический вектор
public:
int magnitude, direction;
Vector(int m. int d) : magnitude(m). direction(d) {}
};
```

Паттерн Посыльный (Messenger)

```
class Space {
public:
static Point translate(Point p. vector v) {
 p.x += V.magnitude + v.direction;
 p.y += V.magnitude + v.direction;
 p.z += V.magnitude + v.direction;
 return p;
}
};
```

Паттерн Посыльный (Messenger)

```
int main() {
 Point p1(2. 3);
 Point p2 = Space::translate(p1, Vector(11, 47));
 cout <<p1  << p2<< endl;
 }
```

Паттерн Накопитель (Collecting parameter)

```
class CollectingParameter : public vector<string> {};
class Filler {
public:
void f(CollectingParameter& cp) {
    cp.push_back( “accumulating” );
}
void g(CollectingParameter& cp) {
    cp.push_back( “items” );
}
void f(CollectingParameter& cp) {
    cp.push_back( “as we go” );
}
};
```

Паттерн Накопитель (Collecting parameter)

```
int main() {
Filler filler;
CollectingParameter cp;
filler.f(cp);
filler.g(cp);
filler.h(cp);
vector<string>::iterator it = cp.begin();
while(it != cp.end())
    cout *it++ ;
cout << endl;
 }
```

Паттерн Синглет
и его разновидности

```
class Singleton {
    static Singleton s;
    int i;
    Singleton(int x) : i(x) {}
    Singleton& operator=(Singleton&);
    Singleton(const Singleton&);
```

Паттерн Синглет
и его разновидности

```
public:
    static Singleton& instance() { return s; }
    int getValue() { return i; }
    void setValue(int x) { i = x; }
};
```

Паттерн Синглет
и его разновидности

```
Singleton Singleton::s(47);
int main() {
Singleton& s = Singleton::instance();
cout<<s.getValue()<< endl;
Singleton& s2 = Singleton::instance();
s2.setValue(9);
cout<<s.getValue()<< endl;
 }
```

Отложенная инициализация - Синглет

```
#ifndef LOGFILE_H
#define LOGFILE_H
#include <fstream>
std:: ofstream& logfile();
# endif
```

Отложенная инициализация - Синглет

```
#include “LogFile.h”
std: :ofstream& logfile() {
static std::ofstream log(“Logfile.log”);
return log;
}
```

Отложенная инициализация - Синглет

```
# ifndef USEL0G1_H
#define USEL0G1_H
void f();
#endif

#include “UseLogl.h”
#include “LogFile.h”
void f() {
Logfile()<<std::endl;
}
```

Одиночка (Singleton)

- **Проблема** Необходим лишь один экземпляр специального класса, различные объекты должны обращаться к этому экземпляру через единственную точку доступа.

- **Решение** Создать класс и определить статический метод класса, возвращающий этот единственный объект.

- **Рекомендации** Разумнее создавать именно статический экземпляр специального класса, а не объявить требуемые методы статическими, поскольку при использовании методов экземпляра можно применить механизм наследования и создавать подклассы. Статические методы в языках программирования не полиморфны и не допускают перекрытия в производных классах.
  Решение на основе создания экземпляра является более гибким.

Абстрактная фабрика (Abstract Factory, Factory)

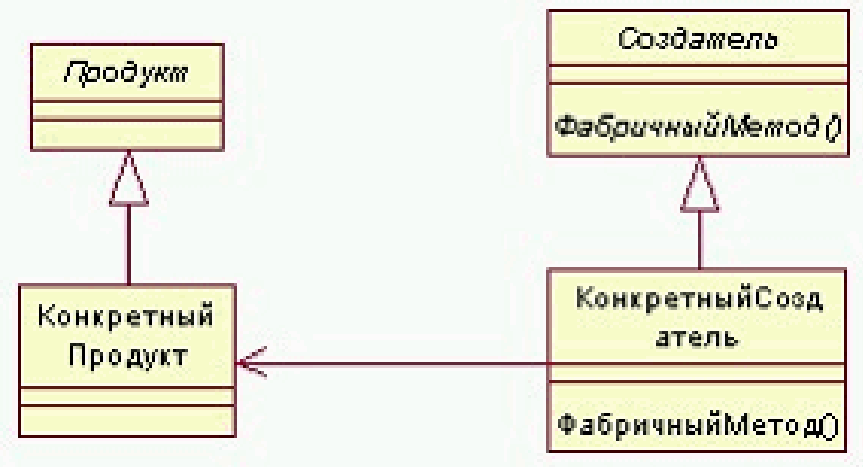
- **Проблема** Создать семейство взаимосвязанных или взаимозависимых объектов (не специфицируя их конкретных классов).
- **Решение** Создать абстрактный класс, в котором объявлен интерфейс для создания конкретных классов.
- **Преимущества** Изолирует конкретные классы. Поскольку "Абстрактная фабрика" инкапсулирует ответственность за создание классов и сам процесс их создания, то она изолирует клиента от деталей реализации классов. Упрощена замена "Абстрактной фабрики", поскольку она используется в приложении только один раз при инстанцировании.
- **Недостатки** Интерфейс "Абстрактной фабрики" фиксирует набор обьектов, которые можно создать. Расширение "Абстрактной фабрики" для изготовления новых объектов часто затруднительно.

Прототип (Prototype)

- **Проблема** Система не должна зависеть от того, как в ней создаются, компонуются и представляются объекты.
- **Решение** Создавать новые объекты с помощью паттерна - прототипа. "Прототип" объявляет интерфейс для клонирования самого себя. "Клиент" создает новый объект, обращаясь к "Прототипу" с запросом клонировать "Прототип".

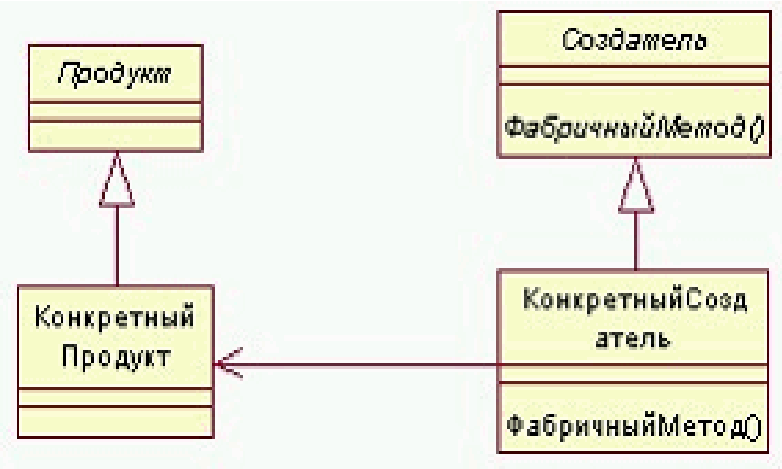
(Фабричный метод) Factory Method или Виртуальный конструктор (Virtual Constructor)

- **Проблема** Определить интерфейс для создания объекта, но оставить подклассам решение о том, какой класс инстанцировать, то есть, делегировать инстанцирование подклассам.



(Фабричный метод) Factory Method или Виртуальный конструктор (Virtual Constructor)

- **Решение** Абстрактный класс "Создатель" объявляет ФабричныйМетод, возвращающий объект типа "Продукт" (абстрактный класс, определяющий интерфейс объектов, создаваемых фабричным методом). "Создатель " также может определить реализацию по умолчанию ФабричногоМетода, который возвращает "КонкретныйПродукт". "КонкретныйСоздатель" замещает ФабричныйМетод, возвращающий объект "КонкретныйПродукт". "Создатель" "полагается" на свои подклассы в определении ФабричногоМетода, возвращающего объект "КонкретныйПродукт".



(Фабричный метод) Factory Method или Виртуальный конструктор (Virtual Constructor)

- **Преимущества** Избавляет проектировщика от необходимости встраивать в код зависящие от приложения классы.
- **Недостатки** Возникает дополнительный уровень подклассов.