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*

Фабричный метод

Фабричный метод (также известен как **Виртуальный конструктор**, Factory Method) – это **порождающий паттерн** проектирования, который определяет общий интерфейс для создания объектов в суперклассе(*базовом*), позволяя подклассам(*наследниках*) изменять тип создаваемых объектов.

• Для того, чтобы система оставалась **независимой** от ***различных типов объектов***, паттерн **Factory Method** использует ***механизм полиморфизма*** т.е. классы всех конечных типов наследуют от одного абстрактного базового класса, предназначенного для полиморфного использования.

• В этом базовом классе определяется ***единый интерфейс***, через который пользователь будет оперировать объектами конечных типов.

• Для обеспечения относительно простого добавления в систему новых типов паттерн Factory Method локализует создание объектов конкретных типов в специальном классе-фабрике.

• ***Методы этого класса***, посредством которых создаются объекты конкретных классов, ***называются фабричными***.

Существуют две разновидности паттерна Factory Method:

- Обобщенный конструктор;
- Классическая реализация;

Фабричный метод (Factory Method) - это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах.

То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

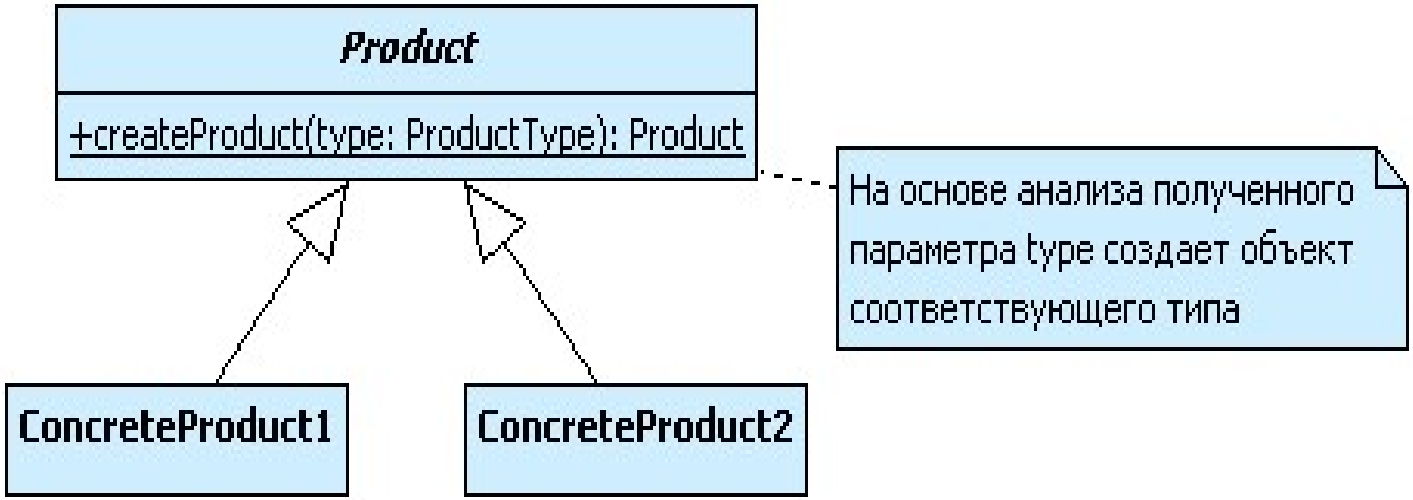
Паттерн применяется когда

1. Когда ***заранее неизвестно***, объекты каких типов необходимо создавать;

2. Когда система ***должна быть независимой*** от ***процесса создания новых объектов и расширяемой***: в нее можно легко вводить новые классы, объекты которых система должна создавать;

3. Когда создание новых объектов необходимо делегировать из базового класса классам наследникам;

Реализация паттерна Factory Method: *обобщенный конструктор*



Реализация паттерна Factory Method:*обобщенный конструктор*

```
enum TypesObjects {ObjectX_ID = 0,ObjectY_ID,ObjectZ_ID };

class MainObject
{
    public:
    virtual void info() = 0;
    virtual ~MainObject() {}
    // Параметризированный статический фабричный метод
    static MainObject* createObject( TypesObjects id );
};
```

Реализация паттерна Factory Method:*обобщенный конструктор*

```
class ObjectX:public MainObject
{
    public:
    void info(){
        cout << "This is ObjectX" << endl;
    }
};

class ObjectY:public MainObject
{
    public:
    void info(){
        cout << "This is ObjectY" << endl;
    }
};

class ObjectZ: public MainObject ……
```

Реализация паттерна Factory Method:*обобщенный конструктор*

```
MainObject* MainObject::createObject(TypesObjects id)
{
    MainObject * p = nullptr;
    switch (id)
    {
        case ObjectX_ID:
        p = new ObjectX();
            break;
        case ObjectY_ID:
            p = new ObjectY();
            break;
        case ObjectZ_ID:
            p = new ObjectZ();
            break;

    }
    return p;
}
```

Реализация паттерна Factory Method:*обобщенный конструктор*

```
int main(int argc, char** argv) {

vector<MainObject*> vectorObj;
vectorObj.push_back(MainObject::createObject(ObjectX_ID));
vectorObj.push_back(MainObject::createObject(ObjectY_ID));
vectorObj.push_back(MainObject::createObject(ObjectZ_ID));

for (int i = 0; i < vectorObj.size(); i++)
vectorObj[i]->info();
return 0;
}
```

Представленный вариант паттерна Factory Method пользуется популярностью благодаря своей простоте.

В нем статический фабричный метод createObject() определен непосредственно в полиморфном базовом классе MainObject.

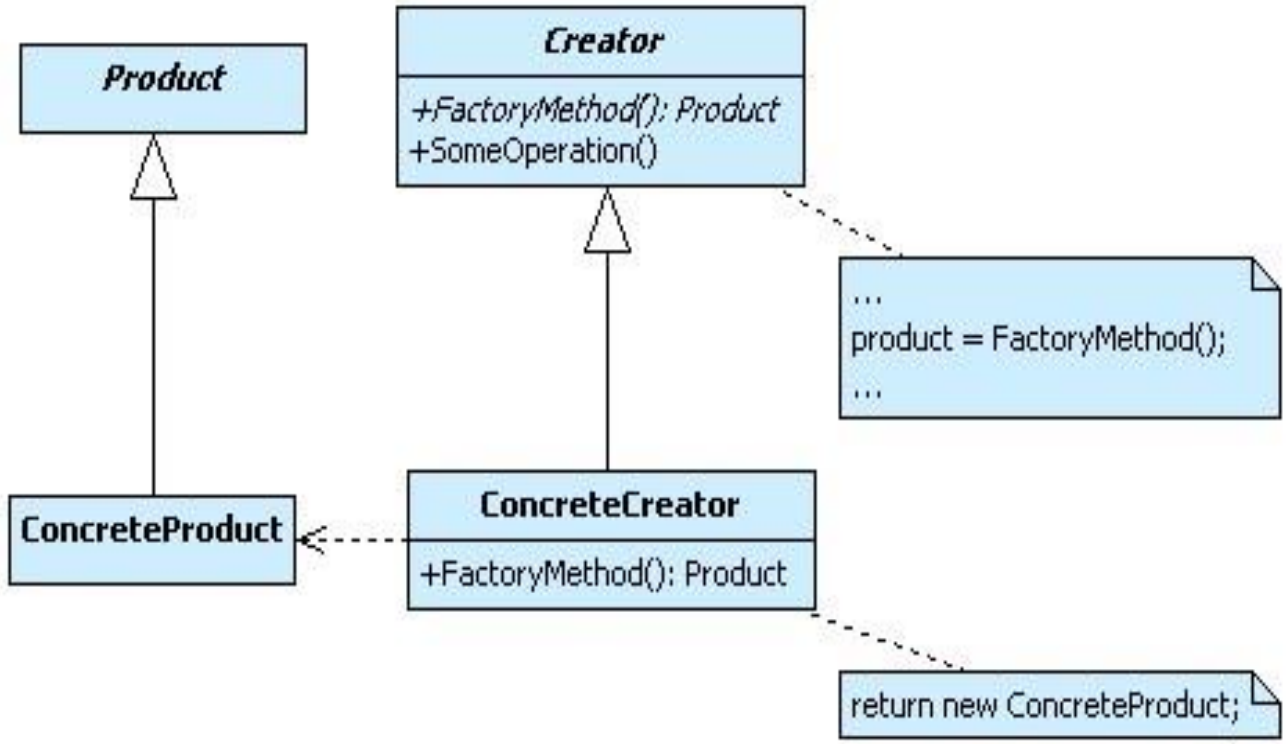
Этот фабричный метод является параметризированным, то есть для создания объекта некоторого типа в createObject() передается соответствующий идентификатор типа.

С точки зрения "чистоты" объектно-ориентированного кода у этого варианта есть следующие недостатки:

- Так как код по созданию объектов всех возможных типов сосредоточен в статическом фабричном методе класса createObject(), то базовый класс MainObject обладает знанием обо всех производных от него классах, что является нетипичным для объектно-ориентированного подхода.
- Подобное использование оператора switch (как в коде фабричного метода createObject()) в объектно-ориентированном программировании также не приветствуется.

Указанные недостатки отсутствуют в классической реализации паттерна Factory Method.

Классическая Реализация паттерна Factory Method



```
#include<assert.h>
#include<iostream>
#include<vector>

class MainObject
{
    public:
    virtual void info() = 0;
    virtual ~MainObject() {}
};
class ObjectX:public MainObject
{
    public:
    void info(){
        cout << "This is ObjectX" << endl;
    }
};
class ObjectY:public MainObject {};
class ObjectZ:public MainObject {};
```

```
#include<assert.h>
#include<iostream>
#include<vector>

class FactoryObj
{
public:
   virtual MainObject* createObject() = 0;
   virtual ~FactoryObj(){}
};
class FactoryObjX: public FactoryObj
{
 public:
    MainObject*createObject(){

    return new ObjectX();
    }

class FactoryObjY: public FactoryObj{};
class FactoryObjZ: public FactoryObj{};
```

```
#include<assert.h>
#include<iostream>
#include<vector>

int main(int argc, char** argv) {

FactoryObjX ObjXCreator;
FactoryObjY ObjYCreator;
FactoryObjZ ObjZCreator;

vector<MainObject*> vectorObj;

vectorObj.push_back( ObjXCreator.createObject());
vectorObj.push_back( ObjYCreator.createObject());
vectorObj.push_back( ObjZCreator.createObject());

for(int i=0; i<vectorObj.size(); i++){
    vectorObj[i]->info();
    //...Do Something
    delete vectorObj[i];
}
return 0;
}
```

Классический вариант паттерна Factory Method использует идею полиморфной фабрики.

Специально выделенный для создания объектов полиморфный базовый класс ***FactoryObj*** объявляет интерфейс фабричного метода **createObject(),** а производные классы его реализуют.

Представленный вариант паттерна Factory Method является наиболее распространенным, но не единственным. Возможны следующие вариации:

Класс ***FactoryObj*** имеет реализацию фабричного метода **createObj()** по умолчанию.

Фабричный метод **createObj()** класса ***FactoryObj*** параметризирован типом создаваемого объекта (как и у представленного ранее, простого варианта Factory Method) и имеет реализацию по умолчанию.

В этом случае, производные от ***FactoryObj*** классы необходимы лишь для того, чтобы определить нестандартное поведение **createObj()**.

Достоинства паттерна Factory Method

- Создает объекты разных типов, позволяя системе оставаться независимой как от самого процесса создания, так и от типов создаваемых объектов.

Недостатки паттерна Factory Method

- В случае классического варианта паттерна даже для порождения единственного объекта необходимо создавать соответствующую фабрику