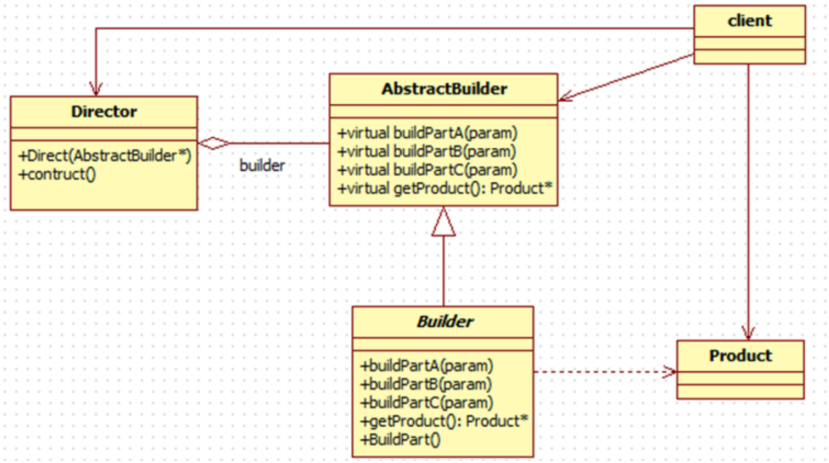
动机（Motivation）:在软件系统中，有时候面临着“一个复杂对象”的创建工作，其通常由各个部分的子对象用一定的算法构成；由于需求的变化，这个复杂对象的各个部分经常面临着剧烈的变化，但是将它们组合在一起的算法却相对稳定。如何应对这种变化？如何提供一种“封装机制”来隔离出“复杂对象的各个部分”的变化，从而保持系统中的“稳定构建算法”不随着需求改变而改变？

定义:将一个复杂对象的构建与其表示相分离，使得同样的构建过程(稳定)可以创建不同的表示(变化)。——《设计模式》GoF



要点总结:Builder 模式主要用于“分步骤构建一个复杂的对象”。在这其中“分步骤”是一个稳定的算法，而复杂对象的各个部分则经常变化。变化点在哪里，封装哪里—— Builder模式主要在于应对“复杂对象各个部分”的频繁需求变动。其缺点在于难以应对“分步骤构建算法”的需求变动.

```
#include <iostream>
#include <string>
using namespace  std;
class Vehicle{
public:
    Vehicle(){
        strBody = "";
        strTyre = "";
        strEngine = "";
    }
    void Show(){
        if(strBody == "" || strTyre == "" || strEngine == ""){
            cout << "车辆还未建造" << endl;
            return;
        }
        cout << "车辆建造完成: " << strBody << ", " << strTyre << ", " << strEngine << endl;
    }
public:
    string strBody;//车身,轮胎,引擎
    string strTyre;
    string strEngine;
};
class BuilderVehicle{
public:
    virtual ~BuilderVehicle(){}
    virtual void BuildBody() = 0;
    virtual void BuildTyre() = 0;
    virtual void BuildEngine() = 0;
};
class BuilderSUV : public BuilderVehicle{
private:
```

```
    Vehicle * m_pVehicle;
public:
    BuilderSUV(Vehicle *pV) : m_pVehicle(pV){}
    void BuildBody(){
        m_pVehicle->strBody = "SUV 车身";
    }
    void BuildTyre(){
        m_pVehicle->strTyre = "SUV 轮胎";
    }
    void BuildEngine(){
        m_pVehicle->strEngine = "SUV 引擎";
    }
};
class BuilderTruck : public BuilderVehicle{
private:
    Vehicle * m_pVehicle;
public:
    BuilderTruck(Vehicle *pV) : m_pVehicle(pV){}
    void BuildBody(){
        m_pVehicle->strBody = "卡车 车身";
    }
    void BuildTyre(){
        m_pVehicle->strTyre = "卡车 轮胎";
    }
    void BuildEngine(){
        m_pVehicle->strEngine = "卡车 引擎";
    }
};
class Director{
public:
    Director(){}
    void Build(BuilderVehicle *pBuild){
        pBuild->BuildBody();
        pBuild->BuildTyre();
        pBuild->BuildEngine();
    }
};
int main(){
    Vehicle V;
    BuilderTruck Truck(&V);
    Director D;
    D.Build(&Truck);
    V.Show();

    BuilderSUV SUV(&V);
    D.Build(&SUV);
    V.Show();
    return 0;
}
```

```
#include <iostream>
#include <vector>
#include <string>
```

```cpp
using namespace std;
class Product{
private:
    vector<string> vecParts;
public:
    void AddAPart(const string strPart){
        vecParts.push_back(strPart);
    }
    virtual void Show() {
        for(int i = 0; i<vecParts.size(); i++){
            cout << vecParts[i] <<  ", ";
        }
        vecParts.clear();
        cout << endl;
    }
};
class Builder{
public:
    virtual void BuildHead() = 0;
    virtual void BuildBody() = 0;
    virtual void BuildHand() = 0;
    virtual void BuildFeet() = 0;
    virtual ~Builder(){}
    virtual Product* GetProduct() = 0;
};
class FatPersonBuilder : public Builder{
private:
    Product * m_pProduct;
public:
    FatPersonBuilder(Product* pPro):m_pProduct(pPro){}
    void BuildHead() {
        m_pProduct->AddAPart("胖子的头");
    }
    void BuildBody() {
        m_pProduct->AddAPart("胖子的身体");
    }
    void BuildHand() {
        m_pProduct->AddAPart("胖子的手");
    }
    void BuildFeet() {
        m_pProduct->AddAPart("胖子的脚");
    }
    Product* GetProduct(){
        return m_pProduct;
    }
};
class ThinPersonBuilder : public Builder{
private:
    Product * m_pProduct;
public:
    ThinPersonBuilder(Product* pPro):m_pProduct(pPro){}
    void BuildHead() {
        m_pProduct->AddAPart("瘦子的头");
    }
```

```
    void BuildBody() {
        m_pProduct->AddAPart("瘦子的身体");
    }
    void BuildHand() {
        m_pProduct->AddAPart("瘦子的手");
    }
    void BuildFeet() {
        m_pProduct->AddAPart("瘦子的脚");
    }
    Product* GetProduct(){
        return m_pProduct;
    }
};
class Director{
public:
    Director(){}
    void CreatPerson(Builder *pBuilder)
    {
        pBuilder->BuildHead();
        pBuilder->BuildBody();
        pBuilder->BuildHand();
        pBuilder->BuildFeet();
    }
};
int main(){
    Product objProduct;
    FatPersonBuilder *pFat = new FatPersonBuilder(&objProduct);
    ThinPersonBuilder *pThin = new ThinPersonBuilder(&objProduct);
    Director objDirector;
    objDirector.CreatPerson(pFat);
    pFat->GetProduct()->Show();
    objDirector.CreatPerson(pThin);
    pThin->GetProduct()->Show();
    delete pFat;
    delete pThin;
    return 0;
}
```

```
192:DesignPattnsStudy weishichun$ ls Builder
Builder_1.cpp      Builder_2.cpp      Builder构建器.pdf
192:DesignPattnsStudy weishichun$ g++ -o Builder1.out Builder_1.cpp
192:DesignPattnsStudy weishichun$ ./Builder1.out
车辆建造完成：卡车 车身，卡车 轮胎，卡车 引擎
车辆建造完成：SUV 车身，SUV 轮胎，SUV 引擎
192:DesignPattnsStudy weishichun$ g++ -o Builder2.out Builder_2.cpp
192:DesignPattnsStudy weishichun$ ./Builder2.out
胖子的头，胖子的身体，胖子的手，胖子的脚，
瘦子的头，瘦子的身体，瘦子的手，瘦子的脚，
192:DesignPattnsStudy weishichun$
```