3.3.5 抽象工厂模式作业

设计一个电脑主板架构，电脑包括（显卡，内存，CPU）3 个固定的插口，显卡具有显示功能（display，功能实现只要打印出意义即可）， 内存具有存储功能（storage），cpu 具有计算功能（calculate）。

现有 Intel 厂商，nvidia 厂商，Kingston 厂商，均会生产以上三种硬件。要求组装两台电脑：

1 台（Intel 的 CPU，Intel 的显卡，Intel 的内存）

1 台（Intel 的 CPU， nvidia 的显卡，Kingston 的内存）

用抽象工厂模式实现。

参考源码：

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
using namespace std;

//抽象显卡
class AbstractGraphics{
public:
    virtual void work() = 0;
};

//因特尔显卡
class IntelGraphics : public AbstractGraphics{
public:
    virtual void work(){
        cout << "因特尔显卡开始工作..." << endl;
    }
};

//英伟达显卡
```

```
class NvidiaGraphics : public AbstractGraphics{
public:
    virtual void work(){
        cout << "英伟达显卡开始工作..." << endl;
    }
};

//金士顿显卡
class KingstonGraphics : public AbstractGraphics{
public:
    virtual void work(){
        cout << "金士顿显卡开始工作..." << endl;
    }
};

//抽象 CPU
class AbstractCPU{
public:
    virtual void work() = 0;
};

//因特尔 CPU
class IntelCPU : public AbstractCPU{
public:
    virtual void work(){
        cout << "因特尔 CPU 开始工作..." << endl;
    }
};

//英伟达 CPU
class NvidiaCPU : public AbstractCPU{
public:
    virtual void work(){
        cout << "英伟达 CPU 开始工作..." << endl;
    }
};

//金士顿 CPU
class KingstonCPU : public AbstractCPU{
public:
    virtual void work(){
        cout << "金士顿 CPU 开始工作..." << endl;
    }
};
```

```
//抽象内存
class AbstractMemory{
public:
    virtual void work() = 0;
};

//因特尔内存
class IntelMemory : public AbstractMemory{
public:
    virtual void work(){
        cout << "因特尔内存开始工作..." << endl;
    }
};

//英伟达内存
class NvidiaMemory : public AbstractMemory{
public:
    virtual void work(){
        cout << "英伟达内存开始工作..." << endl;
    }
};

//金士顿内存
class KingstonMemory : public AbstractMemory{
public:
    virtual void work(){
        cout << "金士顿内存开始工作..." << endl;
    }
};

//计算机
class Computer{
public:
    Computer(){
        this->pCPU = NULL;
        this->pGraphics = NULL;
        this->pMemory = NULL;
    }
    void setCPU(AbstractCPU* cpu){
        pCPU = cpu;
    }
    void setGraphics(AbstractGraphics* graphics){
        pGraphics = graphics;
```

```
    }
    void setMemory(AbstractMemory* memory){
        pMemory = memory;
    }

    //启动电脑
    void run(){
        if (NULL != pCPU){
            this->pCPU->work();
        }
        if (NULL != this->pGraphics){
            this->pGraphics->work();
        }
        if (NULL != this->pMemory){
            this->pMemory->work();
        }
    }
private:
    AbstractCPU* pCPU;
    AbstractGraphics* pGraphics;
    AbstractMemory* pMemory;
};


//抽象工厂
class AbstractFactory{
public:
    virtual AbstractCPU* CreateCPU() = 0;
    virtual AbstractGraphics* CreateGraphics() = 0;
    virtual AbstractMemory* CreateMemory() = 0;
};

//生产第一种电脑的工厂
class FirstComputerFactory : public AbstractFactory{
public:
    virtual AbstractCPU* CreateCPU(){
        return new IntelCPU;
    }
    virtual AbstractGraphics* CreateGraphics(){
        return new IntelGraphics;
    }
    virtual AbstractMemory* CreateMemory(){
        return new IntelMemory;
    }
```

```
};

//生产第二种电脑的工厂
class SecondComputerFactory : public AbstractFactory{
public:
    virtual AbstractCPU* CreateCPU(){
        return new IntelCPU;
    }
    virtual AbstractGraphics* CreateGraphics(){
        return new NvidiaGraphics;
    }
    virtual AbstractMemory* CreateMemory(){
        return new KingstonMemory;
    }
};

void test01(){

    Computer* computer = new Computer;
    AbstractFactory* factory = NULL;
    AbstractCPU* cpu = NULL;
    AbstractGraphics* graphics = NULL;
    AbstractMemory* memory = NULL;


    //创建生产零件的工厂
    factory = new FirstComputerFactory;
    cpu = factory->CreateCPU();
    graphics = factory->CreateGraphics();
    memory = factory->CreateMemory();

    //组装第一台电脑
    computer->setCPU(cpu);  //安装 CPU
    computer->setGraphics(graphics); //安装显卡
    computer->setMemory(memory); //安装内存
    computer->run(); //启动电脑

    delete memory;
    delete graphics;
    delete cpu;
    delete factory;

    cout << "------------------------" << endl;
    factory = new SecondComputerFactory;
```

```
    cpu = factory->CreateCPU();
    graphics = factory->CreateGraphics();
    memory = factory->CreateMemory();

    //组装第二台电脑
    computer->setCPU(cpu);  //安装 CPU
    computer->setGraphics(graphics); //安装显卡
    computer->setMemory(memory); //安装内存
    computer->run(); //启动电脑

    delete memory;
    delete graphics;
    delete cpu;
    delete factory;

    delete computer;

}

int main(){

    test01();

    system("pause");
    return EXIT_SUCCESS;
}
```

5.2.3 命令模式练习

联想路边撸串烧烤场景， 有烤羊肉，烧鸡翅命令，有烤串师傅，和服务员MM。根据命令模式，设计烤串场景。

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<list>
using namespace std;

//烤串大师
```

```
class SkewerMaster{
public:
    void MakeChickenWings(){
        cout << "烤鸡翅!" << endl;
    }
    void MakeMutton(){
        cout << "烤羊肉!" << endl;
    }
};


//抽象烤串命令
class AbstractSkewerCommand{
public:
    virtual void Make() = 0;
};

//烤鸡翅的命令
class MakeChickenWingsCommand : public AbstractSkewerCommand{
public:
    MakeChickenWingsCommand(SkewerMaster* master){
        pMaster = master;
    }
    virtual void Make(){
        pMaster->MakeChickenWings();
    }
private:
    SkewerMaster* pMaster;
};

//烤羊肉的命令
class MakeMuttonCommand : public AbstractSkewerCommand{
public:
    MakeMuttonCommand(SkewerMaster* master){
        pMaster = master;
    }
    virtual void Make(){
        pMaster->MakeMutton();
    }
private:
    SkewerMaster* pMaster;
};
```

```
//服务员
class Waiter{
public:
	void addCommand(AbstractSkewerCommand* command){
		m_list.push_back(command);
	}
	void submitCommand(){
		for (list<AbstractSkewerCommand*>::iterator it = m_list.begin(); it != m_list.end(); it ++){
			(*it)->Make();
		}
	}
private:
	list<AbstractSkewerCommand*> m_list;
};

//测试
void test01(){

	//创建烧烤师傅
	SkewerMaster* master = new SkewerMaster;
	//创建烧烤命令
	AbstractSkewerCommand* command1 = new MakeChickenWingsCommand(master);
	AbstractSkewerCommand* command2 = new MakeMuttonCommand(master);
	//创建服务员
	Waiter* waiter = new Waiter;
	waiter->addCommand(command1);
	waiter->addCommand(command2);
	//服务员批量提交命令
	waiter->submitCommand();

	delete waiter;
	delete command2;
	delete command1;
	delete master;
}

int main(){

	test01();

	system("pause");
	return EXIT_SUCCESS;
}
```

5.3.3 策略模式练习题

商场促销有策略 A（0.8 折） 策略 B（消费满 200，返现 100），用策略模式模拟场景。

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
using namespace std;

//抽象策略类
class AbstractStrategy{
public:
    virtual int CaculateMoney(int) = 0;
};

//8 折策略
class StrategySaleByEight : public AbstractStrategy{
public:
    virtual int CaculateMoney(int money){
        return money * 0.8;
    }
};

//满 200 返现 100 策略
class StrategySale200Return100 : public AbstractStrategy{
public:
    virtual int CaculateMoney(int money){
        return money - (money / 200) * 100;
    }
};

//超市购物类
class Shopping{
public:
    Shopping(){
        pStrategy = NULL;
    }
    void setStrategy(AbstractStrategy* strategy){
        pStrategy = strategy;
    }
```

```
    void PayMoney(int money){
        int realMoney = 0;
        if (NULL == pStrategy){
            realMoney = money;
        }
        else{
            realMoney = pStrategy->CaculateMoney(money);
        }
        cout << "商品折前价格:" << money << ",折后价格:" << realMoney << "元!" << endl;
    }
private:
    AbstractStrategy* pStrategy;
};

void test01(){

    Shopping* shopping = new Shopping;
    AbstractStrategy* strategy = NULL;
    cout << "逢活动 八折优惠" << endl;
    strategy = new StrategySaleByEight;
    shopping->setStrategy(strategy); //设置商场活动为 8 折优惠
    shopping->PayMoney(600);

    delete strategy;

    cout << "再次逢活动 满 200 返现 100" << endl;
    strategy = new StrategySale200Return100;
    shopping->setStrategy(strategy);
    shopping->PayMoney(900);

    delete strategy;
    delete shopping;

}

int main(){

    test01();

    system("pause");
    return EXIT_SUCCESS;
}
```

5.4.3 观察者模式练习题

江湖中有多个帮派，还有一名无人不知，无事不晓的百晓生。当江湖中发生武林打斗事件，百晓生作为天生的大嘴巴会广播武林消息，每个帮派的门第对于事件的处理方式均不同，同帮派被欺负，要报仇，同帮派欺负别人，叫好。用观察者模式模拟场景。

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<string>
#include<list>
using namespace std;

class Infomation;

//抽象观察者
class AbstractGang{
public:
    virtual void Update(Infomation*) = 0;
    virtual string GetGangName() = 0;
};

//江湖消息
class Infomation{
public:
    Infomation(AbstractGang* beat, AbstractGang* beaten){
        this->Beat = beat;
        this->Beaten = beaten;
    }
    AbstractGang* Beat; //打人的帮派
    AbstractGang* Beaten; //被打的帮派
};

//华山派坐等百晓生的广播
class HuashanGang : public AbstractGang{
public:
    HuashanGang(){
        m_GangName = "华山派";
```

```
    }
    virtual void Update(Infomation* info){
        if (info->Beat == this && info->Beaten != this){
            cout << "打死" << info->Beaten->GetGangName() << "," << this->GetGangName()
<< "最厉害!" << endl;
        }
        else if (info->Beat != this && info->Beaten != this){
            cout << this->GetGangName()  <<"坐看" << info->Beat->GetGangName() <<"和" <<
info->Beaten->GetGangName() << "干架!" << endl;
        }
        else if(info->Beat != this && info->Beaten == this){
            cout << info->Beat->GetGangName() << "干我们" << this->GetGangName() << ",
我们要报仇!" << endl;
        }
    }
    virtual string GetGangName(){
        return m_GangName;
    }
private:
    string m_GangName;
};

//昆仑派坐等百晓生的广播
class KunlunGang : public AbstractGang{
public:
    KunlunGang(){
        m_GangName = "昆仑派";
    }
    virtual void Update(Infomation* info){
        if (info->Beat == this && info->Beaten != this){
            cout << "打死" << info->Beaten->GetGangName() << "," << this->GetGangName()
<< "最厉害!" << endl;
        }
        else if (info->Beat != this && info->Beaten != this){
            cout << this->GetGangName() <<"坐看" << info->Beat->GetGangName() <<"和" <<
info->Beaten->GetGangName() << "干架!" << endl;
        }
        else if (info->Beat != this && info->Beaten == this){
            cout << info->Beat->GetGangName() << "干我们" << this->GetGangName() << ",
我们要报仇!" << endl;
        }
    }
    virtual string GetGangName(){
        return m_GangName;
```

```
    }
private:
    string m_GangName;
};

//武当派坐等百晓生的广播
class WudangGang : public AbstractGang{
public:
    WudangGang(){
        m_GangName = "武当派";
    }
    virtual void Update(Infomation* info){
        if (info->Beat == this && info->Beaten != this){
            cout << "打死" << info->Beaten->GetGangName() << "," << this->GetGangName()
<< "最厉害!" << endl;
        }
        else if (info->Beat != this && info->Beaten != this){
            cout << this->GetGangName() <<"坐看" << info->Beat->GetGangName() <<"和" <<
info->Beaten->GetGangName() << "干架!" << endl;
        }
        else if (info->Beat != this && info->Beaten == this){
            cout << info->Beat->GetGangName() << "干我们" << this->GetGangName() << ",
我们要报仇!" << endl;
        }
    }
    virtual string GetGangName(){
        return m_GangName;
    }
private:
    string m_GangName;
};

//百晓生 - 大嘴巴子
class Baixiaosheng{
public:
    void addGang(AbstractGang* gang){
        m_list.push_back(gang);
    }
    void setInfomation(Infomation* info){
        pInfo = info;
    }
    void Notify(){
        for (list<AbstractGang*>::iterator it = m_list.begin(); it != m_list.end(); it ++){
            (*it)->Update(pInfo);
```

```
        }
    }
private:
    list<AbstractGang*> m_list;
    Infomation* pInfo;
};

void test01(){

    AbstractGang* wudang = new WudangGang; //武当派
    AbstractGang* kunlun = new KunlunGang; //昆仑派
    AbstractGang* huashan = new HuashanGang; //华山派

    Baixiaosheng* baixiaosheng = new Baixiaosheng; //百晓生 大嘴巴子
    baixiaosheng->addGang(wudang);
    baixiaosheng->addGang(kunlun);
    baixiaosheng->addGang(huashan);

    Infomation* pInfo = new Infomation(wudang, huashan); //创建江湖消息  武当打华山
    baixiaosheng->setInfomation(pInfo); //告诉百晓生武当打华山了

    //大嘴巴开始向各大门派发消息了
    baixiaosheng->Notify();

}

int main(){

    test01();

    system("pause");
    return EXIT_SUCCESS;
}
```