#include *<iostream>*

**class** **Strategy**

{

**public**:

**virtual** ~Strategy() {}

**virtual** void use() = 0;

};

**class** **Strategy\_1**: **public** Strategy

{

**public**:

void use(){

std::cout << "Strategy\_1" << std::endl;

}

};

**class** **Strategy\_2**: **public** Strategy

{

**public**:

void use(){

std::cout << "Strategy\_2" << std::endl;

}

};

**class** **Strategy\_3**: **public** Strategy

{

**public**:

void use(){

std::cout << "Strategy\_3" << std::endl;

}

};

**class** **Context**

{

**protected**:

Strategy\* operation;

**public**:

**virtual** ~Context() {}

**virtual** void useStrategy() = 0;

**virtual** void setStrategy(Strategy\* v) = 0;

};

**class** **Client**: **public** Context

{

**public**:

void useStrategy()

{

operation->use();

}

void setStrategy(Strategy\* o)

{

operation = o;

}

};

int main(int */\*argc\*/*, char\* */\*argv\*/*[])

{

Client customClient;

Strategy\_1 str1;

Strategy\_2 str2;

Strategy\_3 str3;

customClient.setStrategy(&str1);

customClient.useStrategy();

customClient.setStrategy(&str2);

customClient.useStrategy();

customClient.setStrategy(&str3);

customClient.useStrategy();

**return** 0;

}

«Фабричный метод»

#include *<iostream>*

#include *<string>*

**using** **namespace** std;

**class** **Product**{

**public**:

**virtual** string getName() = 0;

**virtual** ~Product() {}

};

**class** **ConcreteProductA**: **public** Product{

**public**:

string getName() {**return** "ConcreteProductA";}

};

**class** **ConcreteProductB**: **public** Product{

**public**:

string getName() {**return** "ConcreteProductB";}

};

**class** **Creator**{

**public**:

**virtual** Product\* factoryMethod() = 0;

};

**class** **ConcreteCreatorA**: **public** Creator{

**public**:

Product\* factoryMethod() {**return** **new** ConcreteProductA();}

};

**class** **ConcreteCreatorB**: **public** Creator{

**public**:

Product\* factoryMethod() {**return** **new** ConcreteProductB();}

};

int main()

{

**static** **const** size\_t count = 2;

ConcreteCreatorA CreatorA;

ConcreteCreatorB CreatorB;

*// An array of creators*

Creator\*creators[count] = {&CreatorA, &CreatorB};

*// Iterate over creators and create products*

**for**(size\_t i = 0; i<count; i++){

Product\* product=creators[i]->factoryMethod();

cout << product->getName() << endl;

**delete** product;

}

**return** 0;

}