|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
|  |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
|  |

**Практическая работа № 5,**

по дисциплине: **«Технологии и методы программирования»**.

Выполнил студент 3 курса группы БИСО-01-21 Нагибин А. А.

Проверил преподаватель Лесько С.А.

Москва 2024

# Абстрактная фабрика

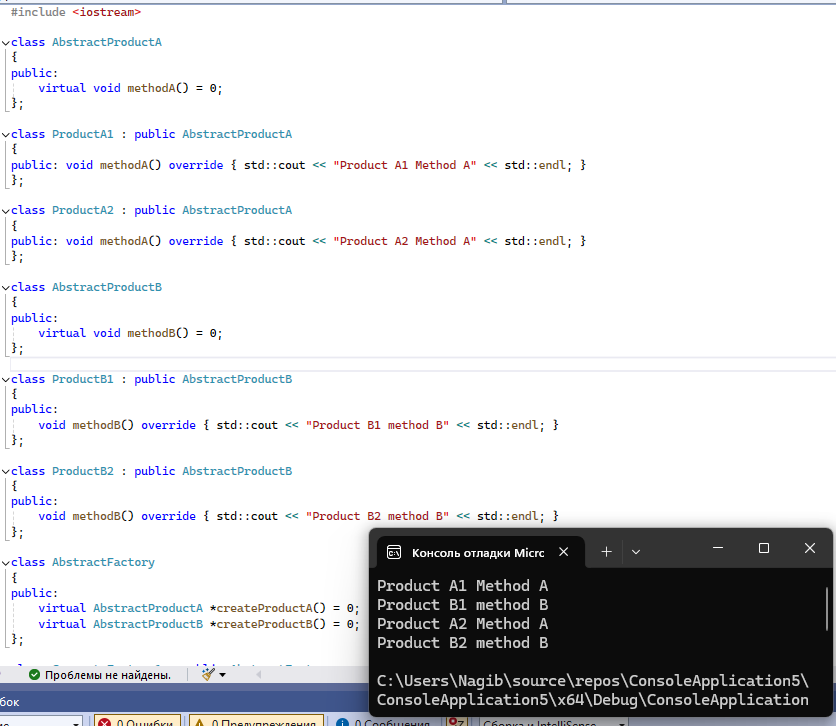


Рисунок 1 – листинг программы и результат работы

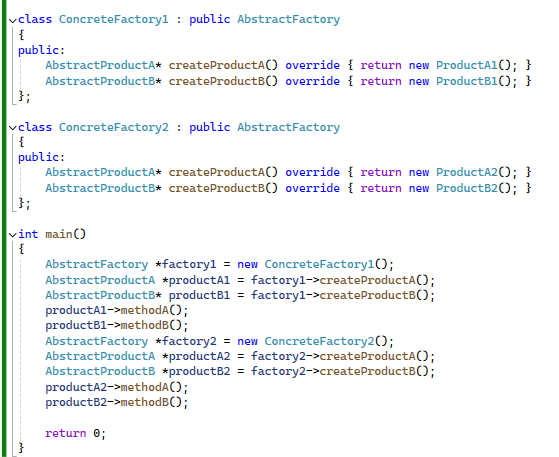


Рисунок 2 – листинг программы

Листинг программы:

#include <iostream>

class AbstractProductA

{

public:

virtual void methodA() = 0;

};

class ProductA1 : public AbstractProductA

{

public: void methodA() override { std::cout << "Product A1 Method A" << std::endl; }

};

class ProductA2 : public AbstractProductA

{

public: void methodA() override { std::cout << "Product A2 Method A" << std::endl; }

};

class AbstractProductB

{

public:

virtual void methodB() = 0;

};

class ProductB1 : public AbstractProductB

{

public:

void methodB() override { std::cout << "Product B1 method B" << std::endl; }

};

class ProductB2 : public AbstractProductB

{

public:

void methodB() override { std::cout << "Product B2 method B" << std::endl; }

};

class AbstractFactory

{

public:

virtual AbstractProductA \*createProductA() = 0;

virtual AbstractProductB \*createProductB() = 0;

};

class ConcreteFactory1 : public AbstractFactory

{

public:

AbstractProductA\* createProductA() override { return new ProductA1(); }

AbstractProductB\* createProductB() override { return new ProductB1(); }

};

class ConcreteFactory2 : public AbstractFactory

{

public:

AbstractProductA\* createProductA() override { return new ProductA2(); }

AbstractProductB\* createProductB() override { return new ProductB2(); }

};

int main()

{

AbstractFactory \*factory1 = new ConcreteFactory1();

AbstractProductA \*productA1 = factory1->createProductA();

AbstractProductB\* productB1 = factory1->createProductB();

productA1->methodA();

productB1->methodB();

AbstractFactory \*factory2 = new ConcreteFactory2();

AbstractProductA \*productA2 = factory2->createProductA();

AbstractProductB \*productB2 = factory2->createProductB();

productA2->methodA();

productB2->methodB();

return 0;

}

# Строитель

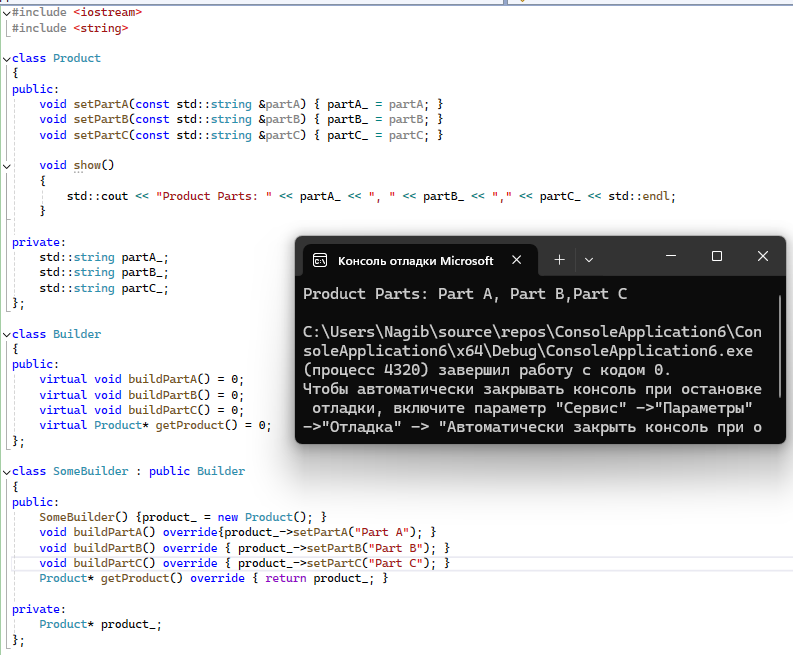


Рисунок 3 – листинг программы и результат работы

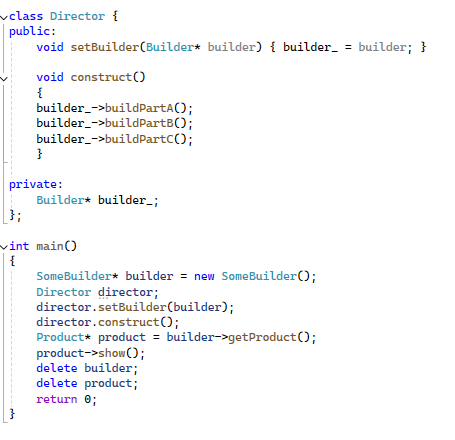


Рисунок 4 – листинг программы

Листинг программы:

#include <iostream>

#include <string>

class Product

{

public:

void setPartA(const std::string &partA) { partA\_ = partA; }

void setPartB(const std::string &partB) { partB\_ = partB; }

void setPartC(const std::string &partC) { partC\_ = partC; }

void show()

{

std::cout << "Product Parts: " << partA\_ << ", " << partB\_ << "," << partC\_ << std::endl;

}

private:

std::string partA\_;

std::string partB\_;

std::string partC\_;

};

class Builder

{

public:

virtual void buildPartA() = 0;

virtual void buildPartB() = 0;

virtual void buildPartC() = 0;

virtual Product\* getProduct() = 0;

};

class SomeBuilder : public Builder

{

public:

SomeBuilder() {product\_ = new Product(); }

void buildPartA() override{product\_->setPartA("Часть A"); }

void buildPartB() override { product\_->setPartB("Часть B"); }

void buildPartC() override { product\_->setPartC("Часть C"); }

Product\* getProduct() override { return product\_; }

private:

Product\* product\_;

};

class Director {

public:

void setBuilder(Builder\* builder) { builder\_ = builder; }

void construct()

{

builder\_->buildPartA();

builder\_->buildPartB();

builder\_->buildPartC();

}

private:

Builder\* builder\_;

};

int main()

{

SomeBuilder\* builder = new SomeBuilder();

Director director;

director.setBuilder(builder);

director.construct();

Product\* product = builder->getProduct();

product->show();

delete builder;

delete product;

return 0;

}

# Адаптер

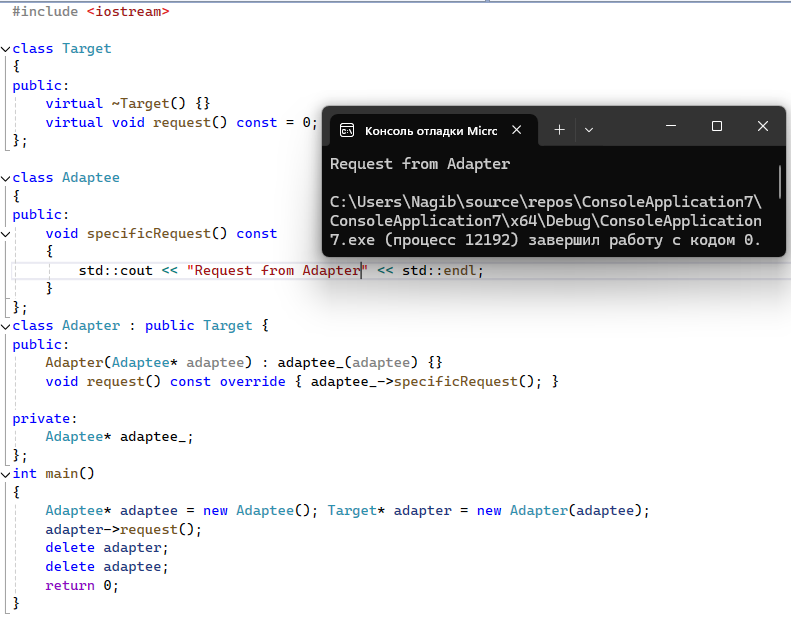


Рисунок 3 – листинг программы и результат работы

Листинг программы:

#include <iostream>

class Target

{

public:

virtual ~Target() {}

virtual void request() const = 0;

};

class Adaptee

{

public:

void specificRequest() const

{

std::cout << "Request from Adapter" << std::endl;

}

};

class Adapter : public Target {

public:

Adapter(Adaptee\* adaptee) : adaptee\_(adaptee) {}

void request() const override { adaptee\_->specificRequest(); }

private:

Adaptee\* adaptee\_;

};

int main()

{

Adaptee\* adaptee = new Adaptee(); Target\* adapter = new Adapter(adaptee);

adapter->request();

delete adapter;

delete adaptee;

return 0;

}

# Посредник



Рисунок 4 – листинг программы и результат работы



Рисунок 5 – листинг программы

Листинг программы:

#include <iostream>

#include <string>

#include <vector>

class Mediator;

class Colleague

{

public:

virtual ~Colleague() {}

virtual void receiveMessage(const std::string &message) = 0;

virtual void sendMessage(const std::string &message) const = 0;

virtual void setMediator(Mediator\* mediator) { mediator\_ = mediator; }

protected:

Mediator\* mediator\_;

};

class Mediator

{

public:

virtual ~Mediator() {}

virtual void sendMessage(const std::string &message, Colleague\* colleague) const = 0;

};

class SomeColleague : public Colleague

{

public:

SomeColleague(const std::string& name) : name\_(name) {}

void receiveMessage(const std::string &message) override { std::cout << name\_ << " received: " << message << std::endl; }

void sendMessage(const std::string &message) const override { mediator\_->sendMessage(message, const\_cast<SomeColleague \*>(this)); }

private:

std::string name\_;

};

class SomeMediator : public Mediator

{

public:

void addColleague(Colleague\* colleague)

{

colleagues\_.push\_back(colleague);

colleague->setMediator(this);

}

void sendMessage(const std::string &message, Colleague\* colleague)

const override

{

for (auto col : colleagues\_)

{

if (col != colleague)

{

col->receiveMessage(message);

}

}

}

private:

std::vector<Colleague \*> colleagues\_;

};

int main()

{

SomeMediator mediator;

SomeColleague colleaguel("Kolega 1");

SomeColleague colleague2("Kolega 2");

mediator.addColleague(&colleaguel);

mediator.addColleague(&colleague2);

colleaguel.sendMessage("Greetings from the College 1");

colleague2.sendMessage("Greetings from the College 2");

return 0;

}