**Урок 3.8.11**

Задание №6:

Дополнить предыдущее задание. Создать вектор указателей на  
«Товар», поместить туда экземпляр «Холодильника» и экземпляр «Чайника».  
Сделать так, чтобы при обходе вектора и вызове метода info() вызывались методы  
дочерних классов. Добавить к дочерним классам динамическое поле (указатель на  
int), хранящее информацию об объёме, который товар занимает на складе. В  
конструкторе выделить память под это поле и задать стартовое значение (очевидно,  
что холодильник должен занимать больше места, чем чайник). Корректно  
реализовать деструкторы для всей иерархии классов. Протестировать корректную  
работу классов в основной программе. Протестируйте работу методов. Отправьте  
преподавателю код программы и скриншот консоли.

// https://github.com/A-l-E-v/CPP\_Synergy/blob/main/U-3811/product\_vector.cpp

#include <iostream>

#include <vector>

using namespace std;

// Родительский класс Product

// template <typename T>

class Product

{

private:

int quantity;

string name;

double price;

// T price;

public:

Product(const string &name, double price = 0, int quantity = 0) : name(name), price(price), quantity(quantity)

// Product(const string &name, T price = 0, int quantity = 0) : name(name), price(price), quantity(quantity)

{

cout << "Конструктор Product()\n";

};

virtual ~Product()

{

cout << "Деструктор Product()\n";

}

// вывод общей информации о любом товаре

virtual void info() const

{

cout << "Название: " << name << endl;

printf("Стоимость единицы товара: %.2f руб.\n", price);

cout << "Количество товара на складе: " << quantity << endl;

}

// Перегрузка оператора сложения

Product &operator+=(int quantity)

{

this->quantity += quantity;

return \*this;

}

};

// Дочерний класс наследуется от Product

// template <typename T>

// class Fridge : public Product<T>

class Fridge : public Product

{

private:

string type;

int \*volume;

public:

Fridge(const string &name, double price, int quantity, const string &type) : Product(name, price, quantity)

// Fridge(const string &name, T price, int quantity, const string &type) : Product<T>(name, price, quantity)

{

this->type = type;

volume = new int(800);

cout << "Конструктор Fridge()\n";

}

virtual ~Fridge()

{

cout << "Деструктор Fridge()\n";

delete volume;

}

// Переопределение метода вывода дополнительной информации

void info() const override

{

Product::info();

// Product<T>::info();

cout << "Тип холодильника: " << type << "\n";

cout << "Объём упаковки холодильника на складе: " << \*volume << " куб.см" << "\n\n";

}

};

// Дочерний класс Kettle

// template <typename T>

// class Kettle : public Product<T>

class Kettle : public Product

{

private:

string type;

int \*volume;

public:

Kettle(const string &name, double price, int quantity, const string &type) : Product(name, price, quantity)

// Kettle(const string &name, T price, int quantity, const string &type) : Product<T>(name, price, quantity)

{

this->type = type;

volume = new int(100);

cout << "Конструктор Kettle()\n";

}

virtual ~Kettle()

{

cout << "Деструктор Kettle()\n";

delete volume;

}

// Переопределение метода вывода дополнительной информации

void info() const override

{

Product::info();

// Product<T>::info();

cout << "Тип чайника: " << type << "\n";

cout << "Объём упаковки чайника на складе: " << \*volume << " куб.см" << "\n\n";

}

};

int main()

{

cout << "=== Подготовительный этап ===\n";

vector<Product \*> Stock;

Product \*fridge = new Fridge("Холодильник Beko", 39500.94, 5, "двухкамерный");

Product \*kettle = new Kettle("Электрочайник Braun", 2500.58, 10, "с функцией термоса");

Stock.push\_back(fridge);

Stock.push\_back(kettle);

cout << "=== Подготовительный этап завершён ===\n\n";

// поэлементный обход вектора и вывод информации о продукте

for (const auto &product : Stock)

{

product->info();

}

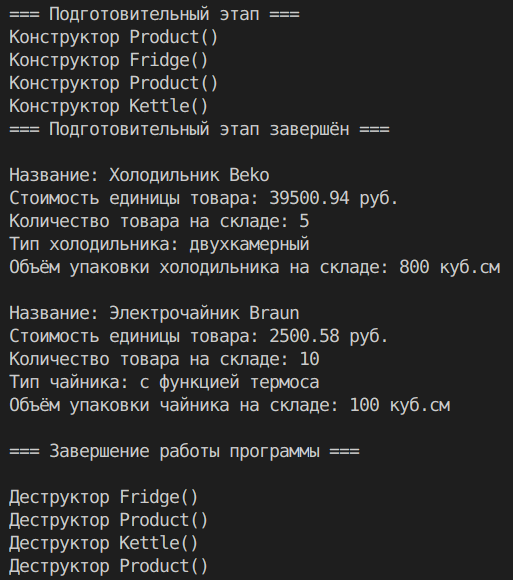
cout << "=== Завершение работы программы ===\n\n";

for (auto product : Stock)

{

delete product;

}



return 0;

}