**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Дисциплина: «Технология программирования»

Лабораторная работа №8.

Тема: «Конструирование классов на основе принципа наследования»

Выполнили: Студенты группы РЦТ-22

Балан К. А.

Орехва В. Э.

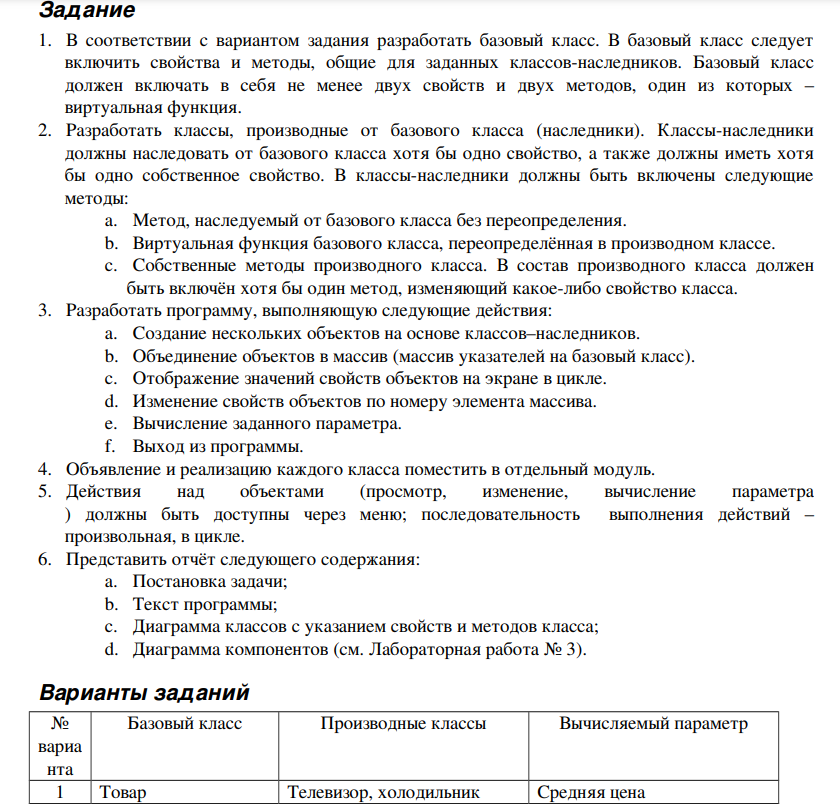
Принял: к.т.н, доцент кафедры ПИиВТ

Дагаев А. В.

Санкт-Петербург

2023г

**Постановка задачи**

****

**Описание класса**

Класс Product:

class Product {

private:

std::string name;

double price;

public:

Product(const std::string& name, double price) : name(name), price(price) {}

virtual ~Product() {}

const std::string& getName() const {}

void setName(const std::string& newName) {}

double getPrice() const {}

void setPrice(double newPrice) {}

void display() {}

void changePrice(double newPrice) {}

virtual void displayAdditionalInfo() {}

virtual void editAdditionalInfo() {}

virtual double calculateAveragePrice() {}

};

Данный класс предназначен для описания товара.

Поля класса:

name (строка): имя продукта.

price (число): цена продукта.

Методы:

display(): Выводит информацию о продукте, включая имя и цену.

changePrice(newPrice): Изменяет цену продукта на новую.

displayAdditionalInfo(): Выводит дополнительную информацию о продукте. Виртуальный метод, который может быть переопределен в подклассах.

editAdditionalInfo(): Редактирует дополнительную информацию о продукте. Виртуальный метод, который может быть переопределен в подклассах.

calculateAveragePrice(): Вычисляет среднюю цену продукта. Виртуальный метод, который может быть переопределен в подклассах.

Класс TV (наследуется от Product):

class TV : public Product {

private:

std::string brand;

std::string resolution;

public:

TV(const std::string& name, double price, const std::string& brand, const std::string& resolution) : Product(name, price), brand(brand), resolution(resolution) {}

~TV() {}

const std::string& getBrand() const {}

void setBrand(const std::string& newBrand) {}

const std::string& getResolution() const {}

void setResolution(const std::string& newResolution) {}

void display() {}

void displayAdditionalInfo() override {}

void editAdditionalInfo() override {}

double calculateAveragePrice() override {}

};

Класс TV расширяет класс “Product” и представляет телевизоры

Новые свойства дочернего класса:

resolution (строка): разрешение телевизора.

Описание методов класса TV:

getBrand(): получает данные о бренде.

setBrand(): устанавливает новые данные о бренде.

getResolution(): получает данные о разрешении телевизора

setResolution(): устанавливает новые данные о разрешении телевизора

display(): показывает все данные о телевизорах.

displayAdditionalInfo(): Переопределяет метод displayAdditionalInfo() из класса Product, выводящий информацию о марке и разрешении телевизора.

editAdditionalInfo(): Переопределяет метод editAdditionalInfo() из класса Product, позволяющий редактировать марку и разрешение телевизора.

calculateAveragePrice():вычисляет среднюю цену холодильника на основе его текущей цены.

Класс Refrigerator (наследуется от Product):

class Refrigerator : public Product {

private:

int capacity;

std::string brand;

int temperature;

public:

Refrigerator(const std::string& name, double price, int temperature const std::string& brand, int capacity) : Product(name, price), brand(brand), capacity(capacity) {}

~Refrigerator() {}

const std::string& getBrand() const {}

void setBrand(const std::string& newBrand) {}

const std::string& gettemperature() const {}

void settemperature(const std::string& newtemperature) {}

int getCapacity() const {}

void setCapacity(int newCapacity) {}

void display() {}

void displayAdditionalInfo() override {}

void editAdditionalInfo() override {}

double calculateAveragePrice() override {}

};

Класс Refrigerator расширяет класс Product и представляет холодильник.

Новые свойства дочернего класса:

capasity (строка): вместимость холодильника.

Temperature (число): температура холодильника.

Описание методов класса Refrigerator:

getBrand(): получает данные о бренде.

setBrand(): устанавливает новые данные о бренде.

gettemperature(): получает данные о температуре.

settemperature():устанавливает новые данные о температуре.

getCapacity(): получает данные о вместимости холодильника.

setCapacity(): устанавливает новые данные о вместимости холодильника.

display(): показывает все данные о холодильниках.

displayAdditionalInfo(): отображает дополнительную информацию о холодильнике.

editAdditionalInfo(newInfo): позволяет редактировать дополнительную информацию о холодильнике.

calculateAveragePrice():вычисляет среднюю цену холодильника на основе его текущей цены.

**Описание программы**

В функции "main" создаются три объекта класса "Product" (два объекта типа "TV" и один объект типа "Refrigerator") и размещаются в массиве "products". Затем выполняется бесконечный цикл, который отображает меню и позволяет пользователю взаимодействовать с продуктами. В зависимости от выбранного пункта меню, выполняются соответствующие операции, такие как отображение информации о продуктах, изменение свойств продукта, вычисление средней цены и выход из программы.

**Результат работы программы**

На Рис.1 представлен пример работы программы. В нем мы видим меню выбора, список товаров и их характеристики. При выходе из программы все товары удаляются.

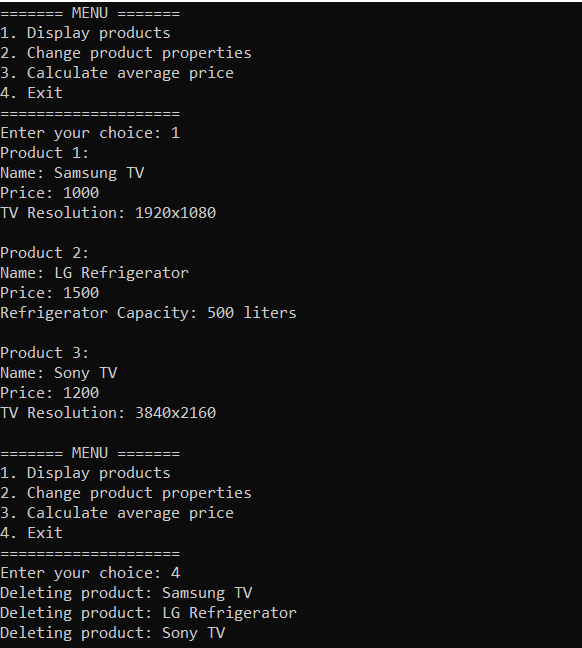
****

Рис.1 Пример работы программы

**Вывод**

В ходе проделанной работы были получены следующие результаты:

1). Разработана программа, соответствующая условию задачи

2). Получены практические навыки работы с классами

**Листинг**

#include <iostream>

#include <string>

class Product {

private:

std::string name;

double price;

public:

Product(const std::string& name, double price) : name(name), price(price) {}

virtual ~Product() {

std::cout << "Deleting product: " << name << std::endl;

}

const std::string& getName() const {

return name;

}

void setName(const std::string& newName) {

name = newName;

}

double getPrice() const {

return price;

}

void setPrice(double newPrice) {

price = newPrice;

}

void display() {

std::cout << "Name: " << getName() << std::endl;

std::cout << "Price: " << getPrice() << std::endl;

displayAdditionalInfo();

}

void changePrice(double newPrice) {

setPrice(newPrice);

}

virtual void displayAdditionalInfo() {}

virtual void editAdditionalInfo() {}

virtual double calculateAveragePrice() {

return getPrice();

}

};

class TV : public Product {

private:

std::string brand;

std::string resolution;

public:

TV(const std::string& name, double price, const std::string& brand, const std::string& resolution) : Product(name, price), brand(brand), resolution(resolution) {}

const std::string& getBrand() const {

return brand;

}

void setBrand(const std::string& newBrand) {

brand = newBrand;

}

const std::string& getResolution() const {

return resolution;

}

void setResolution(const std::string& newResolution) {

resolution = newResolution;

}

void display() {

Product::display();

std::cout << "Brand: " << getBrand() << std::endl;

std::cout << "Resolution: " << getResolution() << std::endl;

}

void displayAdditionalInfo() override {

std::cout << "TV Resolution: " << getResolution() << std::endl;

}

void editAdditionalInfo() override {

std::cout << "Enter new resolution: ";

std::cin >> resolution;

}

double calculateAveragePrice() override {

return getPrice();

}

};

class Refrigerator : public Product {

private:

int capacity;

public:

Refrigerator(const std::string& name, double price, int capacity) : Product(name, price), capacity(capacity) {}

int getCapacity() const {

return capacity;

}

void setCapacity(int newCapacity) {

capacity = newCapacity;

}

void display() {

Product::display();

std::cout << "Capacity: " << getCapacity() << " liters" << std::endl;

}

void displayAdditionalInfo() override {

std::cout << "Refrigerator Capacity: " << getCapacity() << " liters" << std::endl;

}

void editAdditionalInfo() override {

int newCapacity;

std::cout << "Enter new capacity: ";

std::cin >> newCapacity;

changeCapacity(newCapacity);

}

double calculateAveragePrice() override {

return getPrice();

}

};

int main() {

const int numProducts = 3;

Product\* products[numProducts];

products[0] = new TV("Samsung TV", 1000.0, "Samsung", "1920x1080");

products[1] = new Refrigerator("LG Refrigerator", 1500.0, 500);

products[2] = new TV("Sony TV", 1200.0, "Sony", "3840x2160");

int choice;

while (true) {

std::cout << "======= MENU =======" << std::endl;

std::cout << "1. Display products" << std::endl;

std::cout << "2. Change product properties" << std::endl;

std::cout << "3. Calculate average price" << std::endl;

std::cout << "4. Exit" << std::endl;

std::cout << "====================" << std::endl;

std::cout << "Enter your choice: ";

std::cin >> choice;

if (choice == 1) {

for (int i = 0; i < numProducts; ++i) {

std::cout << "Product " << i + 1 << ":" << std::endl;

products[i]->display();

std::cout << std::endl;

}

}

else if (choice == 2) {

int productIndex;

std::cout << "Enter the product number (1-" << numProducts << "): ";

std::cin >> productIndex;

productIndex--; // Convert to zero-based index

if (productIndex >= 0 && productIndex < numProducts) {

int propertyChoice;

std::cout << "1. Change price" << std::endl;

std::cout << "2. Edit additional info" << std::endl;

std::cout << "Enter your choice: ";

std::cin >> propertyChoice;

if (propertyChoice == 1) {

double newPrice;

std::cout << "Enter new price: ";

std::cin >> newPrice;

products[productIndex]->setPrice(newPrice);

}

else if (propertyChoice == 2) {

products[productIndex]->editAdditionalInfo();

}

else {

std::cout << "Invalid choice or product type." << std::endl;

}

}

else {

std::cout << "Invalid product number." << std::endl;

}

}

else if (choice == 3) {

double totalPrice = 0.0;

for (int i = 0; i < numProducts; ++i) {

totalPrice += products[i]->calculateAveragePrice();

}

double averagePrice = totalPrice / numProducts;

std::cout << "Average price: " << averagePrice << std::endl;

}

else if (choice == 4) {

break;

}

else {

std::cout << "Invalid choice." << std::endl;

}

}

// Clean up memory

for (int i = 0; i < numProducts; ++i) {

delete products[i];

}

return 0;

}