МИНИСТЕРСТВО НАУКИ и высшего образования

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**Отчёт по практической работе № 1**

**по учебной дисциплине** «Системное программирование»

**Тема:** «Проектирование классов»

Выполнил студент

специальности 09.02.07

Информационные системы и программирование

IV курса группы 42919/6

Песигин Антон

Станиславович

Преподаватель

Молькова Лолита Юрьевна

Санкт-Петербург,

2024

**Цели работы:**

1. Ознакомление с процессом классификации на примере моделирования классов предметной области;
2. Ознакомление с диаграммами классов и взаимодействия UML.

**Индивидуальное задание 1:**

* Реализовать массив данных в соответствии с вариантом задания;
* Определить метод установки свойств (при недопустимых аргументах функции возвращать «false» и выдавать текст ошибки на экран);
* Написать демонстрационную программу, в которой показать работу с переменными.

Вариант 14.

Пользователь вводит массив Квадратов

Свойства: Сторона

Операции:

* увеличение/уменьшение размера на определенный процент;
* вычисление периметра и площади;
* вычисление диагонали.

Пользователь выбирает элемент массива и выполняемую операцию.

**Ход работы**

1. Создать класс Square (рисунок 1);

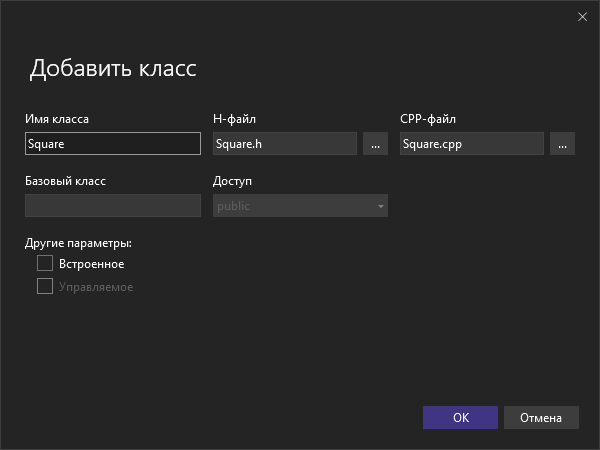


Рисунок 1 – Создание класса

1. Написать свойства, методы класса и использовать разбиение по модулям;

Square.h:

#pragma once

#include <iostream>

class Square

{

double side; //Свойство

public:

//Конструктор класса

Square(double side);

//Увеличение размера стороны на процент

double sizeIncrease(double percent);

//Уменьшение размера стороны на процент

double sizeReduction(double percent);

//Периметр квадрата

double perimeter();

//Площадь квадрата

double square();

//Диагональ квадрата

double diagonal();

};

Square.cpp:

#include "Square.h"

Square::Square(double side):side(side) {}

double Square::sizeIncrease(double percent)

{

double per = (side \* percent) / 100;

double newSide = side + per;

return newSide;

}

double Square::sizeReduction(double percent)

{

double per = (side \* percent) / 100;

double newSide = side - per;

return newSide;

}

double Square::perimeter()

{

double perimeter = 4 \* side;

return perimeter;

}

double Square::square()

{

double square = side \* side;

return square;

}

double Square::diagonal()

{

double d = side \* sqrt(2);

return d;

}

1. В методе main создать динамический массив квадратов и написать код;

#include <iostream>

#include <vector>

#include <cmath>

#include "Square.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

vector<Square> squares; //Динамический массив квадратов

double side, percent, count; //Сторона квадрата, процент, количество квадратов

double index = 0; //Индекс элемента в массиве

cout << "Введите количество квадратов (> 0): ";

//Вывод ошибки

while (!(cin >> count) || count <= 0)

{

cout << "Ошибка" << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); //Отбравсывает неправильно введенное значение

}

//Цикл для заполнения массива

for (int i = 0; i < count; ++i)

{

cout << "Введите длину стороны " << i + 1 << " квадрата" << ": ";

//Вывод ошибки

while (!(cin >> side) || side <=0)

{

cout << "Ошибка" << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

squares.emplace\_back(Square(side));

}

while (true)

{

cout << "\n\nВыберете квадрат в массиве (1 - " << count << "): ";

//Вывод ошибки

while (!(cin >> index) || index <= 0 || index > count)

{

cout << "Ошибка" << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

//Выбор операции пользователем

int operation;

cout << "\nВыберете операцию:\n1 - увеличение размера на процент\n2 - уменьшение размера на процент\n3 - вычисление периметра\n4 - вычисление площади\n5 - вычисление диагонали\n";

cin >> operation;

index--;

//Выбор операции с помощью оператора switch

switch (operation)

{

if (index >= 0 && index < count)

{

case 1:

cout << "Укажите процент, на который нужно увеличить сторону: ";

cin >> percent;

cout << "Результат: " << round(squares[index].sizeIncrease(percent) \* 100) / 100;

break;

case 2:

cout << "Укажите процент, на который нужно уменьшить сторону: ";

cin >> percent;

cout << "Результат: " << round(squares[index].sizeReduction(percent) \* 100) / 100;

break;

case 3:

cout << "Периметр = " << squares[index].perimeter();

break;

case 4:

cout << "Площадь = " << squares[index].square();

break;

case 5:

cout << "Диагональ = " << round(squares[index].diagonal() \* 100) / 100;

break;

default:

cout << "Такой операции нет!"; //Вывод ошибки

break;

}

}

}

}

1. Разработать UML – диаграмму (рисунок 2);

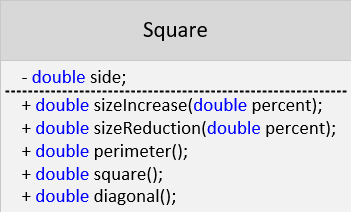


Рисунок 2 – UML – диаграмма

1. Результат (рисунок 3, 4, 5, 6, 7, 8, 9, 10, 11).

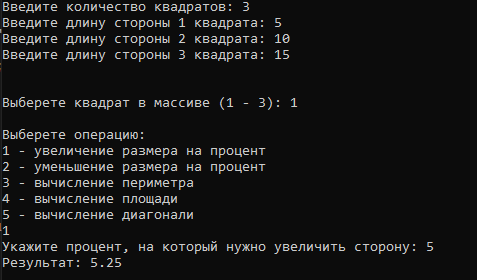


Рисунок 3 – Увеличение размера

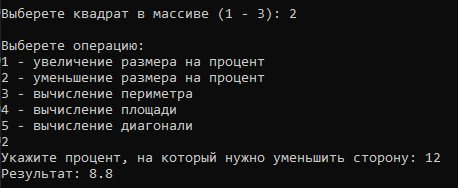


Рисунок 4 – Уменьшение размера

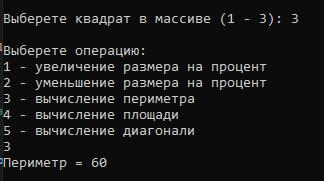


Рисунок 5 – Расчет периметра

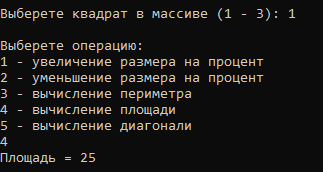


Рисунок 6 – Расчет площади

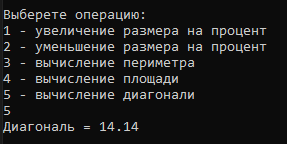


Рисунок 7 – Вычисление диагонали

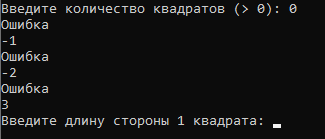


Рисунок 8 – Вывод ошибки

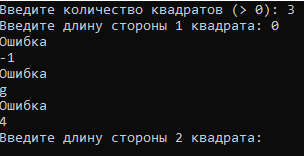


Рисунок 9 – Вывод ошибки

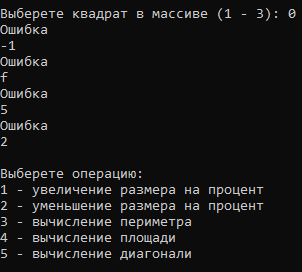


Рисунок 10 – Вывод ошибки

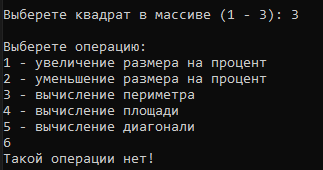


Рисунок 11 – Вывод ошибки

**Индивидуальное задание 2:**

• Перечислите несколько однотипных объектов и конкретные значения их свойств.

• Выделите классы из группы однотипных объектов.

Требования:

5 классов, 7 объектов каждого класса, 3 свойства для каждого класса, 3 метода для каждого класса, имена классов - существительные

Предметная область: Сельское хозяйство

**Ход работы**

1. Создать 5 классов (рисунок 12);

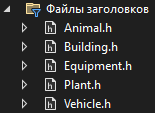


Рисунок 12 – 5 классов

1. Добавить свойства, методы для каждого класса и использовать разбиение по модулям;

Animal.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Animal

{

//Свойства класса

private:

string name;

int age;

int weight;

//Конструктор класса

public:

Animal(string name, int age, int weight);

//Методы класса

void Eat() const;

void Sleep() const;

void Out() const;

};

Animal.cpp:

#include "Animal.h"

Animal::Animal(string name, int age, int weight) : name(name), age(age), weight(weight) {}

void Animal::Eat() const

{

cout << "\n" << name << " ест";

}

void Animal::Sleep() const

{

cout << "\n" << name << " спит";

}

void Animal::Out() const

{

cout << "\n\nЖивотное: " << name;

cout << "\nВозраст: " << age;

cout << "\nВес: " << weight;

}

Класс Building.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Building

{

//Свойства класса

private:

string name;

int square;

string purpose;

//Конструктор класса

public:

Building(string name, int square, string purpose);

//Методы класса

void Construct();

void NotConstruct();

void Out() const;

};

Building.cpp:

#include "Building.h"

Building::Building(string name, int square, string purpose) : name(name), square(square), purpose(purpose) {}

void Building::Construct()

{

cout << name << ": построено";

}

void Building::NotConstruct()

{

cout << name << ": в процессе строительства";

}

void Building::Out() const

{

cout << "\n\nНазвание: " << name;

cout << "\nПлощадь: " << square;

cout << "\nПредназначение: " << purpose;

}

Класс Equipment.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Equipment

{

//Свойства класса

protected:

string name;

int price;

string purpose;

//Конструктор класса

public:

Equipment(string name, int price, string purpose);

//Методы класса

void Repair();

void Plough();

void Out() const;

};

Equipment.cpp:

#include "Equipment.h"

Equipment::Equipment(string name, int price, string purpose) : name(name), price(price), purpose(purpose) {}

void Equipment::Repair()

{

cout << name << " для ремонта";

}

void Equipment::Plough()

{

cout << name << " для вспашки и посева";

}

void Equipment::Out() const

{

cout << "\n\nНазвание: " << name;

cout << "\nЦена: " << price;

cout << "\nПредназначение: " << purpose;

}

Класс Plant.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Plant

{

//Свойства класса

protected:

string name;

string type;

int age;

//Конструктор класса

public:

Plant(string name, string type, int age);

//Методы класса

void Fruit();

void Vegetable();

void Out() const;

};

Plant.cpp:

#include "Plant.h"

Plant::Plant(string name, string type, int age) : name(name), type(type), age(age) {}

void Plant::Fruit()

{

cout << "\n" << name << " - фрукт";

}

void Plant::Vegetable()

{

cout << "\n" << name << " - овощ";

}

void Plant::Out() const

{

cout << "\n\nРастение: " << name;

cout << "\nТип питания: " << type;

cout << "\nСрок созревания (в днях): " << age;

}

Класс Vehicle.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Vehicle

{

//Свойства класса

protected:

string name;

int price;

string purpose;

//Конструктор класса

public:

Vehicle(string name, int price, string purpose);

//Методы класса

void Available();

void Unavailable();

void Out() const;

};

Vehicle.cpp:

#include "Vehicle.h"

Vehicle::Vehicle(string name, int price, string purpose) : name(name), price(price), purpose(purpose) {}

void Vehicle::Available()

{

cout << name << ": есть в наличии";

}

void Vehicle::Unavailable()

{

cout << name << ": нет в наличии";

}

void Vehicle::Out() const

{

cout << "\n\nНазвание: " << name;

cout << "\nЦена: " << price;

cout << "\nПредназначение: " << purpose;

}

1. В методе main создать по 7 объектов каждого класса;

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include "Animal.h"

#include "Plant.h"

#include "Equipment.h"

#include "Vehicle.h"

#include "Building.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "RU");

cout << setw(1000) << "Сельское хозяйство";

cout << "\nЖивотные";

vector<Animal> animals =

{

Animal("Овца", 4, 80),

Animal("Корова", 10, 450),

Animal("Свинья", 6, 110),

Animal("Коза", 10, 60),

Animal("Курица", 2, 2),

Animal("Утка", 2, 2),

Animal("Лошадь", 8, 500)

};

for (const auto& animal : animals)

{

animal.Out();

}

cout << "\n\nРастения";

vector<Plant> plants =

{

Plant("Рис", 1, 3),

Plant("Картофель", 1, 2),

Plant("Помидоры", 2, 3),

Plant("Огурцы", 1, 3),

Plant("Кукуруза", 3, 5),

Plant("Подсолнух", 2, 6),

Plant("Пшеница", 1, 4)

};

for (const auto& plant : plants)

{

plant.Out();

}

1. Разработать UML – диаграммы (рисунок 13, 14, 15, 16, 17);

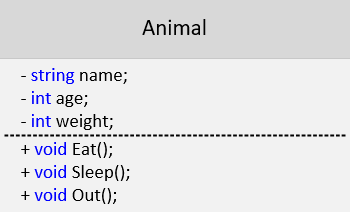


Рисунок 13 – Класс Animal

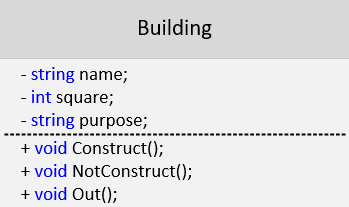


Рисунок 14 – Класс Building

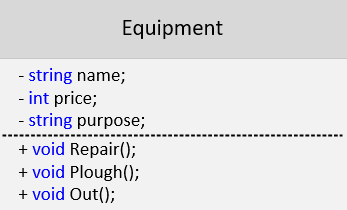


Рисунок 15 – Класс Equipment

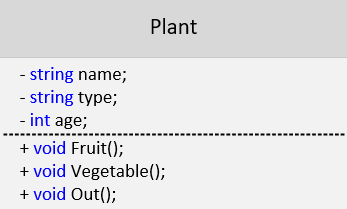


Рисунок 16 – Класс Plant

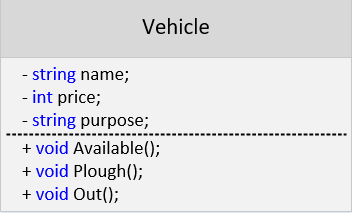


Рисунок 17 – Класс Vehicle

1. Результат (рисунок 18, 19, 20, 21).

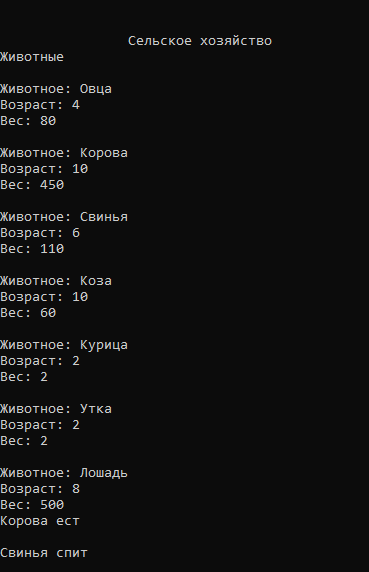
****

Рисунок 18 – Животные

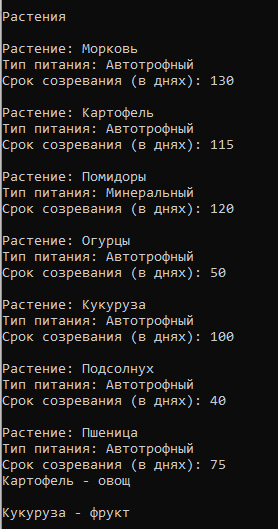
****

Рисунок 19 – Растения

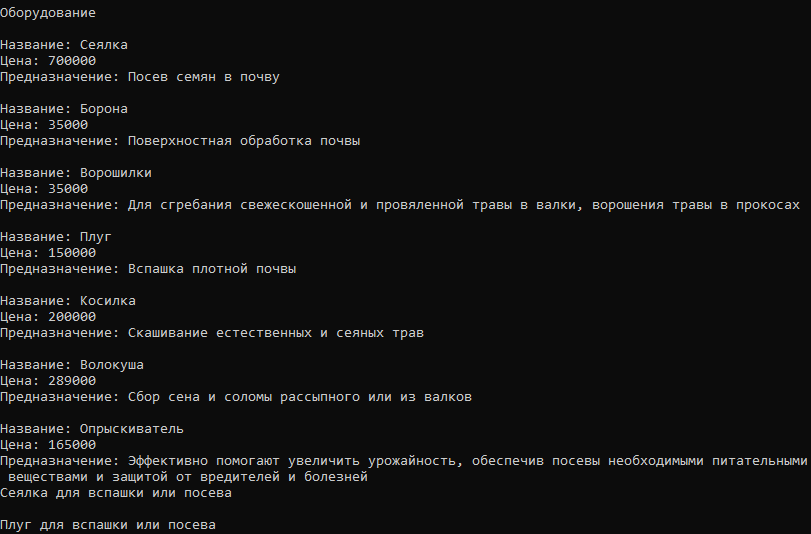


Рисунок 20 – Оборудование

Рисунок 21 – Транспорт

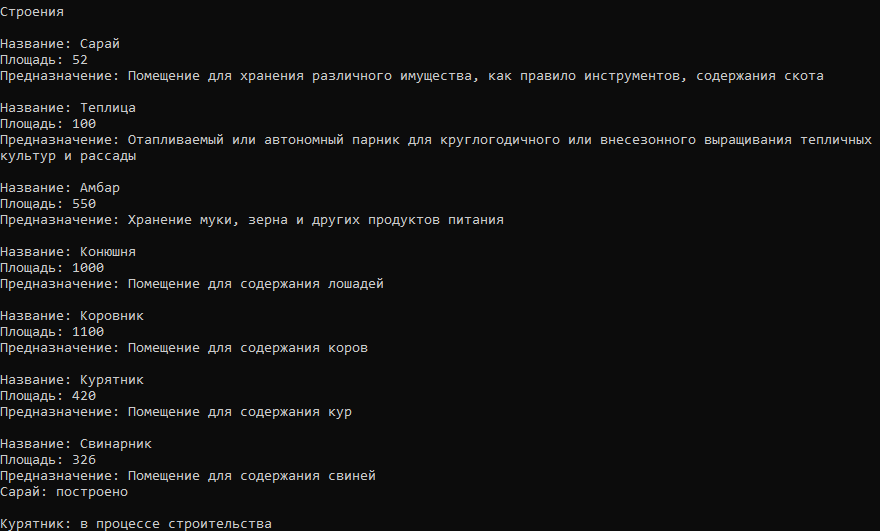


Рисунок 22 – Строения

**Контрольные вопросы**

1. Наследование, инкапсуляция, полиморфизм;
2. Класс – описание объекта; Класс – это модуль, архитектурная единица построения программной системы. Класс – это тип данных, задающий реализацию некоторой абстракции данных, характерной для проблемной области, в интересах которой создается программная система;
3. [атрибуты][модификаторы]class имя\_класса[:список\_родителей] {тело\_класса}. В теле класса могут быть объявлены: константы; поля; конструкторы и деструкторы; методы; события; классы (структуры, делегаты, интерфейсы, перечисления);
4. Диаграмма классов - диаграмма, на которой отображаются классы и их отношения. Является основной диаграммой UML;
5. На диаграмме классов используются следующие основные обозначения: Класс – прямоугольник, разделенный на три секции (раздела), в первом приводится имя класса, во втором – список атрибутов, в третьем – список методов;
6. Атрибуты подобны некоторому дополнительному условию, которое пишется вместе с кодом, и компилятор обязан следовать этому конкретному условию. Эти атрибуты могут быть любым условием или любой информацией, прикрепленной к переменным или функциям.