**ФГБОУ ВО**

**Национальный исследовательский университет**

**«МЭИ»**

**Лабораторная работа № 5**

по курсу

«Технология программирования»

**Множественное наследование**

Вариант 8

**Выполнил:**

студент группы Аэ-21-23

Нестеров А.С.

**Москва, 2024**

1. **Постановка задачи**

**Постановка задачи**

**Часть 1:**

1. Построить иерархию классов: место, область, город, мегаполис;
2. Разработать методы и свойства для каждого из определяемых классов;
3. Реализовать программу на C++ в соответствии с перечисленными выше: место, область, город, мегаполис. Вывести результаты.

**Часть 2:**

1. Изменить иерархию классов с использованием абстрактного виртуального класса в качестве основы иерархии;
2. Реализовать дружественный метод для нескольких классов. Вывести результаты.

**Часть 3:**

1. Реализовать обработку исключительных ситуаций;
2. Рассмотреть и описать, где и какие проблемы возникли при реализации множественного наследования ("алмаз смерти"). Вывести результаты.

**Реализуемые функции:**

* Проверка на валидность ввода данных (наличие цифр в строке, некорректных символов в строке, ввода строки, ввода населения и номера региона);
* Дружественная функция для вывода информации о географических объектах.

**Входные данные:**

* Название города;
* Население города;
* Название региона;
* Номер региона (от 1 до 999);
* Площадь мегаполиса.

**Выходные данные:**

* Соответствуют входным. При отсутствии введенных входных данных выводятся данные по умолчанию.

**Часть I. Иерархия классов**

1. Построение иерархии классов Иерархия классов строится на основе концепции географических объектов, включающих место, область, город и мегаполис.

**~ Класс Place**

Назначение: базовый класс, предоставляет общий функционал для всех географических объектов. Поля:

* Country – название страны, в которой находится географический объект. Методы:
* Деструктор для освобождения памяти;
* Функция display – вывод данных о географическом объекте, включая страну;
* Геттер и сеттер для поля Country.

**~ Класс City (наследуется от Place)**

Назначение: хранит данные о городе. Поля (уникальные свойства):

* CityName – название города;
* CityPopulation – население города. Методы:
* Конструктор с параметрами по умолчанию, включая страну;
* Методы для установки и получения данных – геттеры и сеттеры;
* Переопределенный метод display для вывода информации о городе и стране.

**~ Класс Region (наследующий от Place)**

Назначение: хранит данные о регионе. Поля (уникальные свойства):

* RegionName – название региона;
* RegionNumber – номер региона. Методы:
* Конструктор с параметрами по умолчанию, включая страну;
* Методы для установки и получения данных – геттеры и сеттеры;
* Переопределенный метод display для вывода информации о регионе и стране.

**~ Класс Megapolis (наследующий от City и Region)**

Назначение: хранит данные о мегаполисе. Поля (уникальные свойства):

* area – площадь мегаполиса. Методы:
* Конструктор с параметрами по умолчанию, включая страну, город и регион;
* Методы для установки и получения данных – геттеры и сеттеры;
* Переопределенный метод display для вывода информации о мегаполисе, городе и регионе.

**2.2 Описание полей и методов классов**

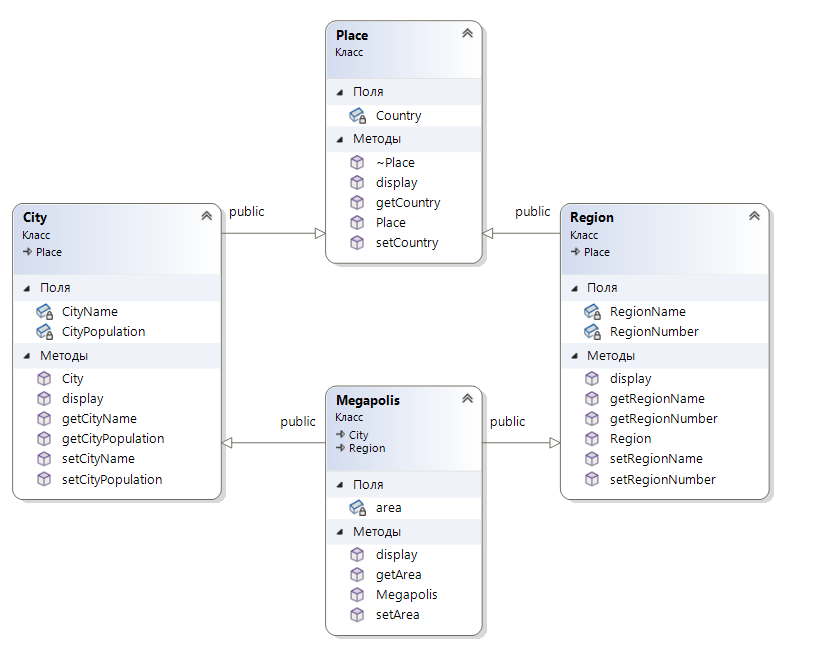


Рис. 1. Диаграмма классов

1. ***Код (реализация классов) и пример вывода результатов***

// Базовый класс

class Place {

private:

string Country; // Название страны

public:

Place(const string& country) : Country(country) {}

virtual void display() const {

cout << "Страна: " << Country << endl;

}

virtual ~Place() {}

// Геттер и сеттер для поля Country

string getCountry() const {

return Country;

}

void setCountry(const string& country) {

if (country.empty()) {

throw runtime\_error("Название страны не может быть пустым.");

}

Country = country;

}

};

// Класс City, наследующий Place

class City : public Place {

private:

string CityName;

int CityPopulation;

public:

City(const string& country, const string& cityName, int cityPopulation)

: Place(country), CityName(cityName), CityPopulation(cityPopulation) {}

void setCityName(const string& cityName) {

if (cityName.empty()) {

throw runtime\_error("Название города не может быть пустым.");

}

this->CityName = cityName;

}

void setCityPopulation(int cityPopulation) {

if (cityPopulation < 0) {

throw runtime\_error("Население не может быть отрицательным.");

}

this->CityPopulation = cityPopulation;

}

// Геттеры

string getCityName() const {

return CityName;

}

int getCityPopulation() const {

return CityPopulation;

}

void display() const override {

Place::display(); // Выводим страну

cout << "Город: " << CityName << ", Население: " << CityPopulation << endl;

}

};

// Класс Region, наследующий Place

class Region : public Place {

private:

string RegionName;

int RegionNumber;

public:

Region(const string& country, const string& regionName, int regionNumber)

: Place(country), RegionName(regionName), RegionNumber(regionNumber) {}

void setRegionName(const string& regionName) {

if (regionName.empty()) {

throw runtime\_error("Название региона не может быть пустым.");

}

this->RegionName = regionName;

}

void setRegionNumber(int regionNumber) {

if (regionNumber < 1 || regionNumber > 999) {

throw runtime\_error("Номер региона должен быть от 1 до 999.");

}

this->RegionNumber = regionNumber;

}

// Геттеры

string getRegionName() const {

return RegionName;

}

int getRegionNumber() const {

return RegionNumber;

}

void display() const override {

Place::display(); // Выводим страну

cout << "Регион: " << RegionName << ", Номер региона: " << RegionNumber << endl;

}

};

// Класс Megapolis, наследующий City и Region

class Megapolis : public City, public Region {

private:

int area; // Площадь мегаполиса

public:

Megapolis(const string& country, const string& cityName, int population,

const string& regionName, int regionNumber, int area)

: City(country, cityName, population), Region(country, regionName, regionNumber), area(area) {

if (area < 0) {

throw runtime\_error("Площадь не может быть отрицательной.");

}

}

void setArea(int area) {

if (area < 0) {

throw runtime\_error("Площадь не может быть отрицательной.");

}

this->area = area;

}

// Геттер для площади

int getArea() const {

return area;

}

void display() const override {

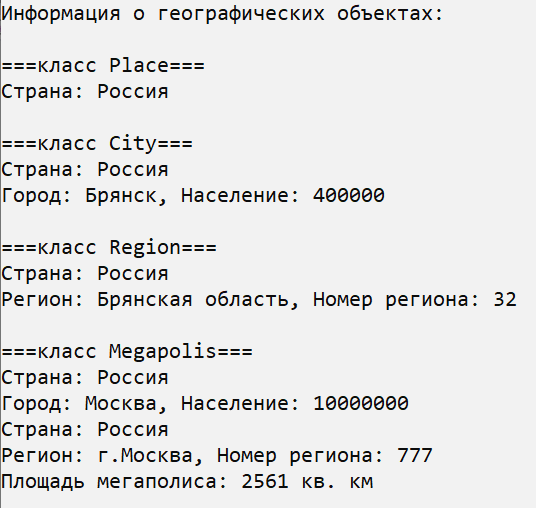
City::display(); // Выводим данные о городе

Region::display(); // Выводим данные о регионе

cout << "Площадь мегаполиса: " << area << " кв. км" << endl;

}

};

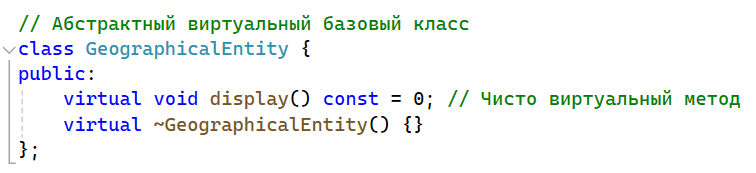


*Рис. 2. Пример вывода данных для части I*

**Часть II. Абстрактный базовый класс**

1. **Изменение структуры классов**

* В данной реализации создается абстрактный виртуальный класс GeographicalEntity, который будет служить базовым классом для всех географических объектов:
* Метод display становится чисто виртуальным, что делает невозможным создание объектов базового класса.
* Производные классы (City, Region, Megapolis) обязаны переопределить метод display.



*Рис. 3. Абстрактный класс GeographicalEntity*

1. **Реализация дружественного метода**

Ко всем наследуемым классам добавлен дружественный метод displayinformation, который:

* Получает ссылки на объекты City, Region и Megapolis.
* Имеет доступ к приватным и защищённым полям.
* Выводит информацию обо всех географических объектах, включая название, население, номер региона и площадь мегаполиса.

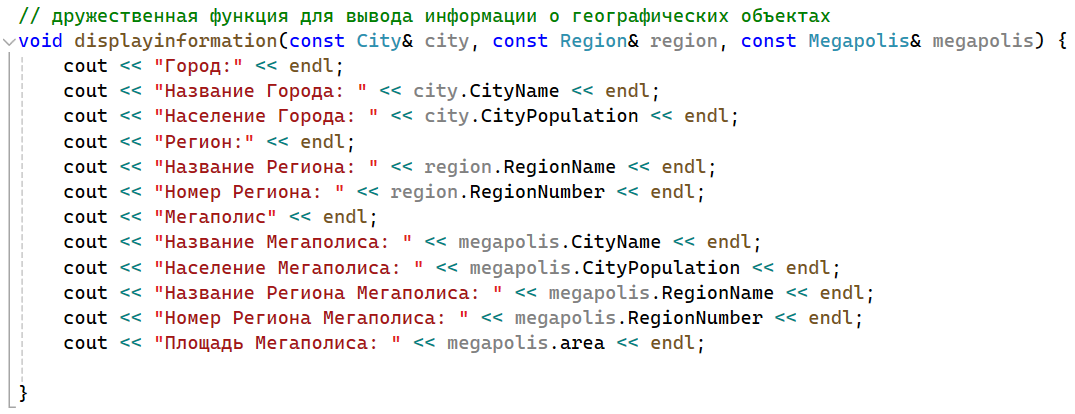
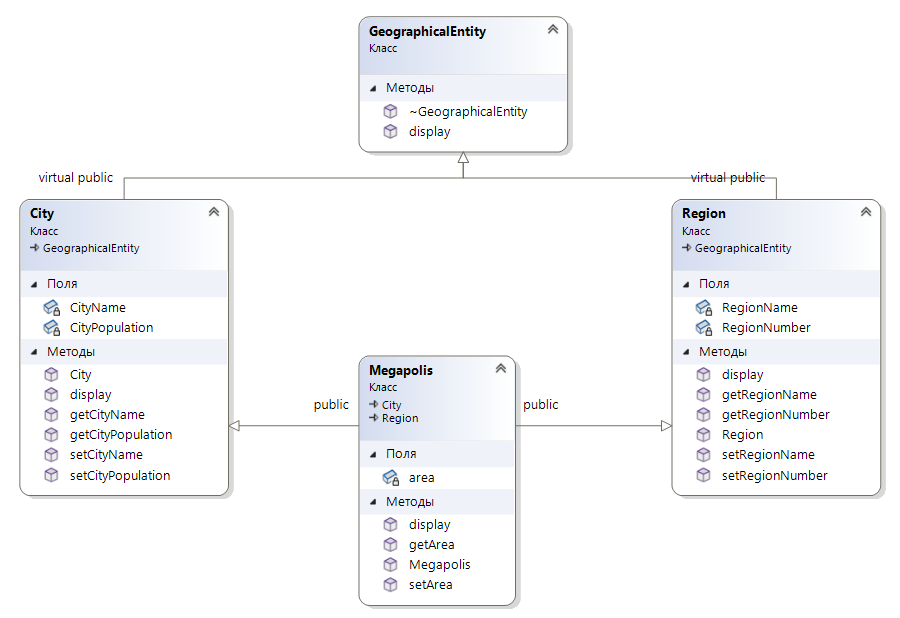


Рис. 4. Реализация дружественного метода displayinformation



*Рис. 5. Диаграмма классов для части II*

1. Код (реализация классов и friend-функции) и пример вывода результатов

// Абстрактный виртуальный базовый класс

class GeographicalEntity {

public:

virtual void display() const = 0; // Чисто виртуальный метод

virtual ~GeographicalEntity() {}

};

// Класс City, наследующий GeographicalEntity

class City : virtual public GeographicalEntity {

private:

string CityName;

int CityPopulation;

public:

City(const string& CityName, int CityPopulation) {

setCityName(CityName);

setCityPopulation(CityPopulation);

}

// Геттеры

string getCityName() const {

return CityName;

}

int getCityPopulation() const {

return CityPopulation;

}

// Сеттеры

void setCityName(const string& CityName) {

if (CityName.empty()) {

throw runtime\_error("Название города не может быть пустым.");

}

this->CityName = CityName;

}

void setCityPopulation(int CityPopulation) {

if (CityPopulation < 0) {

throw runtime\_error("Население не может быть отрицательным.");

}

this->CityPopulation = CityPopulation;

}

void display() const override {

cout << "Город: " << CityName << ", Население: " << CityPopulation << endl;

}

friend void displayinformation(const City& city, const Region& region, const Megapolis& megapolis);

};

// Класс Region, наследующий GeographicalEntity

class Region : virtual public GeographicalEntity {

private:

string RegionName;

int RegionNumber;

public:

Region(const string& RegionName, int RegionNumber) {

setRegionName(RegionName);

setRegionNumber(RegionNumber);

}

// Геттеры

string getRegionName() const {

return RegionName;

}

int getRegionNumber() const {

return RegionNumber;

}

// Сеттеры

void setRegionName(const string& RegionName) {

if (RegionName.empty()) {

throw runtime\_error("Название региона не может быть пустым.");

}

this->RegionName = RegionName;

}

void setRegionNumber(int RegionNumber) {

if (RegionNumber < 1 || RegionNumber > 999) {

throw runtime\_error("Номер региона должен быть от 1 до 999.");

}

this->RegionNumber = RegionNumber;

}

void display() const override {

cout << "Регион: " << RegionName << ", Номер региона: " << RegionNumber << endl;

}

friend void displayinformation(const City& city, const Region& region, const Megapolis& megapolis);

};

// Класс Megapolis, наследующий City и Region

class Megapolis : public City, public Region {

private:

int area; // Площадь мегаполиса

public:

Megapolis(const string& cityName, int population, const string& regionName, int regionNumber, int area)

: City(cityName, population), Region(regionName, regionNumber), area(area) {

if (area < 0) {

throw runtime\_error("Площадь не может быть отрицательной.");

}

}

// Геттеры

int getArea() const {

return area;

}

// Сеттер

void setArea(int area) {

if (area < 0) {

throw runtime\_error("Площадь не может быть отрицательной.");

}

this->area = area;

}

void display() const override {

City::display();

Region::display();

cout << "Площадь мегаполиса: " << area << " кв. км" << endl;

}

friend void displayinformation(const City& city, const Region& region, const Megapolis& megapolis);

};

// дружественная функция для вывода информации о географических объектах

void displayinformation(const City& city, const Region& region, const Megapolis& megapolis) {

cout << "Город:" << endl;

cout << "Название Города: " << city.CityName << endl;

cout << "Население Города: " << city.CityPopulation << endl;

cout << "Регион:" << endl;

cout << "Название Региона: " << region.RegionName << endl;

cout << "Номер Региона: " << region.RegionNumber << endl;

cout << "Мегаполис" << endl;

cout << "Название Мегаполиса: " << megapolis.CityName << endl;

cout << "Население Мегаполиса: " << megapolis.CityPopulation << endl;

cout << "Название Региона Мегаполиса: " << megapolis.RegionName << endl;

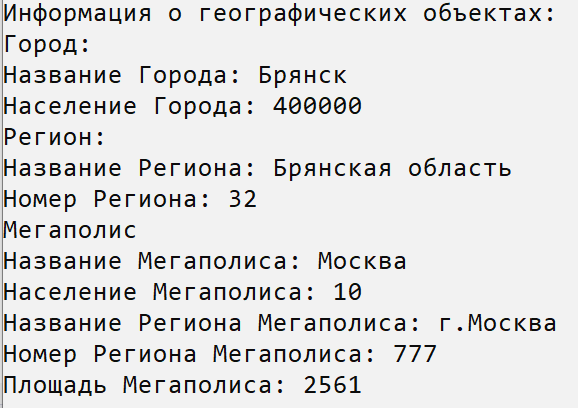
cout << "Номер Региона Мегаполиса: " << megapolis.RegionNumber << endl;

cout << "Площадь Мегаполиса: " << megapolis.area << endl;

}

***Пример работы программы:***

Созданы географические объекты и выведены дружественной функцией (процедурой) displayinformation:



*Рис. 6. Пример вывода данных для части II*

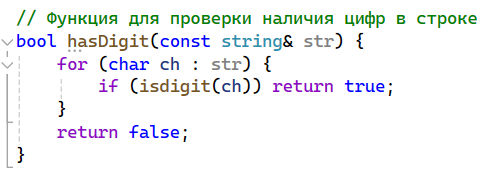
**Часть III. Обработка исключений и валидация ввода**

1. Обработка исключений В данной реализации предусмотрена обработка исключительных ситуаций, связанных с некорректным вводом данных. Для этого используются функции валидации, которые проверяют входные данные на корректность:

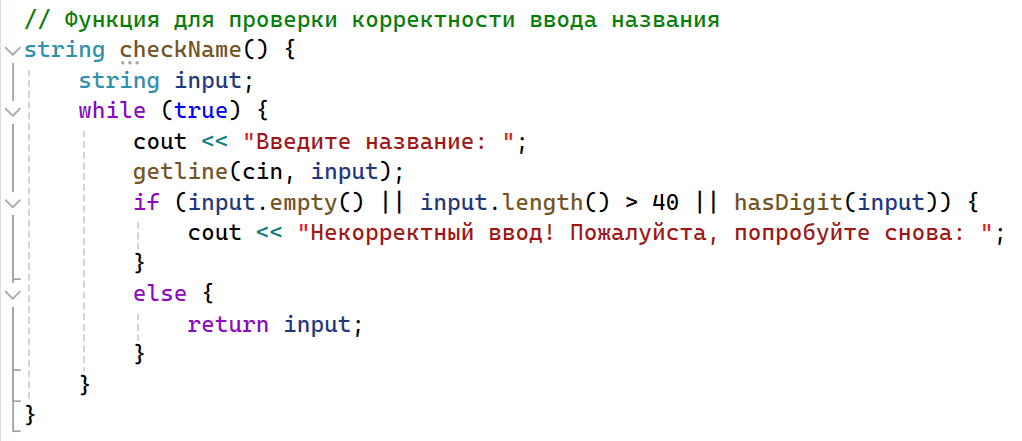
* **Проверка ввода для строк, чисел и специальных форматов** осуществляется с помощью следующих вспомогательных функций:
  + **hasDigit**: проверяет наличие цифр в строке.
  + **checkName**: проверяет корректность ввода названия города или региона, исключая пустые строки и строки, содержащие цифры.
  + **checkPopulation**: проверяет корректность ввода населения, требуя положительное целое число.
  + **checkRegionNumber**: проверяет корректность ввода номера региона, ограничивая его диапазон от 1 до 999.
  + **checkChoice**: проверяет, что вводится только одна буква.

Принцип работы функций:

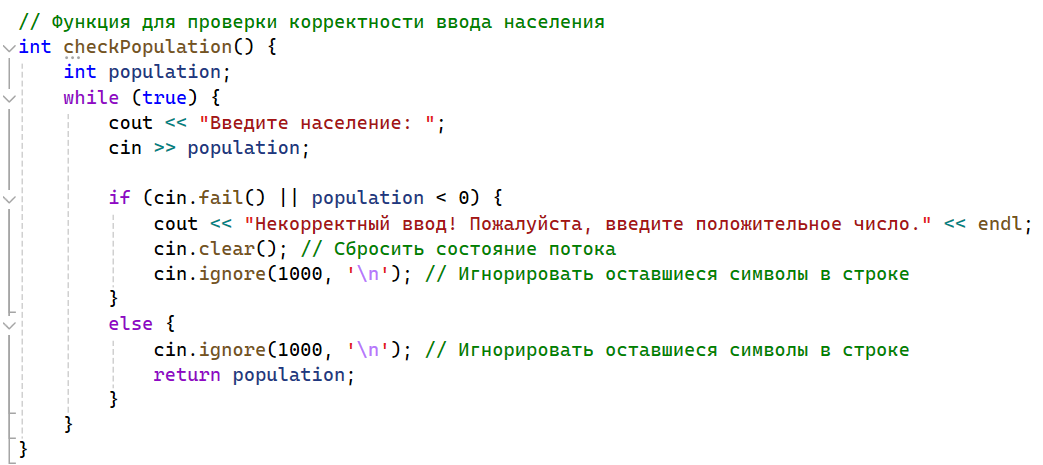
* Бесконечный цикл проверяет ввод данных.
* При некорректных данных отображается сообщение об ошибке.
* После корректного ввода значение возвращается.

**

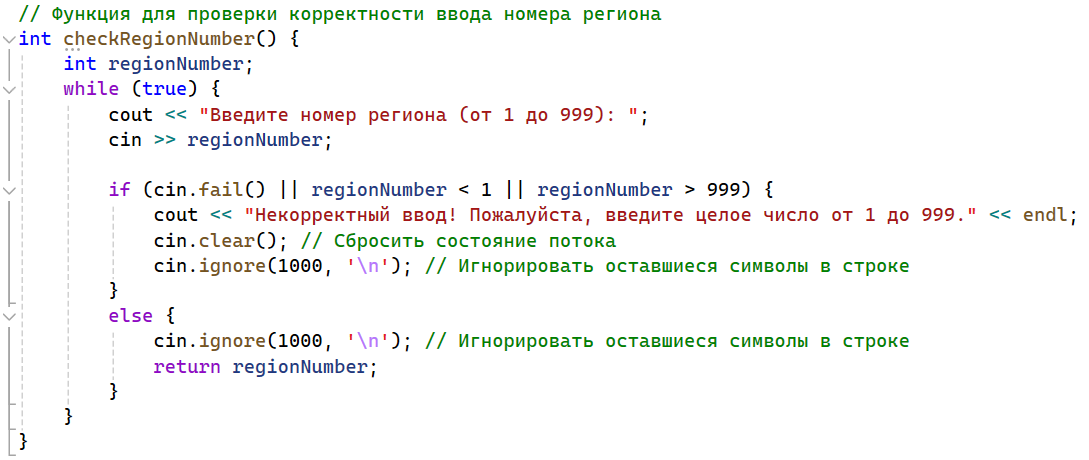
*Рис. 6. Реализация функции hasDigit*

**

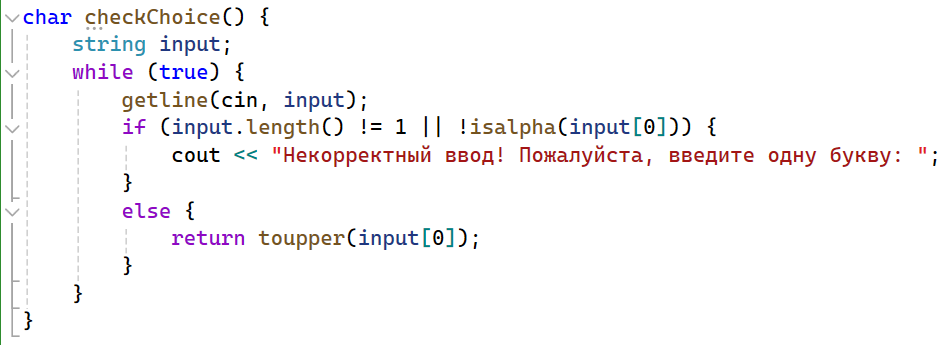
*Рис. 7. Реализация функции checkName*

**

*Рис. 8. Реализация функции checkPopulation*



*Рис. 9. Реализация функции checkRegionNumber*



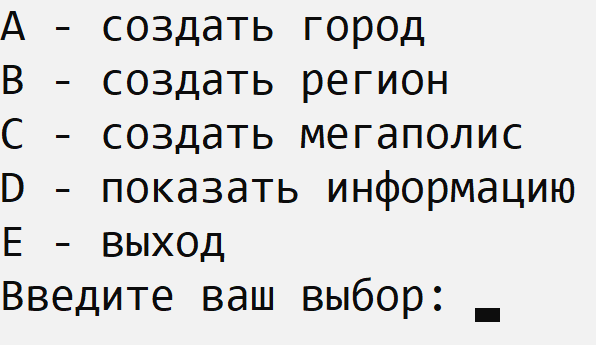
*Рис. 10. Реализация функции checkChoice*

1. Проблемы множественного наследования

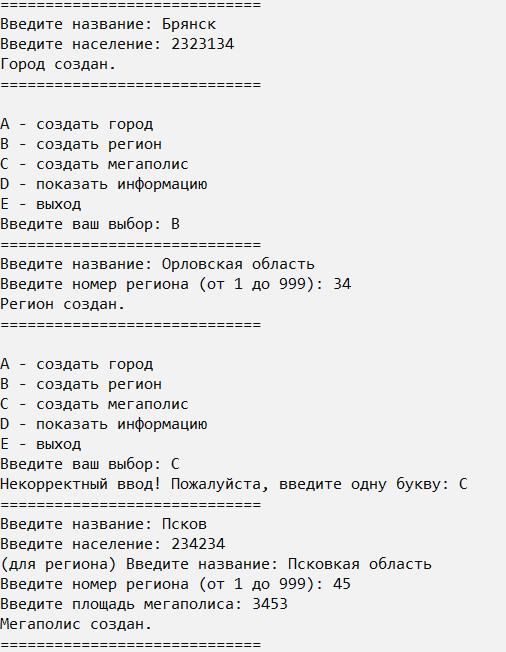
* **Дублирование базового класса**: Если не использовать виртуальное наследование, то класс **Megapolis** получит две копии базового класса **GeographicalEntity** через классы **City** и **Region**. Это может привести к дублированию состояния и конфликтам вызова методов.
* **Необходимость явного вызова конструктора**: Виртуальное наследование требует, чтобы единственный экземпляр базового класса инициализировался в конструкторе конечного класса (**Megapolis**). Это может быть забыто, особенно если структура наследования сложна.
* **Конфликт методов и неявное использование интерфейса**: Методы **display** определены в базовом классе, но переопределяются в производных классах. Это может привести к путанице при вызове методов.
* **Опасность приведения указателей**: Использование указателя **GeographicalEntity\*** для работы с экземплярами **City**, **Region** или **Megapolis** может дать неожиданные результаты из-за множественного наследования.

1. Использование виртуального наследования Для решения проблем, связанных с множественным наследованием, было использовано виртуальное наследование. Это предотвращает дублирование базового класса и обеспечивает правильную инициализацию объектов. В приложении представлены интерфейсы программы и функциональные тесты, а также полный код программы.

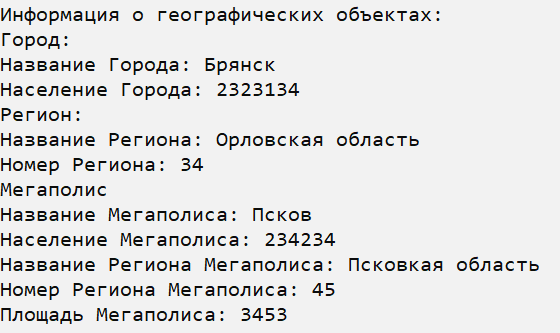
В **Приложении** представлены изображения интерфейса программы и функциональных тестов, а также полный код программы.



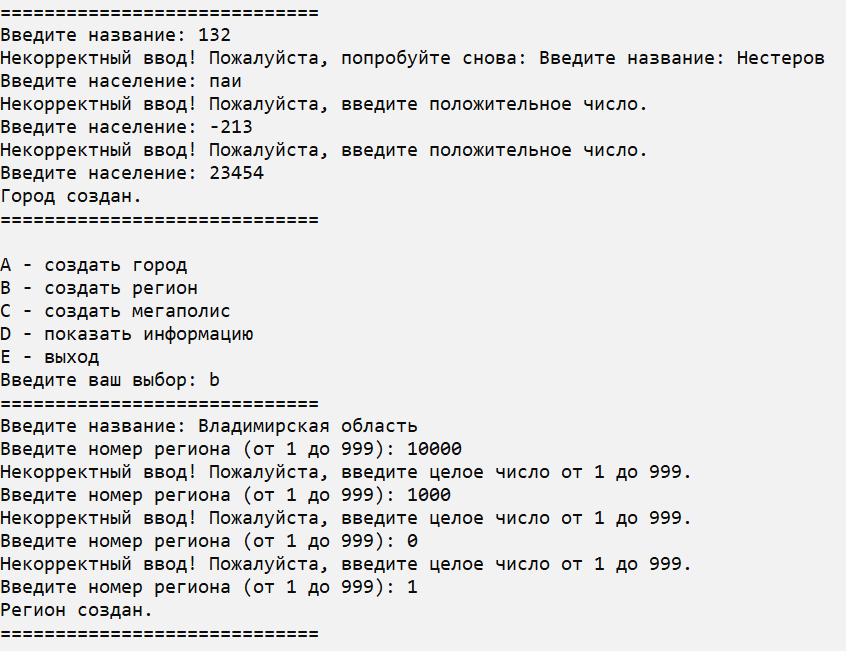
*Рис. 11. Главное меню*



*Рис. 12. Ввод значений*



*Рис. 13. Вывод значений*



*Рис. 13. Проверка исключительных ситуаций при вводе*

***Программный код для трех частей:***

#include <iostream>

#include <string>

#include <stdexcept>

#include <windows.h>

using namespace std;

// Впереди объявление классов

class Region; // Предварительное объявление класса Region

class Megapolis; // Предварительное объявление класса Megapolis

// Функция для проверки наличия цифр в строке

bool hasDigit(const string& str) {

for (char ch : str) {

if (isdigit(ch)) return true;

}

return false;

}

// Функция для проверки корректности ввода названия

string checkName() {

string input;

while (true) {

cout << "Введите название: ";

getline(cin, input);

if (input.empty() || input.length() > 40 || hasDigit(input)) {

cout << "Некорректный ввод! Пожалуйста, попробуйте снова: ";

}

else {

return input;

}

}

}

// Функция для проверки корректности ввода населения

int checkPopulation() {

int population;

while (true) {

cout << "Введите население: ";

cin >> population;

if (cin.fail() || population < 0) {

cout << "Некорректный ввод! Пожалуйста, введите положительное число." << endl;

cin.clear(); // Сбросить состояние потока

cin.ignore(1000, '\n'); // Игнорировать оставшиеся символы в строке

}

else {

cin.ignore(1000, '\n'); // Игнорировать оставшиеся символы в строке

return population;

}

}

}

// Функция для проверки корректности ввода номера региона

int checkRegionNumber() {

int regionNumber;

while (true) {

cout << "Введите номер региона (от 1 до 999): ";

cin >> regionNumber;

if (cin.fail() || regionNumber < 1 || regionNumber > 999) {

cout << "Некорректный ввод! Пожалуйста, введите целое число от 1 до 999." << endl;

cin.clear(); // Сбросить состояние потока

cin.ignore(1000, '\n'); // Игнорировать оставшиеся символы в строке

}

else {

cin.ignore(1000, '\n'); // Игнорировать оставшиеся символы в строке

return regionNumber;

}

}

}

char checkChoice() {

string input;

while (true) {

getline(cin, input);

if (input.length() != 1 || !isalpha(input[0])) {

cout << "Некорректный ввод! Пожалуйста, введите одну букву: ";

}

else {

return toupper(input[0]);

}

}

}

// Абстрактный виртуальный базовый класс

class GeographicalEntity {

public:

virtual void display() const = 0; // Чисто виртуальный метод

virtual ~GeographicalEntity() {}

};

// Класс City, наследующий GeographicalEntity

class City : virtual public GeographicalEntity {

private:

string CityName;

int CityPopulation;

public:

City(const string& CityName, int CityPopulation) {

setCityName(CityName);

setCityPopulation(CityPopulation);

}

// Геттеры

string getCityName() const {

return CityName;

}

int getCityPopulation() const {

return CityPopulation;

}

// Сеттеры

void setCityName(const string& CityName) {

if (CityName.empty()) {

throw runtime\_error("Название города не может быть пустым.");

}

this->CityName = CityName;

}

void setCityPopulation(int CityPopulation) {

if (CityPopulation < 0) {

throw runtime\_error("Население не может быть отрицательным.");

}

this->CityPopulation = CityPopulation;

}

void display() const override {

cout << "Город: " << CityName << ", Население: " << CityPopulation << endl;

}

friend void displayinformation(const City& city, const Region& region, const Megapolis& megapolis);

};

// Класс Region, наследующий GeographicalEntity

class Region : virtual public GeographicalEntity {

private:

string RegionName;

int RegionNumber;

public:

Region(const string& RegionName, int RegionNumber) {

setRegionName(RegionName);

setRegionNumber(RegionNumber);

}

// Геттеры

string getRegionName() const {

return RegionName;

}

int getRegionNumber() const {

return RegionNumber;

}

// Сеттеры

void setRegionName(const string& RegionName) {

if (RegionName.empty()) {

throw runtime\_error("Название региона не может быть пустым.");

}

this->RegionName = RegionName;

}

void setRegionNumber(int RegionNumber) {

if (RegionNumber < 1 || RegionNumber > 999) {

throw runtime\_error("Номер региона должен быть от 1 до 999.");

}

this->RegionNumber = RegionNumber;

}

void display() const override {

cout << "Регион: " << RegionName << ", Номер региона: " << RegionNumber << endl;

}

friend void displayinformation(const City& city, const Region& region, const Megapolis& megapolis);

};

// Класс Megapolis, наследующий City и Region

class Megapolis : public City, public Region {

private:

int area; // Площадь мегаполиса

public:

Megapolis(const string& cityName, int population, const string& regionName, int regionNumber, int area)

: City(cityName, population), Region(regionName, regionNumber), area(area) {

if (area < 0) {

throw runtime\_error("Площадь не может быть отрицательной.");

}

}

// Геттеры

int getArea() const {

return area;

}

// Сеттер

void setArea(int area) {

if (area < 0) {

throw runtime\_error("Площадь не может быть отрицательной.");

}

this->area = area;

}

void display() const override {

City::display();

Region::display();

cout << "Площадь мегаполиса: " << area << " кв. км" << endl;

}

friend void displayinformation(const City& city, const Region& region, const Megapolis& megapolis);

};

// дружественная функция для вывода информации о географических объектах

void displayinformation(const City& city, const Region& region, const Megapolis& megapolis) {

cout << "Город:" << endl;

cout << "Название Города: " << city.CityName << endl;

cout << "Население Города: " << city.CityPopulation << endl;

cout << "Регион:" << endl;

cout << "Название Региона: " << region.RegionName << endl;

cout << "Номер Региона: " << region.RegionNumber << endl;

cout << "Мегаполис" << endl;

cout << "Название Мегаполиса: " << megapolis.CityName << endl;

cout << "Население Мегаполиса: " << megapolis.CityPopulation << endl;

cout << "Название Региона Мегаполиса: " << megapolis.RegionName << endl;

cout << "Номер Региона Мегаполиса: " << megapolis.RegionNumber << endl;

cout << "Площадь Мегаполиса: " << megapolis.area << endl;

}

// Главная функция

int main() {

setlocale(LC\_ALL, "Russian");

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

system("color F0");

char choice;

string cityName, regionName;

int population, regionNumber, area;

City myCity("Брянск", 400000);

Region myRegion("Брянская область", 32);

Megapolis myMegapolis("Москва", 10, "г.Москва", 777, 2561);

do {

cout << "\nA - создать город\n"

<< "B - создать регион\n"

<< "C - создать мегаполис\n"

<< "D - показать информацию\n"

<< "E - выход\n"

<< "Введите ваш выбор: ";

choice = checkChoice();

//cin >> choice;

//cin.ignore(); // Игнорировать оставшиеся символы в строке

try {

switch (choice) {

case 'A':

cout << "=============================" << endl;

cityName = checkName();

population = checkPopulation();

myCity.setCityName(cityName);

myCity.setCityPopulation(population);

cout << "Город создан." << endl;

cout << "=============================" << endl;

break;

case 'B':

cout << "=============================" << endl;

regionName = checkName();

regionNumber = checkRegionNumber();

myRegion.setRegionName(regionName);

myRegion.setRegionNumber(regionNumber);

cout << "Регион создан." << endl;

cout << "=============================" << endl;

break;

case 'C':

cout << "=============================" << endl;

cityName = checkName();

population = checkPopulation();

cout << "(для региона) ";

regionName = checkName();

regionNumber = checkRegionNumber();

cout << "Введите площадь мегаполиса: ";

cin >> area;

myMegapolis.setRegionName(regionName);

myMegapolis.setRegionNumber(regionNumber);

myMegapolis.setArea(area);

myMegapolis.setCityName(cityName);

myMegapolis.setCityPopulation(population);

cout << "Мегаполис создан." << endl;

cout << "=============================" << endl;

break;

case 'D':

// Здесь можно добавить код для отображения информации о созданных объектах

cout << "Информация о географических объектах:" << endl;

displayinformation(myCity, myRegion, myMegapolis);

// Например, myCity.display(); и т.д.

cout << "=============================" << endl;

break;

case 'E':

return 0;

default:

cout << "Некорректный ввод! Пожалуйста, попробуйте снова." << endl;

}

}

catch (const exception& e) {

cout << "Произошла ошибка: " << e.what() << endl;

}

} while (true);

return 0;

}