МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Наследование. Полиморфизм»**

по дисциплине: «*Программирование*»

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-320», «АВТФ»  *Мартынова Е.Д.*  «10» января 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Г.А.*  «\_\_\_» \_\_\_\_\_\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

Оглавление

[Задание к работе: 3](#_Toc187389327)

[Методика выполнения работы: 3](#_Toc187389328)

[Задание по варианту 8 4](#_Toc187389329)

[Диаграмма классов в нотации UML 6](#_Toc187389330)

[Листинг программы 7](#_Toc187389331)

[main.cpp 7](#_Toc187389332)

[stagesGame.cpp 7](#_Toc187389333)

[class.h 7](#_Toc187389334)

[header.h 7](#_Toc187389335)

[Результат работы программы 7](#_Toc187389336)

[Вывод 9](#_Toc187389337)

[Ссылки на репозитории 11](#_Toc187389338)

Задание к работе: Написать программу с использованием принципов ООП и паттернов программирования.

# Методика выполнения работы:

1. Разработать алгоритмы решения задачи по индивидуальному заданию. Задание выбирается студентом самостоятельно. Вес работы определяет минимальный балл за лабораторную работу.
2. Написать и отладить программы решения задачи.
3. Протестировать работу программ на различных исходных данных.
4. По запросу преподавателя быть готовым модифицировать/добавить функционал программы.
5. Ответить на теоретические вопросы к лабораторной работе на выбор преподавателя.

# Задание по варианту 8

Галактическая экономика "Империя звезд"

Вы управляете галактической экономикой, где несколько планет торгуют ресурсами. У каждой планеты есть уникальные характеристики: производительность, спрос, технология. Игрок должен построить торговые маршруты между планетами, чтобы оптимизировать прибыль, учитывая затраты натранспортировку и конкуренцию между торговыми корпорациями. Базовый класс CelestialBody описывает общее поведение космических объектов.

Подкласс Planet описывает планету с уникальными характеристиками.

Подкласс Asteroid представляет астероидные пояса, которые можно добывать за дополнительную плату.

Экономика:

1. Планеты производят ресурсы (ресурс и объем производства) и
2. потребляют другие ресурсы (спрос).
3. Цена ресурса определяется спросом и предложением.
4. Транспортировка между планетами требует затрат топлива.

Корпорации:

1. Базовый класс Corporation описывает торговую корпорацию.
2. Подклассы:
3. LogisticsCompany — минимизирует транспортные расходы.
4. TechTrader — торгует только высокотехнологичными ресурсами.
5. MinerCompany — добывает ресурсы на астероидах.

Галактика:

1. Представлена как граф с вершинами (планеты) и рёбрами (торговые маршруты).
2. Каждый маршрут имеет цену и уровень безопасности.

Условия:

1. Игра начинается с создания галактики: игрок задаёт количество планет, маршрутов и типов корпораций.
2. За каждый ход планеты производят ресурсы, корпорации транспортируют их, и игрок получает прибыль.
3. Игрок может инвестировать в:
   1. Улучшение маршрутов (снижение затрат топлива).
   2. Развитие планет (увеличение производительности).
4. Выигрывает игрок, который достигнет наибольшей прибыли за указанное количество ходов.

# Диаграмма классов в нотации UML

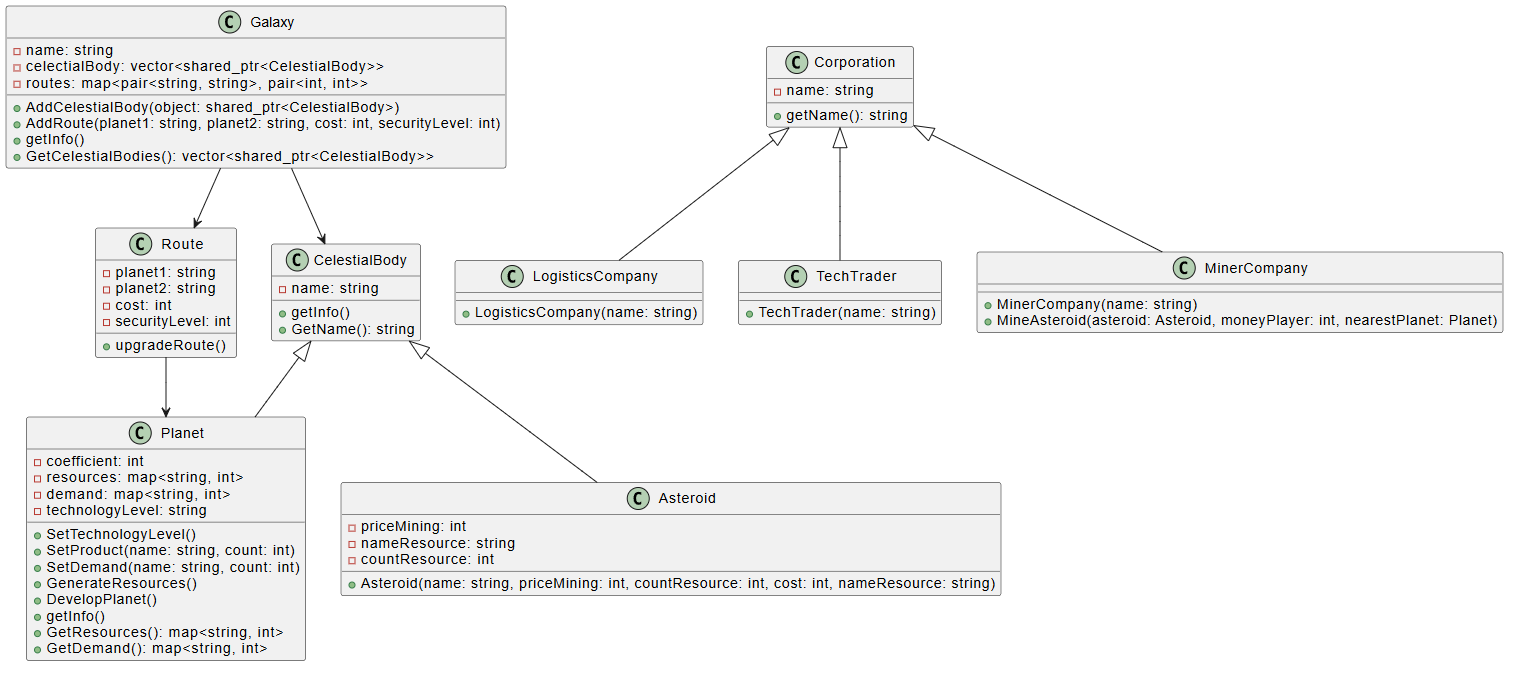


Рисунок 1. UML-диаграмма классов.

# Листинг программы

## main.cpp

#include <iostream>

#include <vector>

#include <memory>

#include <thread> // this\_thread::sleep\_for

#include <chrono> // chrono::milliseconds

#include <sstream>

#include <algorithm>

using namespace std;

#include <map>

#include "header.h"

int main() {

setlocale(LC\_ALL, "rus");

Galaxy galaxy;

PrintText("ДОБРО ПОЖАЛОВАТЬ В ИГРУ <ИМПЕРИЯ ЗВЕЗД>!!!", 20);

cout << endl;

this\_thread::sleep\_for(chrono::milliseconds(500));

PrintText("ПРАВИЛА ОЧЕНЬ ПРОСТЫ: СОЗДАЙТЕ ПЛАНЕТЫ, МАРШРУТЫ МЕЖДУ НИМИ, ГАЛАКТИЧЕСКИЕ КОРПОРАЦИИ, А ЗАТЕМ ЗАРАБАТЫВАЙТЕ НА ТОРГОВЛЕ!", 20);

cout << endl;

PrintText("ПОБЕДИТ ИГРОК, КОТОРЫЙ СМОГ ДОСТИЧЬ МАКСИМАЛЬНОЙ ПРИБЫЛИ.", 20);

cout << endl << endl;

this\_thread::sleep\_for(chrono::milliseconds(500));

PrintText("УДАЧИ!!!", 20);

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

system("clear");

PrintText("Количество игроков для игры: 2", 20);

PrintText("Количество раундов: 3", 20);

cout << endl << endl;

PrintText("Каждому игроку в начале игры выдается 10000 шекелей, 600 единиц топлива и 3 планеты, каждой из которых нужно выбрать главный производимый ресурс (Одна планета принадлежит Игроку);", 20);

PrintText("Создать маршруты между этими планетами, которые будут рандомно затрачивать топливо для пути и формировать уровень безопасности;", 20);

PrintText("Также необходимо выбрать два вида корпорации, с которыми игрок будет сотрудничать.", 20);

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

int countPlayers = 2;

int countRounds = 3;

int moneyPlayer1 = 10000;

int moneyPlayer2 = 10000;

int fuelPlayer1 = 600;

int fuelPlayer2 = 600;

vector<shared\_ptr<Corporation>> corporationsPlayer1;

vector<shared\_ptr<Corporation>> corporationsPlayer2;

CreatePlanets(galaxy, "Игрок 1");

vector<string> resources1 = ChooseResources(galaxy, 1);

if (resources1.empty()) return 0;

vector<Route> routes1;

system("clear");

galaxy.getInfo();

cout << endl;

PrintText("Далее Вам будет необходимо построить маршруты между планетами, поэтому выбирайте с умом!", 20);

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

cout << endl;

CreateRoutes(galaxy, routes1);

vector<int> CorporationPlayer1;

ChooseCorporations(corporationsPlayer1, galaxy);

galaxy.getInfo();

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

CreatePlanets(galaxy, "Игрок 2");

vector<string> resources2 = ChooseResources(galaxy, 2);

if (resources2.empty()) return 0;

vector<Route> routes2;

system("clear");

galaxy.getInfo();

cout << endl;

PrintText("Далее Вам будет необходимо построить маршруты между планетами, поэтому выбирайте с умом!", 20);

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

cout << endl;

CreateRoutes(galaxy, routes2);

vector<int> CorporationPlayer2;

ChooseCorporations(corporationsPlayer2, galaxy);

galaxy.getInfo();

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

system("clear");

PrintText("Галактика обновлена!", 1);

this\_thread::sleep\_for(chrono::milliseconds(2000));

// Основной цикл игры

for (int round = 1; round <= countRounds; round++) {

system("clear");

PrintText("РАУНД: ", 20);

cout << round << endl;

PrintText("ХОД ИГРОКА 1 ", 20);

TradeBetweenPlanets(routes1, galaxy, moneyPlayer1, fuelPlayer1, corporationsPlayer1);

MakeInvestment(moneyPlayer1, routes1, galaxy);

cout << endl << endl << "Нажмите клавишу Enter для продолжения...";

cin.ignore();

system("clear");

PrintText("ХОД ИГРОКА 2 ", 20);

TradeBetweenPlanets(routes2, galaxy, moneyPlayer2, fuelPlayer2, corporationsPlayer2);

MakeInvestment(moneyPlayer2, routes2, galaxy);

this\_thread::sleep\_for(chrono::milliseconds(1000));

cout << endl;

PrintText("Раунд завершён...", 20);

this\_thread::sleep\_for(chrono::milliseconds(1000));

PrintText("Вывод информации о планете...", 20);

this\_thread::sleep\_for(chrono::milliseconds(1000));

galaxy.getInfo();

cout << endl;

PrintText("Количество шекелей у Игрока 1: ", 20);

cout << moneyPlayer1;

cout << endl;

PrintText("Количество шекелей у Игрока 2: ", 20);

cout << moneyPlayer2;

string temp;

cout << endl << endl << "Введите любую клавишу для продолжения...";

cin >> temp;

}

PrintGameResults(moneyPlayer1, moneyPlayer2);

return 0;

}

## stagesGame.cpp

#include <iostream>

#include <vector>

#include <memory>

#include <thread> // this\_thread::sleep\_for

#include <chrono> // chrono::milliseconds

#include <sstream>

#include <algorithm>

using namespace std;

#include <map>

#include "header.h"

void PrintText(const string& text, int milliseconds) {

for (char ch : text) {

cout << ch << flush; // Выводим символ без перехода на новую строку

this\_thread::sleep\_for(chrono::milliseconds(milliseconds)); // Задержка

}

cout << endl;

}

void CreatePlanets(Galaxy& galaxy, const string& playerName) {

system("clear");

cout << endl;

PrintText("ХОД " + playerName, 20);

cout << endl;

string temp;

PrintText("Введите название планет через пробел: ", 20);

getline(cin, temp);

stringstream ss(temp);

string planetName;

while (ss >> planetName) {

shared\_ptr<CelestialBody> planet = make\_shared<Planet>(planetName);

galaxy.AddCelestialBody(planet);

}

}

vector<string> ChooseResources(Galaxy& galaxy, int currentPlayer) {

PrintText("Выберите для своих планет 3 производимых ресурса и введите через пробел: ", 20);

cout << "Древесина, Каучук, Еда, Сталь, Нефть, Железо, Медикаменты\n==>> ";

string temp1;

getline(cin, temp1);

stringstream ssResources(temp1);

vector<string> resources;

string resource;

while (ssResources >> resource && resources.size() < 3) {

resources.push\_back(resource);

}

if (resources.size() != 3) {

PrintText("Ошибка! Выберите ровно три ресурса!", 20);

return {};

}

auto celestialBodies = galaxy.GetCelestialBodies();

vector<shared\_ptr<Planet>> planets;

for (auto& body : celestialBodies) {

auto planet = dynamic\_pointer\_cast<Planet>(body);

if (planet) {

planets.push\_back(planet);

}

}

int startIndex = (currentPlayer == 1) ? 0 : 3;

for (int i = 0; i < 3; ++i) {

int planetIndex = startIndex + i;

auto& planet = planets[planetIndex];

string& res = resources[i];

int resourceCount = rand() % 300 + 1;

planet->SetProduct(res, resourceCount);

// Спрос для текущего ресурса и двух следующих по порядку

for (int j = 1; j <= 2; ++j) {

int demandIndex = (i + j) % 3;

string& nextRes = resources[demandIndex];

int demandCount = rand() % 1000 + 1;

planet->SetDemand(nextRes, demandCount);

}

planet->SetTechnologyLevel();

}

return resources;

}

void CreateRoutes(Galaxy& galaxy, vector<Route>& routes) {

int countRoutes;

PrintText("Введите количество маршрутов (максимально 3): ", 20);

cin >> countRoutes;

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

string planet1, planet2;

int cost, securityLevel;

PrintText("Введите названия двух планет для маршрута (через пробел): ", 20);

cin >> planet1 >> planet2;

cost = rand() % 1000 + 1;

securityLevel = rand() % 50 + 1;

galaxy.AddRoute(planet1, planet2, cost, securityLevel);

}

}

void ChooseCorporations(vector<shared\_ptr<Corporation>>& corporationsPlayer, Galaxy& galaxy) {

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

PrintText("Введите цифры видов корпораций через пробел, с которыми Игрок будет сотрудничать (максимально 2)", 20);

cout << "1 - LogisticsCompany — минимизирует транспортные расходы.\n";

cout << "2 - TechTrader — торгует только высокотехнологичными ресурсами.\n";

cout << "3 - MinerCompany — добывает ресурсы на астероидах.\n";

string temp11;

getline(cin, temp11);

stringstream ssCorp(temp11);

string corporation;

int count = 0;

while (getline(ssCorp, corporation, ' ') && count < 2) {

int corpType = stoi(corporation);

shared\_ptr<Corporation> corp;

switch (corpType) {

case 1:

corp = make\_shared<LogisticsCompany>("LogisticsCompany");

break;

case 2:

corp = make\_shared<TechTrader>("TechTrader");

break;

case 3:

corp = make\_shared<MinerCompany>("MinerCompany");

break;

default:

cout << "Некорректный ввод: " << corporation << ". Корпорация не добавлена." << endl;

continue;

}

if (corp) {

corporationsPlayer.push\_back(corp);

count++;

cout << "Корпорация добавлена: " << corp->getName() << endl;

}

}

}

void MakeInvestment(int& money, vector<Route>& routes, Galaxy& galaxy) {

PrintText("Вы можете инвестировать в улучшение маршрутов или в развитие планет!", 20);

cout << "1 - Улучшение маршрутов (1456 шекелей за уменьшение на 50 единиц топлива)\n";

cout << "2 - Развитие планет (778 шекелей за увеличение на 10 единиц для каждого ресурса)\n";

int investmentChoice;

cout << "Введите ваш выбор: ";

cin >> investmentChoice;

if (investmentChoice == 1) {

PrintText("Вы улучшили маршруты! Затраты топлива снижены.", 20);

for (auto& route : routes) {

route.upgradeRoute();

}

money -= 1456;

}

else if (investmentChoice == 2) {

PrintText("Вы инвестировали в развитие планет!", 20);

for (auto& body : galaxy.GetCelestialBodies()) {

shared\_ptr<Planet> planet = dynamic\_pointer\_cast<Planet>(body);

if (planet) {

planet->DevelopPlanet();

}

}

money -= 778;

}

else {

PrintText("Неверный выбор. Попробуйте снова.", 20);

}

}

int CalculatePrice(const string& resource, int demand, int supply) {

int basePrice = 10;

if (supply == 0) {

return basePrice;

}

int price = basePrice \* demand / supply;

return max(price, 1);

}

void TradeBetweenPlanets(vector<Route>& routes, Galaxy& galaxy, int& moneyPlayer, int& fuelPlayer, vector<shared\_ptr<Corporation>>& corporations) {

PrintText("Начало торговли между планетами с корпорациями...", 20);

for (auto& route : routes) {

string planet1 = route.planet1;

string planet2 = route.planet2;

shared\_ptr<Planet> p1 = nullptr;

shared\_ptr<Planet> p2 = nullptr;

for (auto& body : galaxy.GetCelestialBodies()) {

shared\_ptr<Planet> planet = dynamic\_pointer\_cast<Planet>(body);

if (planet && planet->GetName() == planet1) {

p1 = planet;

}

if (planet && planet->GetName() == planet2) {

p2 = planet;

}

}

if (p1 && p2) {

for (auto& corp : corporations) {

if (auto logisticsCompany = dynamic\_pointer\_cast<LogisticsCompany>(corp)) {

route.cost = max(0, route.cost - 5);

cout << "Логистическая компания оптимизировала маршрут между " << planet1 << " и " << planet2 << ". Новый расход топлива: " << route.cost << endl;

}

else if (auto techTrader = dynamic\_pointer\_cast<TechTrader>(corp)) {

for (auto it = p2->GetDemand().begin(); it != p2->GetDemand().end(); ++it) {

auto& resource = it->first;

auto& demand = it->second;

if (resource == "Сталь" || resource == "Медикаменты" || resource == "Нефть") {

auto p1ResourceIt = p1->resources.find(resource);

if (p1ResourceIt != p1->resources.end() && p1ResourceIt->second > 0) {

int supply = p1ResourceIt->second;

int pricePerUnit = CalculatePrice(resource, demand, supply);

int tradeAmount = min(demand, supply);

int totalPrice = pricePerUnit \* tradeAmount;

if (moneyPlayer >= totalPrice) {

p1ResourceIt->second -= tradeAmount;

p2->resources[resource] += tradeAmount;

moneyPlayer -= totalPrice;

cout << "TechTrader успешно продал " << tradeAmount << " единиц " << resource << " между " << planet1 << " и " << planet2 << endl;

}

}

}

}

}

else if (auto minerCompany = dynamic\_pointer\_cast<MinerCompany>(corp)) {

for (auto& body : galaxy.GetCelestialBodies()) {

shared\_ptr<Asteroid> asteroid = dynamic\_pointer\_cast<Asteroid>(body);

if (asteroid) {

if (p1) {

minerCompany->MineAsteroid(\*asteroid, moneyPlayer, \*p1);

}

else if (p2) {

minerCompany->MineAsteroid(\*asteroid, moneyPlayer, \*p2);

}

}

}

}

if (fuelPlayer >= route.cost) {

fuelPlayer -= route.cost;

}

else {

cout << "Недостаточно топлива для маршрута между " << planet1 << " и " << planet2 << "." << endl;

continue;

}

if (route.securityLevel < 20) {

cout << "Уровень безопасности маршрута между " << planet1 << " и " << planet2 << " слишком низкий. Возможны потери ресурсов." << endl;

for (auto& resource : p1->resources) {

int loss = resource.second \* (20 - route.securityLevel) / 100;

resource.second = max(0, resource.second - loss);

cout << "Потеряно " << loss << " единиц ресурса " << resource.first << " на маршруте." << endl;

}

}

}

}

PrintText("Торговля с корпорациями завершена!", 20);

}

}

void PrintGameResults(int moneyPlayer1, int moneyPlayer2) {

PrintText("ИТОГИ ИГРЫ:", 20);

cout << endl;

PrintText("Игрок 1: ", 20);

cout << "Деньги: " << moneyPlayer1 << " шекелей" << endl;

PrintText("Игрок 2: ", 20);

cout << "Деньги: " << moneyPlayer2 << " шекелей" << endl;

if (moneyPlayer1 > moneyPlayer2) {

PrintText("ПОБЕДИЛ ИГРОК 1!", 20);

}

else if (moneyPlayer1 < moneyPlayer2) {

PrintText("ПОБЕДИЛ ИГРОК 2!", 20);

}

else {

PrintText("НИЧЬЯ! Оба игрока завершили игру с одинаковым количеством топлива.", 20);

}

}

## class.h

#pragma once

#include <iostream>

#include <vector>

#include <memory>

#include <thread> // this\_thread::sleep\_for

#include <chrono> // chrono::milliseconds

#include <sstream>

#include <algorithm>

using namespace std;

#include <map>

class CelestialBody {

protected:

string name;

public:

CelestialBody(string name) {

this->name = name;

}

virtual void getInfo() {

cout << endl << "Название: " << name << endl;

}

string GetName() {

return name;

}

};

class Planet : public CelestialBody {

public:

int coefficient;

map <string, int> resources;

map <string, int> demand;

string technologyLevel;

Planet(string& name) : CelestialBody(name), technologyLevel("Неопределённый") {}

void SetTechnologyLevel() {

for (auto& res : resources) {

if (res.first == "Древесина" || res.first == "Еда") {

technologyLevel = "Недоразвитая";

break;

}

else if (res.first == "Каучук" || res.first == "Сталь" || res.first == "Железо") {

technologyLevel = "Развивающаяся";

}

else {

technologyLevel = "Развитая";

}

}

}

void SetProduct(string& name, int& count) {

resources[name] = count;

}

void SetDemand(string& name, int& count) {

demand[name] = count;

}

void GenerateResources() {

for (auto element : resources) {

element.second \*= coefficient;

}

}

void DevelopPlanet() {

for (auto& res : resources) {

res.second += 50;

}

}

void getInfo() {

cout << endl << "---------------PLANET INFORMATION---------------" << endl;

cout << "Название: " << name;

cout << endl << "Технологический уровень: " << technologyLevel;

cout << endl << "### Ресурсы ###" << endl;

for (auto& element : resources) {

cout << element.first << " : " << element.second << " единиц | ";

}

cout << endl << "### Cпрос ###" << endl;

for (auto& element : demand) {

cout << element.first << " : " << element.second << " единиц | ";

cout << endl;

}

cout << endl;

}

const map<string, int>& GetResources() {

return resources;

}

map<string, int>& GetDemand() {

return demand;

}

};

class Asteroid : CelestialBody {

private:

int priceMining;

string nameResource;

int countResource;

public:

Asteroid(string& name, int& priceMining, int& countResources, int& cost, string& nameResource) : CelestialBody(name), countResource(cost), nameResource(nameResource) {

priceMining = cost;

}

};

class Corporation {

protected:

string name;

public:

Corporation(string name) {

this->name = name;

}

virtual ~Corporation() = default;

string getName() {

return name;

}

};

class LogisticsCompany : public Corporation {

public:

LogisticsCompany(string name) : Corporation(name) {}

};

class TechTrader : public Corporation {

public:

TechTrader(string name) : Corporation(name) {}

};

class MinerCompany : public Corporation {

public:

MinerCompany(string name) : Corporation(name) {}

void MineAsteroid(Asteroid& asteroid, int& moneyPlayer, Planet& nearestPlanet) {

int miningCost = 1000;

int minedResources = 100;

vector<string> possibleResources = { "Сталь", "Железо" };

// Рандомный выбор ресурса

string resourceName = possibleResources[rand() % possibleResources.size()];

if (moneyPlayer >= miningCost) {

moneyPlayer -= miningCost;

nearestPlanet.resources[resourceName] += minedResources;

cout << "MinerCompany добыла " << minedResources << " единиц ресурса \"" << resourceName

<< "\" с астероида и доставила на планету "

<< nearestPlanet.GetName() << "." << endl;

}

else {

cout << "Недостаточно средств для добычи ресурса \"" << resourceName << "\" с астероида !!!" << endl;

}

}

};

class Galaxy {

private:

string name = "Галактика Андромеды";

vector <shared\_ptr<CelestialBody>> celectialBody; //shared ptr умный указатель объект класса

map < pair<string, string>, pair<int, int> > routes;

public:

void AddCelestialBody(shared\_ptr<CelestialBody>& object) {

celectialBody.push\_back(object);

}

void AddRoute(string& planet1, string& planet2, int cost, int securityLevel) {

routes[{planet1, planet2}] = { cost, securityLevel };

}

void getInfo() {

cout << endl << "---------------GALAXY INFORMATION---------------" << endl;

cout << "Название: " << name;

for (auto& element : celectialBody) {

element->getInfo();

}

cout << endl << "Маршруты:" << endl;

for (auto it = routes.begin(); it != routes.end(); it++) {

auto& planets = it->first;

auto& route = it->second;

cout << endl << planets.first << " <-> " << planets.second << " стоит " << route.first << " с уровнем безопасности " << route.second << endl;

}

}

const vector<shared\_ptr<CelestialBody>>& GetCelestialBodies() const {

return celectialBody;

}

};

class Route {

public:

string planet1;

string planet2;

int cost;

int securityLevel;

Route(const string& p1, const string& p2, int c, int s)

: planet1(p1), planet2(p2), cost(c), securityLevel(s) {}

void upgradeRoute() {

cost -= 50;

}

};

## header.h

#pragma once

#include <iostream>

#include <vector>

#include <memory>

#include <thread> // this\_thread::sleep\_for

#include <chrono> // chrono::milliseconds

#include <sstream>

#include <algorithm>

using namespace std;

#include <map>

#include "class.h"

void PrintText(const string& text, int milliseconds);

void CreatePlanets(Galaxy& galaxy, const string& playerName);

vector<string> ChooseResources(Galaxy& galaxy, int currentPlayer);

void CreateRoutes(Galaxy& galaxy, vector<Route>& routes);

void ChooseCorporations(vector<shared\_ptr<Corporation>>& corporationsPlayer, Galaxy& galaxy);

void MakeInvestment(int& money, vector<Route>& routes, Galaxy& galaxy);

int CalculatePrice(const string& resource, int demand, int supply);

void TradeBetweenPlanets(vector<Route>& routes, Galaxy& galaxy, int& moneyPlayer, int& fuelPlayer, vector<shared\_ptr<Corporation>>& corporations);

void PrintGameResults(int moneyPlayer1, int moneyPlayer2);

# Результат работы программы

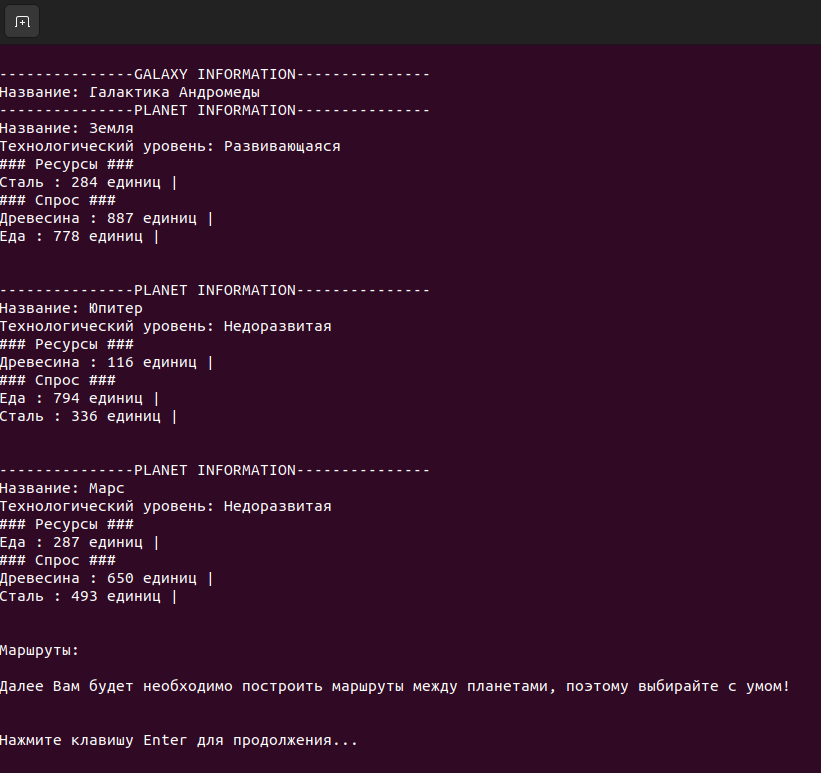


Рисунок 2. Пример работы программы.

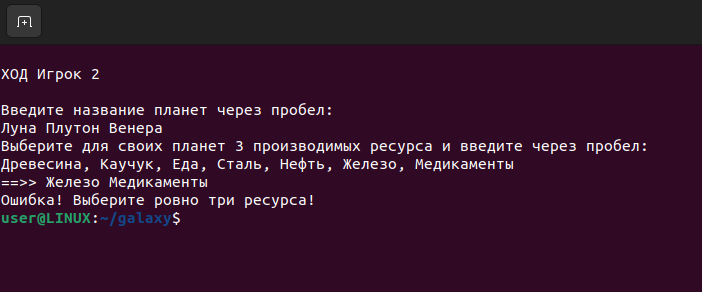


Рисунок 3. Пример работы программы с неправильным вводом.

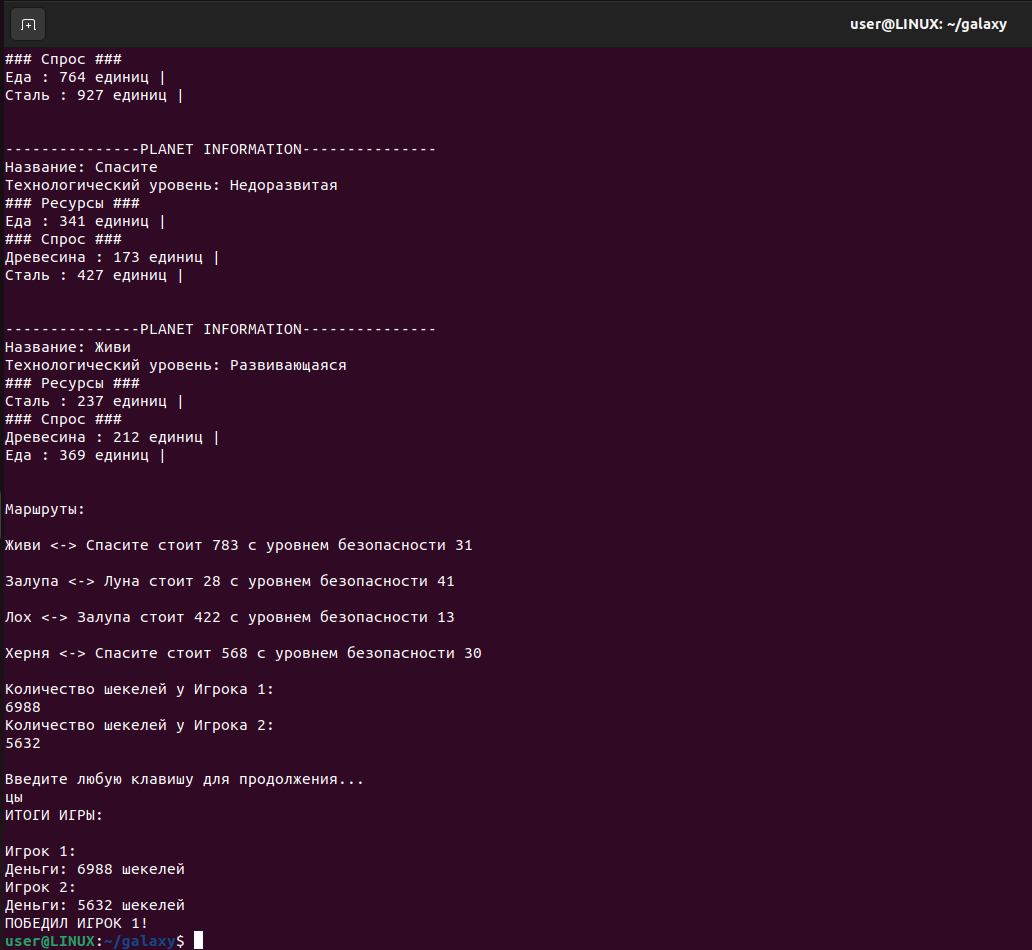


Рисунок 4. Пример работы программы.

# Вывод

*Используемые технологии ООП:*

**Наследование:**

Применяется для создания иерархий классов. Например, Planet, Asteroid, LogisticsCompany, TechTrader, MinerCompany наследуют от соответствующих родительских классов: CelestialBody и Corporation.

**Полиморфизм:**

Использование виртуальных методов в родительских классах (CelestialBody, Corporation) и переопределение их в подклассах (например, getInfo() в Planet и Asteroid).

**Абстракция:**

Создание абстрактных классов (Corporation, CelestialBody), которые задают общие интерфейсы для специализированных классов.

*Используемые паттерны проектирования:*

**Фабрика:**

Использование умных указателей (shared\_ptr) на объекты в классе Galaxy позволяет динамически создавать объекты классов, таких как Planet, Asteroid, и добавлять их в коллекцию.

**Стратегия (Strategy):**

Классы корпораций (LogisticsCompany, TechTrader, MinerCompany) представляют собой разные стратегии для выполнения определенных бизнес-операций.

***Достоинства:***

* **Гибкость и расширяемость:** Использование наследования и полиморфизма позволяет легко расширять систему, добавляя новые типы планет, корпораций или ресурсов, не изменяя существующий код.
* **Простота в поддержке:** Классы хорошо организованы, что облегчает их поддержку и улучшение. Добавление новых классов не требует изменения существующих.
* **Использование умных указателей (shared\_ptr):** Это снижает вероятность утечек памяти, так как память автоматически очищается, когда объект больше не используется.

***Недостатки:***

* **Сложность для новичков:** Из-за использования умных указателей, наследования и полиморфизма, код может быть сложным для восприятия новичками в ООП.
* **Избыточность в структуре:** Иногда использование множества классов, наследования и ассоциаций может приводить к избыточной сложности, особенно если проект растет. В некоторых случаях это может усложнить поддержку и развитие программы.

# Ссылки на репозитории

Github: <https://github.com/mamam111a/laboratory-5>