Министерство науки и высшего образования Российской Федерации

Федеральное государственное образовательное учреждение

высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с отметкой\_\_\_\_\_

Преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

« \_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г.

Отчет

по лабораторной работе №10

**«Исключения»**

по дисциплине «Программирование - 3 семестр»

Студент группы ПИ-02 Немчинов В. И.

Преподаватель доцент, к.т.н. Троцкий В.

Барнаул 2022 г.

**Задание:**

В проектах на C++, C# и Java

1) Придумать разумное использование в ваших проектах защищенных блоков (try-блоки), перехватов исключений (catch-блоки) и инициализации исключений (инструкция throw);

2) Продемонстрировать все это;

!) Git обязателен;

!) В отчете должна быть ссылка на github.

***Репозиторий:*** https://github.com/Zireael-of-Hope/Lab10-11

Код программы:

**C++**

Lab9\_C++.cpp:

#include <iostream>

#include <Droid.h>

#include <conio.h>

#include <Color.h>

int main()

{

int droid\_quantity = 4;

Droid\* OOM = new Droid[droid\_quantity];

OOM[0].SetPower(true);

OOM[0].AssignPersonalObjective(66);

OOM[0].PersonalOrderExecution();

Droid::AssignGeneralObjective(77);

Droid::GeneralOrderExecution(OOM, true, 0, 2);

Droid::GetStatus(OOM);

OOM[0].AddSpeciallTool("Bazooka");

OOM[0].AddSpeciallTool("Blaster");

OOM[1].AddSpeciallTool("Bazooka");

OOM[2].AddSpeciallTool("Blaster");

OOM[1].SetPower(true);

OOM[1].AssignPersonalObjective(66);

Droid::GeneralOrderExecution(OOM, false, 0, 2);

Droid::AddDroid(OOM, droid\_quantity);

Droid::GetStatus(OOM);

delete[] OOM;

try {

if (Droid::GetCount() != 0)

throw exception("Old droid squad isn't deleted yet!");

}

catch(const exception& ex) {

cout << ex.what();

}

int droid\_quantity\_i = 4;

const int droid\_quantity\_j = 5;

//Для двумерного массивы не доступен метод добавдения нового дроида

Droid\*\* OOP = new Droid \* [droid\_quantity\_i];

for (int i = 0; i < droid\_quantity\_i; i++) {

OOP[i] = new Droid[droid\_quantity\_j];

}

OOP[0][0].AddSpeciallTool("Bazooka");

OOP[0][0].AddSpeciallTool("Blaster");

OOP[1][0].AddSpeciallTool("Bazooka");

OOP[2][0].AddSpeciallTool("Blaster");

OOP[0][1].SetPower(true);

OOP[0][2].SetPower(true);

OOP[1][1].SetPower(true);

OOP[2][1].SetPower(true);

OOP[3][1].SetPower(true);

OOP[3][3].SetPower(true);

Droid::GetStatus(OOP, droid\_quantity\_i, droid\_quantity\_j);

}

Droid.h:

#pragma once

#include <iostream>

#include <string>

#include <Color.h>

using namespace std;

class Droid

{

private:

static int count;

static int general\_order;

int personal\_order;

int executing\_order;

string id;

bool isOnline;

bool isGeneralOrderExecuting;

string\* special\_tools;

int quantity\_tools;

public:

Droid();

/\*Droid(bool isOnline);

Droid(bool isOnline, int personal\_order, int executing\_order, bool isGeneralOrderExecuting);

Droid(const Droid& other);\*/

~Droid();

static void GetStatus(Droid OOM[]);

static void GetStatus(Droid\* OOM[], int droid\_quantity\_i, const int droid\_quantity\_j);

static int GetCount();

static bool AssignGeneralObjective(int order\_number);

static bool GeneralOrderExecution(Droid OOM[], bool isExecuting, int fromIndex, int toIndex);

static int GetAssignedHGeneralObjective();

bool AssignPersonalObjective(int order\_number);

bool PersonalOrderExecution();

int GetAssignedPersonalObjective();

string GetID();

void SetPower(bool isPowerON);

bool GetPowerStatus();

void AddSpeciallTool(string new\_tool);

bool DeleteSpeciallTool(string deleting\_tool\_name);

void GetSpecialTools();

static void AddDroid(Droid\*& OOM, int& droid\_quantity);

Droid& operator= (const Droid& other);

};

Droid.cpp:

#include "Droid.h"

int Droid::count = 0;

int Droid::general\_order = 0;

Droid::Droid()

{

id = "OOM-" + to\_string(count + 4577);

count++;

isOnline = false;

personal\_order = 0;

executing\_order = 0;

isGeneralOrderExecuting = false;

quantity\_tools = 0;

special\_tools = nullptr;

}

//Droid::Droid(bool isOnline)

//{

// id = "OOM-" + to\_string(count + 4577);

// count++;

// this->isOnline = isOnline;

// personal\_order = 0;

// executing\_order = 0;

// isGeneralOrderExecuting = false;

//}

//

//Droid::Droid(bool isOnline, int personal\_order, int executing\_order, bool isGeneralOrderExecuting)

//{

// id = "OOM-" + to\_string(count + 4577);

// count++;

// this->isOnline = isOnline;

// this->personal\_order = personal\_order;

// this->executing\_order = executing\_order;

// this->isGeneralOrderExecuting = isGeneralOrderExecuting;

//}

//

//Droid::Droid(const Droid& other)

//{

// id = other.id;

// count++;

// isOnline = other.isOnline;

// personal\_order = other.personal\_order;

// executing\_order = other.executing\_order;

// isGeneralOrderExecuting = other.isGeneralOrderExecuting;

// quantity\_tools = other.quantity\_tools;

// if (other.special\_tools == nullptr)

// special\_tools = nullptr;

// else {

// special\_tools = new string[quantity\_tools];

// for (int i = 0; i < quantity\_tools; i++)

// special\_tools[i] = other.special\_tools[i];

// }

//}

Droid::~Droid()

{

if (special\_tools != nullptr)

delete[] special\_tools;

count--;

}

void Droid::GetStatus(Droid OOM[])

{

cout << "General Order: " << general\_order << "\n" << endl;

for (int i = 0; i < Droid::count; i++) {

if (OOM[i].isOnline)

SetColor(2, 0);

else SetColor(4, 0);

cout << "Droid " << OOM[i].id << " status: ";

if (OOM[i].isOnline)

cout << "Online ";

else

cout << "Offline";

cout << " | Executing order: " << OOM[i].executing\_order;

if (OOM[i].executing\_order == 0)

cout << " ";

cout << " | Personal Order: " << OOM[i].personal\_order;

if (OOM[i].personal\_order == 0)

cout << " ";

cout << " Special Tools: ";

if (OOM[i].quantity\_tools == 0)

cout << "none";

else

for (int j = 0; j < OOM[i].quantity\_tools; j++) {

cout << OOM[i].special\_tools[j] << " ";

}

cout << endl;

}

SetColor(15, 0);

}

void Droid::GetStatus(Droid\* OOM[], int droid\_quantity\_i, const int droid\_quantity\_j)

{

cout << "General Order: " << general\_order << "\n" << endl;

for (int i = 0; i < droid\_quantity\_i; i++) {

SetColor(15, 0);

cout << "Squad: " << i+1 << endl;

for (int j = 0; j < droid\_quantity\_j; j++) {

if (OOM[i][j].isOnline)

SetColor(2, 0);

else SetColor(4, 0);

cout << " Droid " << OOM[i][j].id << " status: ";

if (OOM[i][j].isOnline)

cout << "Online ";

else

cout << "Offline";

cout << " | Executing order: " << OOM[i][j].executing\_order;

if (OOM[i][j].executing\_order == 0)

cout << " ";

cout << " | Personal Order: " << OOM[i][j].personal\_order;

if (OOM[i][j].personal\_order == 0)

cout << " ";

cout << " Special Tools: ";

if (OOM[i][j].quantity\_tools == 0)

cout << "none";

else

for (int k = 0; k < OOM[i][j].quantity\_tools; k++) {

cout << OOM[i][j].special\_tools[k] << " ";

}

cout << endl;

}

}

SetColor(15, 0);

}

int Droid::GetCount()

{

return count;

}

bool Droid::AssignGeneralObjective(int order\_number)

{

if (order\_number != 66 && order\_number != 77 && order\_number != 0)

return false;

else {

Droid::general\_order = order\_number;

return true;

}

}

bool Droid::GeneralOrderExecution(Droid OOM[], bool isExecuting, int fromIndex, int toIndex)

{

if (fromIndex < 0 || toIndex > count || fromIndex > count || toIndex < 0) {

return false;

}

else {

if (isExecuting)

for (int i = fromIndex; i <= toIndex; i++)

OOM[i].executing\_order = Droid::general\_order;

else

for (int i = fromIndex; i <= toIndex; i++)

OOM[i].executing\_order = OOM[i].personal\_order;

return true;

}

}

int Droid::GetAssignedHGeneralObjective()

{

return general\_order;

}

bool Droid::AssignPersonalObjective(int order\_number)

{

if (order\_number != 66 && order\_number != 77 && order\_number != 0)

return false;

else {

personal\_order = order\_number;

return true;

}

}

string Droid::GetID()

{

return id;

}

void Droid::SetPower(bool isPowerON)

{

isOnline = isPowerON;

}

bool Droid::GetPowerStatus()

{

return isOnline;

}

void Droid::AddSpeciallTool(string new\_tool)

{

int i;

string\* new\_special\_tools = new string[quantity\_tools + 1];

for (i = 0; i < quantity\_tools; i++)

new\_special\_tools[i] = special\_tools[i];

new\_special\_tools[quantity\_tools] = new\_tool;

delete[] special\_tools;

special\_tools = new\_special\_tools;

quantity\_tools++;

}

bool Droid::DeleteSpeciallTool(string deleting\_tool\_name)

{

int i;

bool is\_tool\_found = false;

string\* new\_special\_tools;

for (i = 0; i < quantity\_tools; i++) {

if (special\_tools[i] == deleting\_tool\_name) {

for (i; i < quantity\_tools - 1; i++)

special\_tools[i] = special\_tools[i + 1];

new\_special\_tools = new string[quantity\_tools - 1];

for (i = 0; i < quantity\_tools - 1; i++)

new\_special\_tools[quantity\_tools - 1] = special\_tools[i];

delete[] special\_tools;

special\_tools = new\_special\_tools;

quantity\_tools--;

return true;

break;

}

}

return false;

}

void Droid::GetSpecialTools()

{

if (quantity\_tools != 0) {

cout << "Droid's special tools: " << endl;

for (int i = 0; i < quantity\_tools; i++) {

cout << " " << special\_tools[i] << endl;

}

}

else {

cout << "Droid doest have any tools." << endl;

}

}

void Droid::AddDroid(Droid\*& OOM, int& droid\_quantity)

{

int i;

Droid\* new\_OOM = new Droid[droid\_quantity + 1];

new\_OOM[droid\_quantity].id = new\_OOM[0].id;

for (i = 0; i < droid\_quantity; i++)

new\_OOM[i] = OOM[i];

delete[] OOM;

OOM = new\_OOM;

droid\_quantity++;

}

Droid& Droid::operator=(const Droid& other)

{

id = other.id;

isOnline = other.isOnline;

personal\_order = other.personal\_order;

executing\_order = other.executing\_order;

isGeneralOrderExecuting = other.isGeneralOrderExecuting;

quantity\_tools = other.quantity\_tools;

if (other.special\_tools == nullptr)

special\_tools = nullptr;

else {

special\_tools = new string[quantity\_tools];

for (int i = 0; i < quantity\_tools; i++)

special\_tools[i] = other.special\_tools[i];

}

return \*this;

}

bool Droid::PersonalOrderExecution()

{

if (isGeneralOrderExecuting)

return false;

else {

executing\_order = personal\_order;

return true;

}

}

int Droid::GetAssignedPersonalObjective()

{

return personal\_order;

}

**Java:**

**Main.Java:**

package com.company;

public class Main {

public static void main(String[] args) {

int droid\_quantity = 4;

Droid[] OOM = new Droid[droid\_quantity];

for (int i = 0; i < droid\_quantity; i++) {

OOM[i] = new Droid();

}

OOM[0].SetPower(true);

OOM[0].AssignPersonalObjective(66);

OOM[0].PersonalOrderExecution();

Droid.AssignGeneralObjective(77);

Droid.GeneralOrderExecution(OOM, true, 0, 2);

Droid.GetStatus(OOM);

OOM[1].SetPower(true);

OOM[1].AssignPersonalObjective(66);

Droid.GeneralOrderExecution(OOM, false, 0, 2);

Droid.GetStatus(OOM);

Droid.ClearCount();

OOM = null;

try {

if (Droid.GetCount() != 0)

throw new DroidsException("Old droid squad isn't deleted yet!");

}

catch(DroidsException ex) {

System.out.println(ex.what());

}

int droid\_quantity\_i = 2;

int droid\_quantity\_j = 3;

Droid[][] OOP = new Droid[droid\_quantity\_i][droid\_quantity\_j];

for (int i = 0; i < droid\_quantity\_i; i++) {

for (int j = 0; j < droid\_quantity\_j; j++) {

OOP[i][j] = new Droid();

}

}

OOP[0][0].SetPower(true);

OOP[0][1].SetPower(true);

OOP[0][2].SetPower(true);

OOP[1][0].SetPower(true);

OOP[1][1].SetPower(true);

Droid.GetStatus(OOP, droid\_quantity\_i, droid\_quantity\_j);

}

}

Droid.Java:

package com.company;

public class Droid {

private static int count;

private static int general\_order;

private int personal\_order;

private int executing\_order;

private int id;

private boolean isOnline;

private boolean isGeneralOrderExecuting;

public Droid() {

id = count + 4577;

count++;

isOnline = false;

personal\_order = 0;

executing\_order = 0;

isGeneralOrderExecuting = false;

}

public static void GetStatus(Droid[] OOM) {

System.out.println("General Order: " + general\_order + "\n");

for (int i = 0; i < Droid.count; i++) {

System.out.print("Droid " + OOM[i].id + " status: ");

if (OOM[i].isOnline)

System.out.print("Online ");

else

System.out.print("Offline");

System.out.print(" | Executing order: " + OOM[i].executing\_order);

if (OOM[i].executing\_order == 0)

System.out.print(" ");

System.out.println(" | Personal Order: " + OOM[i].personal\_order);

}

}

public static void GetStatus(Droid[][] OOM, int droid\_quantity\_i, int droid\_quantity\_j) {

System.out.println("General Order: " + general\_order + "\n");

for (int i = 0; i < droid\_quantity\_i; i++) {

System.out.println("Squad: " + (i + 1));

for (int j = 0; j < droid\_quantity\_j; j++) {

System.out.print("Droid " + OOM[i][j].id + " status: ");

if (OOM[i][j].isOnline)

System.out.print("Online ");

else

System.out.print("Offline");

System.out.print(" | Executing order: " + OOM[i][j].executing\_order);

if (OOM[i][j].executing\_order == 0)

System.out.print(" ");

System.out.println(" | Personal Order: " + OOM[i][j].personal\_order);

}

}

}

public static int GetCount() {

return count;

}

public static void ClearCount() {

count = 0;

}

public static boolean AssignGeneralObjective(int order\_number) {

if (order\_number != 66 && order\_number != 77 && order\_number != 0)

return false;

else {

Droid.general\_order = order\_number;

return true;

}

}

public static boolean GeneralOrderExecution(Droid[] OOM, boolean isExecuting, int fromIndex, int toIndex) {

if (fromIndex < 0 || toIndex > count || fromIndex > count || toIndex < 0) {

return false;

}

else {

if (isExecuting)

for (int i = fromIndex; i <= toIndex; i++)

OOM[i].executing\_order = Droid.general\_order;

else

for (int i = fromIndex; i <= toIndex; i++)

OOM[i].executing\_order = OOM[i].personal\_order;

return true;

}

}

public static int GetAssignedHGeneralObjective() {

return general\_order;

}

public boolean AssignPersonalObjective(int order\_number) {

if (order\_number != 66 && order\_number != 77 && order\_number != 0)

return false;

else {

personal\_order = order\_number;

return true;

}

}

public boolean PersonalOrderExecution() {

if (isGeneralOrderExecuting)

return false;

else {

executing\_order = personal\_order;

return true;

}

}

public int GetAssignedPersonalObjective() {

return personal\_order;

}

public int GetID() {

return id;

}

public void SetPower(boolean isPowerON) {

isOnline = isPowerON;

}

public boolean GetPowerStatus() {

return isOnline;

}

}

DroidsException.Java:

package com.company;

public class DroidsException extends Exception {

private String warning;

DroidsException(String warning\_message) {

warning = warning\_message;

}

public String what() {

return warning;

}

}

**C#:**

**Program.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lab1011\_CS

{

public class Program

{

public static void Main(string[] args)

{

int droid\_quantity = 4;

Droid[] OOM = new Droid[droid\_quantity];

for (int i = 0; i < droid\_quantity; i++)

{

OOM[i] = new Droid();

}

OOM[0].SetPower(true);

OOM[0].AssignPersonalObjective(66);

OOM[0].PersonalOrderExecution();

Droid.AssignGeneralObjective(77);

Droid.GeneralOrderExecution(OOM, true, 0, 2);

Droid.GetStatus(OOM);

OOM[1].SetPower(true);

OOM[1].AssignPersonalObjective(66);

Droid.GeneralOrderExecution(OOM, false, 0, 2);

Droid.GetStatus(OOM);

Droid.ClearCount();

OOM = null;

try

{

if (Droid.GetCount() != 0)

throw new Exception("Old droid squad isn't deleted yet!");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

int droid\_quantity\_i = 2;

int droid\_quantity\_j = 3;

Droid[,] OOP = new Droid[droid\_quantity\_i,droid\_quantity\_j];

for (int i = 0; i < droid\_quantity\_i; i++)

{

for (int j = 0; j < droid\_quantity\_j; j++)

{

OOP[i,j] = new Droid();

}

}

OOP[0,0].SetPower(true);

OOP[0,1].SetPower(true);

OOP[0,2].SetPower(true);

OOP[1,0].SetPower(true);

OOP[1,1].SetPower(true);

Droid.GetStatus(OOP, droid\_quantity\_i, droid\_quantity\_j);

}

}

}

Droid.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1011\_CS

{

internal class Droid

{

private static int count;

private static int general\_order;

private int personal\_order;

private int executing\_order;

private int id;

private bool isOnline;

private bool isGeneralOrderExecuting;

public Droid()

{

id = count + 4577;

count++;

isOnline = false;

personal\_order = 0;

executing\_order = 0;

isGeneralOrderExecuting = false;

}

public static void GetStatus(Droid[] OOM)

{

Console.WriteLine("General Order: " + general\_order + "\n");

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

{

Console.Write("Droid " + OOM[i].id + " status: ");

if (OOM[i].isOnline)

Console.Write("Online ");

else

Console.Write("Offline");

Console.Write(" | Executing order: " + OOM[i].executing\_order);

if (OOM[i].executing\_order == 0)

Console.Write(" ");

Console.WriteLine(" | Personal Order: " + OOM[i].personal\_order);

}

}

public static void GetStatus(Droid[,] OOM, int droid\_quantity\_i, int droid\_quantity\_j)

{

Console.WriteLine("General Order: " + general\_order + "\n");

for (int i = 0; i < droid\_quantity\_i; i++)

{

Console.WriteLine("Squad: " + (i + 1));

for (int j = 0; j < droid\_quantity\_j; j++)

{

Console.Write("Droid " + OOM[i,j].id + " status: ");

if (OOM[i,j].isOnline)

Console.Write("Online ");

else

Console.Write("Offline");

Console.Write(" | Executing order: " + OOM[i,j].executing\_order);

if (OOM[i,j].executing\_order == 0)

Console.Write(" ");

Console.WriteLine(" | Personal Order: " + OOM[i,j].personal\_order);

}

}

}

public static int GetCount()

{

return count;

}

public static void ClearCount()

{

count = 0;

}

public static bool AssignGeneralObjective(int order\_number)

{

if (order\_number != 66 && order\_number != 77 && order\_number != 0)

return false;

else

{

Droid.general\_order = order\_number;

return true;

}

}

public static bool GeneralOrderExecution(Droid[] OOM, bool isExecuting, int fromIndex, int toIndex)

{

if (fromIndex < 0 || toIndex > count || fromIndex > count || toIndex < 0)

{

return false;

}

else

{

if (isExecuting)

for (int i = fromIndex; i <= toIndex; i++)

OOM[i].executing\_order = Droid.general\_order;

else

for (int i = fromIndex; i <= toIndex; i++)

OOM[i].executing\_order = OOM[i].personal\_order;

return true;

}

}

public static int GetAssignedHGeneralObjective()

{

return general\_order;

}

public bool AssignPersonalObjective(int order\_number)

{

if (order\_number != 66 && order\_number != 77 && order\_number != 0)

return false;

else

{

personal\_order = order\_number;

return true;

}

}

public bool PersonalOrderExecution()

{

if (isGeneralOrderExecuting)

return false;

else

{

executing\_order = personal\_order;

return true;

}

}

public int GetAssignedPersonalObjective()

{

return personal\_order;

}

public int GetID()

{

return id;

}

public void SetPower(bool isPowerON)

{

isOnline = isPowerON;

}

public bool GetPowerStatus()

{

return isOnline;

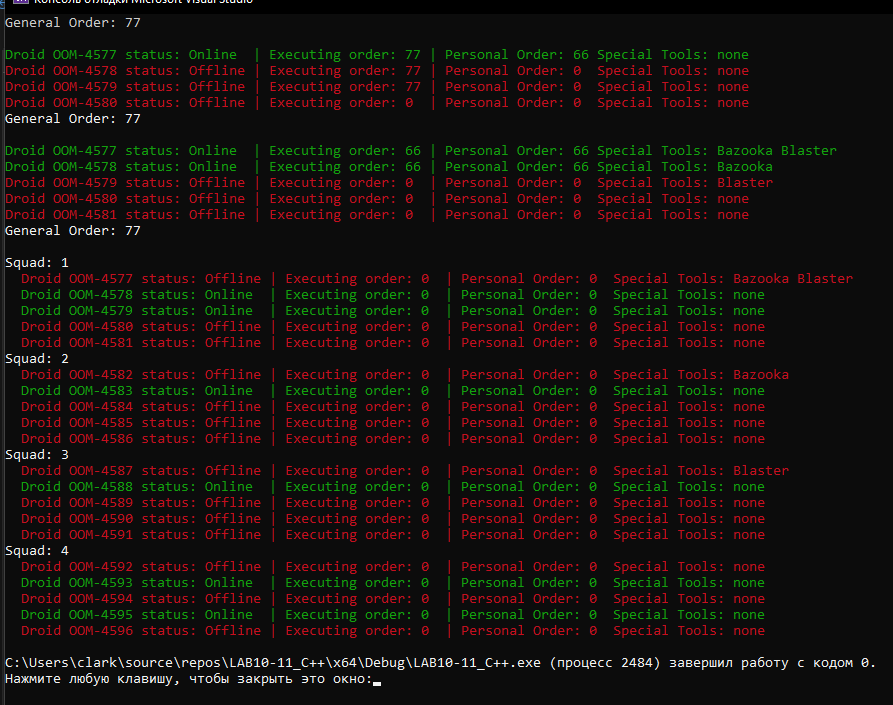
}

}

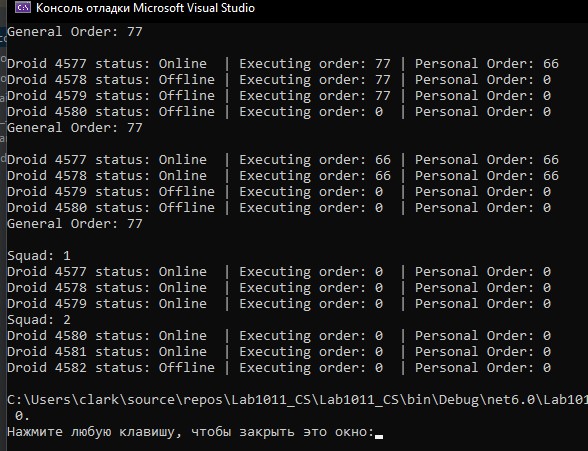
}

Результаты работы программы:

C++



C#



Java

