***Нечаев Евгений***

**Задание**:

Есть коллекция объектов, набор типов которых может расширяться.

Клиент получает эту коллекцию и выдает другую по правилу:

A-1 A-5

B-3  B-5

C-5 C-5

**My Collection**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**main.cpp**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#include "mycollection.hpp"

int main()

{

int const NUMTYPES = 3;

int const RANGE = 100;

Collection<Prototype> MyCollection(NUMTYPES, RANGE);

MyCollection.CreateCollection(new A(), new B(), new C());

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

cout << "count = " << MyCollection.collection[i].size() << endl;

Client clt;

clt.EqualizeCollection(MyCollection);

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

cout << "count = " << MyCollection.collection[i].size() << endl;

return 0;

}

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**mycollection.hpp**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#ifndef MYCOLLECTION\_HPP

#define MYCOLLECTION\_HPP

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <vector>

using namespace std;

template<typename T>

struct Collection

{

int numTypes;

int countTypes;

int rangeObj;

vector<T\*>\* collection;

public:

Collection(int \_numTypes = 0, int \_rangeObj = 0)

: numTypes(\_numTypes)

, rangeObj(\_rangeObj)

, countTypes(0)

{

srand(time(0));

collection = new vector<T\*>[numTypes];

}

template<typename O, typename... TYPES>

void CreateCollection(O\* obj, TYPES\*... objTypes)

{

if (countTypes >= numTypes) return;

int num = rand() % rangeObj;

cout << "random number = " << num << endl;

collection[countTypes].resize(num);

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

{

collection[countTypes].at(i) = obj->Clone();

}

++countTypes;

CreateCollection(objTypes...);

}

template<typename TO>

void CreateCollection(TO\* obj)

{

int num = rand() % rangeObj;

cout << "random number = " << num << endl;

collection[countTypes].resize(num);

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

{

collection[countTypes].at(i) = obj->Clone();

}

++countTypes;

}

};

class Prototype

{

public:

virtual ~*Prototype*() {}

virtual Prototype\* *Clone*() const = 0;

};

class A : public Prototype

{

public:

A() {}

A(A const&) {}

A\* *Clone*() const { return new A(\*this); }

};

class B : public Prototype

{

public:

B() {}

B(B const&) {}

B\* *Clone*() const { return new B(\*this); }

};

class C : public Prototype

{

public:

C() {}

C(C const&) {}

C\* *Clone*() const { return new C(\*this); }

};

class Client

{

public:

void EqualizeCollection(Collection<Prototype>& );

};

void Client::EqualizeCollection(Collection<Prototype>& cltn)

{

int max = cltn.collection[0].size();

for (int i = 1; i < cltn.numTypes; ++i)

{

if (max <= cltn.collection[i].size())

max = cltn.collection[i].size();

}

cout << "max number = " << max << endl;

for (int i = 0; i < cltn.numTypes; ++i)

{

int lastIdx = cltn.collection[i].size();

while (lastIdx < max)

{

cltn.collection[i].push\_back(cltn.collection[i].at(lastIdx - 1)->Clone());

lastIdx = cltn.collection[i].size();

}

}

}

#endif // MYCOLLECTION\_HPP

