Практична робота №6

Тема: Розробка і використання шаблонних класів.

Мета: Закріпити навички створення і використання шаблонних класів.

Хід роботи

1. У Git-репозиторію із попередніх практичних робіт створюю нову гілку «PR6» і переходжу в неї для виконання даної практичної роботи.

2. Створюю шаблонний клас Vector, який являтиметься контейнером даних, або динамічним масивом.

3. Оголошую та реалізовую для даного класу необхідні методи.

4. Пишу функцію main, в якій перевірите роботу шаблонного класу-контейнера з різними вбудованими типами даних.

5. Перевіряю роботу шаблонного класу-контейнера, помістивши в нього вказівника на класи, розроблені в попередніх практичних роботах.

6. Роблю коміт проєкту із повідомленням «done practical work №6».

7. Зливаю гілку PR6 у гілку main (merge) і надсилаю зміни у гілці main у віддалений репозиторій (git push).

8. Оформлюю звіт.

ПРОГРАМНИЙ КОД

CustomVector.h:

```
#pragma once

template<typename T>
class CustomVector
{
private:
T *arr;
int size = 0;
int capacity = 1;
public:
int getSize();
int getCapacity();
void push_back(T element);
void pop_back();
T at(int index);
T operator [] (int index);

CustomVector();
    ~CustomVector();
};

template<typename T>
CustomVector<T>::CustomVector() : size(0), capacity(10)
{
    arr = new T[this->capacity];
```

```
}

template<typename T>
CustomVector<T>::~CustomVector()
{
    delete this->arr;
}


template<typename T>
int CustomVector<T>::getSize()
{
    return this->size;
}

template<typename T>
int CustomVector<T>::getCapacity()
{
    return this->capacity;
}

template<typename T>
void CustomVector<T>::push_back(const T element)
{
    if (size >= capacity) {
        capacity++;
        T* temporaryArr = new T[this->capacity];
        for (int x = 0; x < size; x++) {
            temporaryArr[x] = this->arr[x];
        }
        delete[] arr;
        this->arr = temporaryArr;
        delete[] temporaryArr;
    }
    arr[size] = element;
    size++;
}

template<typename T>
void CustomVector<T>::pop_back()
{
    T* temporaryArr = new T[capacity];
    for (int x = 0; x < size - 1; x++) {
        temporaryArr[x] = arr[x];
    }
    delete[] arr;
    this->arr = temporaryArr;
    delete[] temporaryArr;

    size--;
}

template<typename T>
T CustomVector<T>::at(int index)
{
    return this->arr[index];
}

template<typename T>
T CustomVector<T>::operator[](int index)
{
```

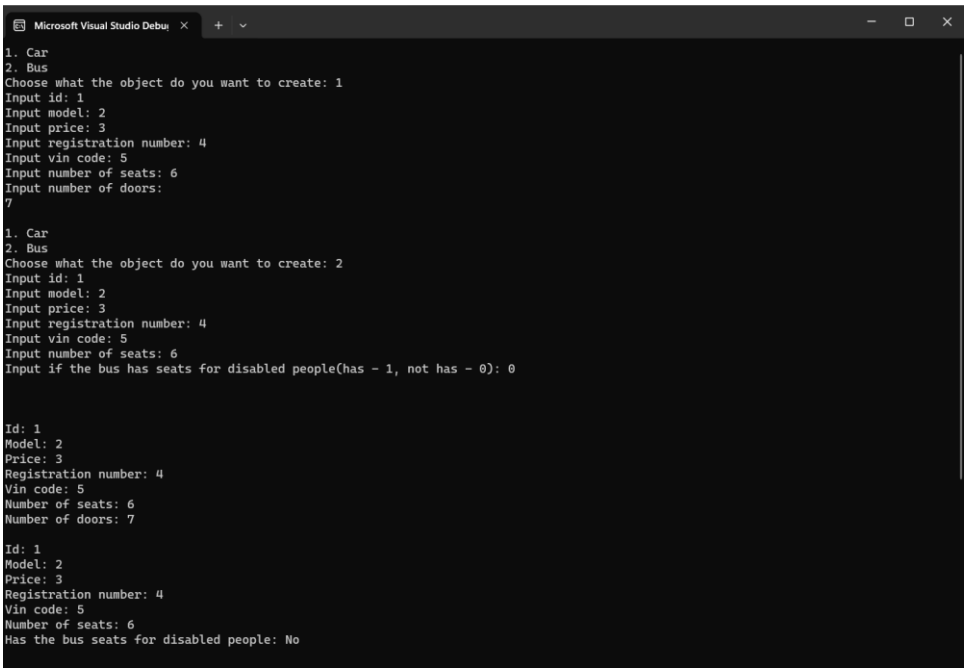
```
    return at(index);
}
```

Main.cpp:

```
#include <iostream>
#include "Car.h"
#include "Bus.h"
#include "Vehicle.h"
#include "CustomVector.h"
#include <vector>
#include <algorithm>
#include <list>
#include <map>

using namespace std;

int randInt(int minInclusiveValue, int maxInclusiveValue) {
    return (minInclusiveValue + (rand() % maxInclusiveValue - minInclusiveValue + 1));
}

int randIntOdd(int minInclusiveValue, int maxInclusiveValue) {
    int randOdd = 0;

    do {
        randOdd = randInt(minInclusiveValue, maxInclusiveValue);
    } while (randOdd % 2 == 0);

    return randOdd;
}

int randIntEven(int minInclusiveValue, int maxInclusiveValue) {
    int randOdd = 0;

    do {
        randOdd = randInt(minInclusiveValue, maxInclusiveValue);
    } while (randOdd % 2 != 0);

    return randOdd;
}

int main()
{
    srand(time(nullptr));

#pragma region lab5
    Vehicle* vehicles[2];
    int choice;

    for (short x = 0; x < 2; x++) {
        cout << "1. Car\n2. Bus\nChoose what the object do you want to create: "; cin >>
choice;

        if (choice == 1) {
            vehicles[x] = new Car;
            vehicles[x]->input();
        }
        else {
            vehicles[x] = new Bus;
```

```
            vehicles[x]->input();
        }

        cout << endl;
    }

    for (int x = 0; x < 2; x++) {
        vehicles[x]->beep();
    }

    cout << endl << endl;

#pragma endregion (done)

#pragma region lab6

    CustomVector<Vehicle*> list;

    for (int x = 0; x < 2; x++) {
        list.push_back(vehicles[x]);
    }

    for (int x = 0; x < 2; x++) {
        list[x]->output();
        cout << endl;
    }

    for (int x = 0; x < list.getSize(); x++) {
        list[x]->output();
        cout << endl;
    }
}
```

Скріншот виконання програми:



Висновок: під час виконання даної практичної роботи, закріплено навички створення і використання шаблонних класів.