МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ

ТЕХНОЛОГИЙ И ТЕЛЕКОММУНИКАЦИЙ

КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

**Лабораторная работа №7**

по дисциплине

«Наследование»

**Выполнил:**

Ермолаев Кирилл Александрович

Студент 2 курса группы \_ПИН-б-о-22-1

Направления подготовки

09.03.03 Прикладная информатика

очной формы обучения

Ставрополь, 2023 г.

**Тема:** Основы объектно-ориентированного программирования на ЯП С++.

**Цель работы:** Наследование.

**Ход работы:**

**Вариант 7** SymbString DecString operator +(T&, T&)

Написать программу, демонстрирующую работу с объектами двух типов, T1 и T2, для чего создать систему соответствующих классов. Каждый объект должен иметь идентификатор (в виде произвольной строки символов) и одно или несколько полей для хранения состояния (текущего значения) объекта.

Клиенту (функции main) должны быть доступны следующие основные операции (методы): создать объект, удалить объект, показать значение объекта и прочие дополнительные операции (зависят от варианта). Операции по созданию и удалению объектов инкапсулировать в классе Factory. Предусмотреть меню, позволяющее продемонстрировать заданные операции. При необходимости в разрабатываемые классы добавляются дополнительные методы (например, конструктор копирования, операция присваивания и т.п.) для обеспечения надлежащего функционирования этих классов.

// Factory.h :

#ifndef FACTORY\_H

#define FACTORY\_H

#include <string>

template <typename T>

class Factory

{

public:

    static T \*create(const std::string &id);

    static void destroy(T \*obj);

};

#endif // FACTORY\_H

// Factory.cpp :

#include "Factory.h"

template <typename T>

T \*Factory<T>::create(const std::string &id)

{

    return new T(id);

}

template <typename T>

void Factory<T>::destroy(T \*obj)

{

    delete obj;

}

// T1.h:

#ifndef T1\_H

#define T1\_H

#include <string>

template <typename T>

class T1

{

public:

    std::string identifier;

    T1(const std::string &id);

};

#endif // T1\_H

// T1.cpp:

#include "T1.h"

template <typename T>

T1<T>::T1(const std::string &id) : identifier(id) {}

// T2.h

#ifndef T2\_H

#define T2\_H

#include <string>

template <typename T, typename U>

class T2

{

public:

    std::string identifier;

    U value;

    T2(const std::string &id, U val);

};

#endif // T2\_H

// T2.cpp:

#include "T2.h"

template <typename T, typename U>

T2<T, U>::T2(const std::string &id, U val) : identifier(id), value(val) {}

// DecString.h

#ifndef DECSTRING\_H

#define DECSTRING\_H

#include <string>

template <typename T, typename U>

class DecString

{

public:

    std::string identifier;

    U value;

    DecString(const std::string &id, U val);

    void showValue() const;

};

#endif // DECSTRING\_H

// DecString.cpp

#include "DecString.h"

#include <iostream>

template <typename T, typename U>

DecString<T, U>::DecString(const std::string &id, U val) : identifier(id), value(val) {}

template <typename T, typename U>

void DecString<T, U>::showValue() const

{

    std::cout << "DecString " << identifier << " has value: " << value << std::endl;

}

// SymbString.h

#ifndef SYMBSTRING\_H

#define SYMBSTRING\_H

#include <string>

template <typename T>

class SymbString

{

public:

    T identifier;

    SymbString(const T &id);

    void showIdentifier() const;

};

#endif // SYMBSTRING\_H

// SymbString.cpp

#include "SymbString.h"

#include <iostream>

template <typename T>

SymbString<T>::SymbString(const T &id) : identifier(id) {}

template <typename T>

void SymbString<T>::showIdentifier() const

{

    std::cout << "SymbString identifier: " << identifier << std::endl;

}

// main.cpp:

#include "Factory.h"

#include "T1.h"

#include "T2.h"

#include "DecString.h"

#include "SymbString.h"

int main()

{

    using T1Type = T1<std::string>;

    using T2Type = T2<std::string, int>;

    using DecStringType = DecString<std::string, int>;

    using SymbStringType = SymbString<std::string>;

    T1Type \*t1 = Factory<T1Type>::create("T1\_Object");

    T2Type \*t2 = Factory<T2Type>::create("T2\_Object", 10);

    DecStringType \*decString = Factory<DecStringType>::create("DecString\_Object", 25);

    SymbStringType \*symbString = Factory<SymbStringType>::create("SymbString\_Object");

    // Демонстрация операции создания объектов

    std::cout << "Created objects with identifiers: " << t1->identifier << ", " << t2->identifier << ", "

              << decString->identifier << ", " << symbString->identifier << std::endl;

    // Демонстрация операции сложения

    T2 result = (\*t1) + (\*t2);

    std::cout << "Result of T1 + T2: " << result.identifier << " with value: " << result.value << std::endl;

    // Демонстрация методов DecString и SymbString

    decString->showValue();

    symbString->showIdentifier();

    Factory<T1Type>::destroy(t1);

    Factory<T2Type>::destroy(t2);

    Factory<DecStringType>::destroy(decString);

    Factory<SymbStringType>::destroy(symbString);

    return 0;

}

<https://github.com/MoeTomatoki/OOP>

**Вывод:** изучил базовые понятия в использование паттернов проектирования; технология проектирования программ с учетом будущих изменений. А также более подробно изучил тему наследования классов и полиморфизм.