Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

по лабораторной работе № 3

«Множественное наследование.

Дружественные классы и функции.»

Выполнил

ст. гр. ИТб22

Серегин А.В.

Проверил:

Тимофеев И.С.

Севастополь

2015

1. Цель работы

Приобретение практических навыков при написании объектно-ориентированных программ. Освоение особенностей отладки объектно-ориентированных программ. Изучение механизма множественного наследования. Приобретение навыков работы с механизмом «дружественности».

2. Варианты заданий

Описать интерфейс ввода-вывода. Описать иерархию классов, заданную по варианту. Для каждого класса описать конструкторы и деструктор (по необходимости), функции ввода и вывода значений полей. В каждом классе должны присутствовать минимум одно уникальное поле и один уникальный метод. Проиллюстрировать корректную работу механизма множественного наследования. Класс-наследник дол-жен реализовывать описанный ранее интерфейс. Описать дружественную функцию для класса-наследника и заданного по варианту класса.

Вариант 5

Базовые классы: Вода (объем), Вещество (вес, название, тип). Класс-наследник: Раствор.

Дружественность: класс Сосуд.

1. Текст программы

Bulb.h:

#pragma once

#ifndef \_BULB\_

#define \_BULB\_

#include <iostream>

#include <iomanip>

#include "Liquor.h"

class Bulb

{

private:

std::string title\_;

std::string liquorTitle\_;

Liquor liquor\_;

public:

Bulb(std::string \_title);

void addLiquor(Liquor& liquor);

Liquor getLiquor();

void printInfo() const;

};

#endif

Liquor.h:

#pragma once

#ifndef \_LIQUOR\_

#define \_LIQUOR\_

#include <string>

#include "Water.h"

#include "Substance.h"

class Liquor : public Water, public Substance

{

private:

std::string title\_;

public:

friend class Bulb;

Liquor();

Liquor(const std::string& \_liquorTitle, Water& \_water, Substance& \_substance);

void addLiquor(Liquor& \_liquor);

Liquor getLiquor();

std::string getLiquorTitle() const;

};

#endif

Substance.h:

#pragma once

#ifndef \_SUBSTANCE\_

#define \_SUBSTANCE\_

#include <string>

class Substance

{

private:

double mass\_;

std::string title\_;

std::string type\_;

public:

friend class Bulb;

Substance();

Substance(const std::string& \_title,

const std::string \_type);

Substance(const double& \_mass, const std::string& \_title,

const std::string \_type);

Substance(Substance& sub);

void addSubstance(Substance& sub);

void addSubstance(Substance& sub, const double& \_mass);

Substance getSubstance(double mass = -1.0);

std::string getSubstanceTitle() const;

std::string getSubstanceType() const;

double getSubstanceMass() const;

};

#endif

Water.h:

#pragma once

#ifndef \_WATER\_

#define \_WATER\_

#include <cstdlib>

class Water

{

private:

double capacity\_;

public:

friend class Bulb;

Water();

Water(Water& water);

Water(const double& \_capacity);

void addWater(Water& \_water);

void addWater(Water& \_water, const double& \_capacity);

Water getWater(double capacity = -1.0);

double getCapacity() const;

};

#endif

Bulb.cpp:

#include "Bulb.h"

Bulb::Bulb(std::string \_title):liquor\_("None",Water(0.0),Substance("None","None"))

{

title\_ = \_title;

liquorTitle\_ = "None";

}

void Bulb::addLiquor(Liquor& liquor)

{

liquorTitle\_ = liquor.title\_;

liquor\_ = liquor;

}

Liquor Bulb::getLiquor()

{

if (liquorTitle\_ == "None")

abort();

Liquor temp = liquor\_;

liquor\_ = Liquor("None", Water(0.0), Substance("None", "None"));

liquorTitle\_ = "None";

return temp;

}

void Bulb::printInfo() const

{

if (liquor\_.getLiquorTitle() == "None")

{

std::cout << title\_ << " is empty." << std::endl;

return;

}

std::cout.setf(std::ios::left);

std::cout << std::setw(20) << "Luquor title: " << liquorTitle\_ << std::endl;

std::cout << std::setw(20) << "Water capacity: " << liquor\_.Water::capacity\_ << std::endl;

std::cout << std::setw(20) << "Substance title: " << liquor\_.Substance::title\_ << std::endl;

std::cout << std::setw(20) << "Substance type: " << liquor\_.Substance::type\_ << std::endl;

std::cout << std::setw(20) << "Substance mass: " << liquor\_.Substance::mass\_ << std::endl;

}

Liquor.cpp:

#include "Liquor.h"

Liquor::Liquor()

{

title\_ = "None";

}

Liquor::Liquor(const std::string& \_liquorTitle, Water& \_water,

Substance& \_substance):Water(\_water),

Substance(\_substance)

{

title\_ = \_liquorTitle;

}

void Liquor::addLiquor(Liquor& \_liquor)

{

if (\_liquor.title\_ != title\_)

abort();

this->addWater(\_liquor.getWater());

this->addSubstance(\_liquor.getSubstance());

}

Liquor Liquor::getLiquor()

{

if (title\_ == "None")

abort();

Liquor temp = (\*this);

this->getWater();

this->getSubstance();

return temp;

}

std::string Liquor::getLiquorTitle() const

{

return title\_;

}

Substance.cpp:

#include "Substance.h"

Substance::Substance()

{

mass\_ = 0.0;

title\_ = "None";

type\_ = "None";

}

Substance::Substance(const std::string & \_title, const std::string \_type)

{

mass\_ = 0.0;

title\_ = \_title;

type\_ = \_type;

}

Substance::Substance(const double & \_mass, const std::string & \_title, const std::string \_type)

{

mass\_ = \_mass;

title\_ = \_title;

type\_ = \_type;

}

Substance::Substance(Substance & sub)

{

title\_ = sub.title\_;

type\_ = sub.type\_;

mass\_ = sub.mass\_;

sub = Substance();

}

void Substance::addSubstance(Substance& sub)

{

if (title\_ != "None" && type\_ != "None")

{

if (title\_ != sub.title\_ || type\_ != sub.type\_)

abort();

}

else

{

title\_ = sub.title\_;

type\_ = sub.type\_;

}

mass\_ += sub.mass\_;

sub.mass\_ = 0.0;

}

void Substance::addSubstance(Substance & sub, const double & \_mass)

{

if (title\_ != "None" && type\_ != "None")

{

if (title\_ != sub.title\_ || type\_ != sub.type\_ || \_mass > sub.mass\_)

abort();

}

else

{

title\_ = sub.title\_;

type\_ = sub.type\_;

}

mass\_ += \_mass;

sub.mass\_ -= \_mass;

}

Substance Substance::getSubstance(double mass)

{

if (mass\_ < mass || type\_ == "None" || title\_ == "None")

abort();

if (mass < 0)

mass = mass\_;

Substance temp = \*this;

temp.mass\_ = mass;

mass\_ -= mass;

return temp;

}

inline std::string Substance::getSubstanceTitle() const

{

return title\_;

}

inline std::string Substance::getSubstanceType() const

{

return type\_;

}

inline double Substance::getSubstanceMass() const

{

return mass\_;

}

Water.cpp:

#include "Water.h"

Water::Water()

{

capacity\_ = 0;

}

Water::Water(Water& water)

{

capacity\_ = water.capacity\_;

water.capacity\_ = 0;

}

Water::Water(const double& \_capacity)

{

capacity\_ = \_capacity;

}

void Water::addWater(Water& \_water)

{

capacity\_ += \_water.capacity\_;

\_water.capacity\_ = 0.0;

}

void Water::addWater(Water& \_water, const double& \_capacity)

{

if (\_capacity < \_water.capacity\_)

abort();

capacity\_ += \_water.capacity\_;

\_water.capacity\_ -= 0.0;

}

Water Water::getWater(double capacity)

{

if (capacity > capacity\_)

abort();

if (capacity < 0)

capacity = capacity\_;

capacity\_ -= capacity;

return Water(capacity);

}

inline double Water::getCapacity() const

{

return capacity\_;

}

main.cpp:

#include <iostream>

#include "Bulb.h"

void printInfo(const Bulb& \_first, const Bulb& \_second)

{

std::cout << std::endl << "First bottle:" << std::endl;

\_first.printInfo();

std::cout << std::endl << "Second bottle:" << std::endl;

\_second.printInfo();

}

int main()

{

Bulb bottle1("Bottle");

Bulb bottle2("Bottle");

printInfo(bottle1, bottle2);

bottle1.addLiquor(Liquor("Salt water", Water(0.3), Substance(0.01, "Salt", "Salt")));

std::cout << std::endl << "Salt water added in the first bottle." << std::endl;

printInfo(bottle1, bottle2);

std::cout << std::endl << "Salt water trancefused into the second bottle of the first." << std::endl;

bottle2.addLiquor(bottle1.getLiquor());

printInfo(bottle1, bottle2);

system("pause");

return 0;

}

1. Тестовые примеры.

На рисунке ниже представлено выполнение выше описанной программы.

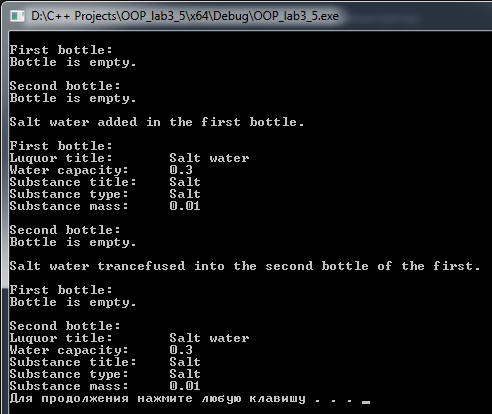


Рисунок 1 – результаты выполнения программы.

Выводы

В ходе выполнения лабораторной работы был изучен механизм множественного наследования, а так же его основные проблемы. Также были изучены дружественные функции.