Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Об’єктно – орієнтоване програмування

ЗВІТ ДО

ЛАБОРАТОРНОЇ РОБОТИ №4

ВАРІАНТ №7

Оцінка «\_\_\_\_\_\_\_»

Дата «\_\_» \_\_\_\_\_\_\_ 2017р

Виконав студент ІІ курсу

Гр. ТІ-62

Заїчко Олексій Павлович

Перевірив:\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2017

***Склад програми:***

main.cpp

#include "stdafx.h"

#include "Windows.h"

#include "string.h"

#include "Operation.h"

#include "DailyBalance.h";

int main()

{

Operation first(1.2, "Fedko", "Varenik", 18, 10, 2017, DIZEL, 1.1),

second( 1.2, "Vasia", "Galushka", 19, 10, 2017, AI95, 1.6);

first.print("/");

printf("%s %s %s\n", second.getName(), second.getSurname(), second.toString());

Operation mass[] = { first, second };

DailyBalance today(mass, 2, 19, 10, 2017);

DailyBalance test(today);

test.printOperations();

system("pause");

return 0;

}

Date.h

#pragma once

class Date {

protected:

int year;

short month;

short day;

static int monthsDays[13];

public:

Date(int day = 1, int month = 1, int year = 1980);

Date(const Date& p);

void setDate(int dd = 1, int mm = 1, int yy = 1980);

char\* getDateStr(char\* s) const;

virtual void print(char\* s) const = 0;

virtual void Show() const;

};

Date.cpp

#include "stdafx.h"

#include "Date.h"

int Date::monthsDays[13] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };

Date::Date(int day, int month, int year) {

setDate(day, month, year);

}

Date::Date(const Date& p) {

day = p.day;

month = p.month;

year = p.year;

}

void Date::setDate(int dd, int mm, int yy) {

month = (mm >= 1 && mm <= 12) ? mm : 1;

year = (yy >= 1980 && yy <= 3000) ? yy : 1980;

if (month == 2 && year % 4 == 0 && (year % 400 == 0 || year % 100 != 0)) {

day = (dd >= 1 && dd <= 29) ? dd : 1;

}

else {

day = (dd >= 1 && dd <= monthsDays[month]) ? dd : 1;

}

}

char\* Date::getDateStr(char\* s) const {

char\* res = new char[11];

sprintf(res, "%d%s%d%s%d", day, s, month, s, year);

return res;

}

void Date::Show() const {

printf("I'm a Date Object!\n");

}

Operator.h

#pragma once

#include "Date.h"

#include "stdafx.h"

#include "string.h"

class Operator : public Date {

protected:

char\* name;

char\* surname;

public:

Operator(char\* n = "Name", char\* s = "Surname", int dd = 1, int mm = 1, int yy = 1980);

Operator(const Operator& p);

~Operator();

void setName(char\* s);

void setSurname(char\* s);

void checkAndSetName(char\* s);

void checkAndSetSurname(char\* s);

const char\* getName() const;

const char\* getSurname() const;

virtual void print(char\* s) const;

virtual void Show() const;

};

Operator.cpp

#include "Operator.h"

Operator::Operator(char\* n, char\* s, int dd, int mm, int yy)

: Date(dd, mm, yy) {

if (strlen(n) <= 20) {

name = new char[strlen(n) + 1];

strcpy(name, n);

}

else {

name = new char[5];

strcpy(name, "Name");

}

if (strlen(s) <= 20) {

surname = new char[strlen(s) + 1];

strcpy(surname, s);

}

else {

surname = new char[8];

strcpy(name, "Surname");

}

}

Operator::Operator(const Operator& p) :

Date (p) {

name = new char[strlen(p.name) + 1];

strcpy(name, p.name);

surname = new char[strlen(p.surname) + 1];

strcpy(surname, p.surname);

}

Operator::~Operator() {

delete[] name;

delete[] surname;

}

void Operator::setName(char\* s) {

name = new char[strlen(s) + 1];

strcpy(name, s);

}

void Operator::setSurname(char\* s) {

surname = new char[strlen(s) + 1];

strcpy(surname, s);

}

void Operator::checkAndSetName(char\* s) {

if (strlen(s) <= 20) {

setName(s);

}

}

void Operator::checkAndSetSurname(char\* s) {

if (strlen(s) <= 20) {

setSurname(s);

}

}

const char\* Operator::getName() const {

return name;

}

const char\* Operator::getSurname() const {

return surname;

}

void Operator::print(char\* s) const {

printf("Operators birthday is: %d%s%d%s%d\n", day, s, month, s, year );

}

void Operator::Show() const {

printf("I'm operator Object!");

}

Fuel.h

#pragma once

enum FuelType { AI92 = 1, AI95, AI98, DIZEL };

class Fuel {

protected:

FuelType fuelType;

double price;

public:

Fuel(FuelType type = AI95, double p = 0.0);

void setType(FuelType);

FuelType getType() const;

void setPrice(double p);

void setAndCheckPrice(double p);

double getPrice() const;

const char\* toString() const;

};

Fuel.cpp

#pragma once

enum FuelType { AI92 = 1, AI95, AI98, DIZEL };

class Fuel {

protected:

FuelType fuelType;

double price;

public:

Fuel(FuelType type = AI95, double p = 0.0);

void setType(FuelType);

FuelType getType() const;

void setPrice(double p);

void setAndCheckPrice(double p);

double getPrice() const;

const char\* toString() const;

};

Operation.h

#pragma once

#include "Fuel.h"

#include "Operator.h"

class Operation : public Operator, public Fuel {

protected:

double amount;

double totalPrice;

void setTotalPrice();

public:

Operation(double sAmount = 0, char\* name = "Name", char\* surname = "Surname", int dd = 1,

int mm = 1, int yy = 1980, FuelType type = AI95, double price = 0.0) :

Operator(name, surname, dd, mm, yy), Fuel(type, price) {

amount = sAmount;

setTotalPrice();

}

Operation(const Operation& p);

void setAmount(double p);

void checkAndSetAmount(double p);

Operation& operator= (const Operation& p);

double getAmount() const;

double getTotalPrice() const;

virtual void print(char\* s) const;

virtual void Show() const;

};

Operation.cpp

#include "Operation.h"

Operation::Operation(const Operation& p) :

Operator(p), Fuel(p) {

totalPrice = p.totalPrice;

amount = p.amount;

}

Operation& Operation::operator= (const Operation& p) {

if (this == &p) return \*this;

setName(p.name);

setSurname(p.surname);

amount = p.amount;

day = p.day;

month = p.month;

year = p.year;

fuelType = p.fuelType;

price = p.price;

setTotalPrice();

return \*this;

}

void Operation::setTotalPrice() {

totalPrice = amount \* price;

}

void Operation::setAmount(double p) {

amount = p;

setTotalPrice();

}

void Operation::checkAndSetAmount(double p) {

amount = (p > 0)? p : 0.0;

}

double Operation::getAmount() const {

return amount;

}

double Operation::getTotalPrice() const {

return totalPrice;

}

void Operation::print(char\* s) const {

printf("Operation date: %d%s%d%s%d\n", day, s, month, s, year);

}

void Operation::Show() const {

printf("I'm operation Object!");

}

DailyBalance.h

#include "Operation.h"

class DailyBalance : public Date {

private:

Operation\* operationList;

int size;

void printTableDivider(int sym) const;

public:

DailyBalance(Operation\*, int size, int dd = 1, int mm = 1, int yy = 1980);

DailyBalance(const DailyBalance& p);

~DailyBalance();

void printOperations() const;

virtual void print(char\* s) const;

virtual void Show() const;

};

DailyBalance.cpp

#include "DailyBalance.h"

DailyBalance::DailyBalance(Operation\* mass, int s, int dd, int mm, int yy) {

setDate(dd, mm, yy);

operationList = new Operation[s];

size = s;

for (int i = 0; i < s; i++) {

operationList[i] = mass[i];

}

}

DailyBalance::DailyBalance(const DailyBalance& p) : Date(p) {

operationList = new Operation[p.size];

size = p.size;

for (int i = 0; i < p.size; i++) {

operationList[i] =p.operationList[i];

}

}

DailyBalance::~DailyBalance() {

delete[] operationList;

}

void DailyBalance::printOperations() const {

print(":");

printTableDivider(194);

printf("%c%-20s%c%-20s%c%-10s%c%-5s%c%-5s%c%-10s%c%-5s%c\n",179, "Name", 179,

"Surname", 179, "Date", 179, "Price", 179, "Amount", 179,"Total",179, "Type", 179);

for (int i = 0; i < size; i++) {

printTableDivider(197);

printf("%c%-20s%c%-20s%c%-10s%c%5.2lf%c%6.2lf%c%10.2lf%c%-5s%c\n",

179, (operationList[i]).getName(),

179, (operationList[i]).getSurname(),

179, (operationList[i]).getDateStr("/"),

179, (operationList[i]).getPrice(),

179, (operationList[i]).getAmount(),

179, (operationList[i]).getTotalPrice(),

179, (operationList[i]).toString(),

179);

}

printTableDivider(193);

}

void DailyBalance::print(char\* s) const {

printf("Daily Balance for %d%s%d%s%d\n", day, s, month, s, year);

}

void DailyBalance::printTableDivider(int sym) const {

int sideFirst;

int sideSecond;

switch (sym) {

case 194:

sideFirst = 218;

sideSecond = 191;

break;

case 197:

sideFirst = 195;

sideSecond = 180;

break;

case 193:

sideFirst = 192;

sideSecond = 217;

break;

default: break;

}

putchar(sideFirst);

for (int i = 0; i < 82; i++) {

if (i == 20 || i == 41 || i == 52 || i == 58 || i == 65 || i == 76) {

putchar(sym);

}

else {

putchar(196);

}

}

putchar(sideSecond);

putchar('\n');

}

void DailyBalance::Show() const {

printf("I'm daily balance Object!");

}

Клас *Date* – абстрактний базовий клас. Функція *print() –* чиста віртуальна функція.

Клас *Fuel –* базовий клас.

Клас *Operator* – похідний клас, який відкрито наслідує клас *Date.*

*Клас Operation* – похідний клас, який відкрито наслідує *Operator* і *Fuel.*

Клас *DailyBalance* – похідний клас, відкрито наслідує *Data.* В якості поля має масив об’єктів класу *Operation.*

Відповіді на контрольні питання:

1. *Шаблон функції –* узагальнений опис функції, визначення якої здійснюється за допомогою термінів узагальнених типів. Терміни узагальнених типів при побудові узагальненої функції замінюються конкретними типами, що передаються у ф-цію як параметри. Сам по собі шаблон не викликає генерації будь-якого програмного коду.

2. *Реалізація шаблонних класів –* Класи реалізовуються за допомогою визначеного об’єкту, що використовує шаблонний аргумент. При створенні об’єкту компілятор резервує область даних для цього об’єкту, скрізь де з’являються аргументи параметру типу цього класу. Резервується місце під методи, якщо вона не була зарезервована іншим об’єктом цього типу.

3. *Модифікатор protected* – визначає елементи класу, як доступні для будь-якого похідного класу, в той же час, недосяжний поза класами цієї ієрархії.