Experiment #4

1.

Order:

1. Date .h/.cpp
2. Employee .h/.cpp
3. SalariedEmployee .h/.cpp
4. HourlyEmployee .h/.cpp
5. CommissionEmployee .h/.cpp
6. BasePlusCommissionEmployee .h/.cpp
7. Main .cpp

//Date.h

#ifndef Date\_h

#define Date\_h

class Date {

friend ostream& operator<<(ostream&, const Date&);

public:

Date(int m = 1, int d = 1, int y = 1900);//constructor

void setDate(int, int, int);//set month, day, year

static bool leapYear(int);//is date in a leap year?

bool endOfMonth(int) const;//is the date at the end of month?

bool sameMonth(const Date&);//month\_1 == month\_2 ?

private:

unsigned int month, day, year;

static const array<unsigned int, 13>days;//days per month

void helpIncrement();//utility function for incrementing date

};

#endif // !Date\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Date.cpp

#include<iostream>

#include<string>

#include<array>

using namespace std;

#include"Date.h"

const array<unsigned int, 13>Date::days =

{ 0,31,28,31,30,31,30,31,31,30,31,30,31 };

//initialize static member; one classwide copy

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

setDate(month, day, year);

}//constructor

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

if (mm > 01 && mm <= 12)

month = mm;

else

throw invalid\_argument("Month must be 1-12");

if (yy >= 1900 && yy <= 2100)

year = yy;

else

throw invalid\_argument("Year must be >= 1900 and <= 2100");

//test for a leap year

if ((month == 2 && leapYear(year) && dd >= 1 && dd <= 29) ||

(dd >= 1 && dd <= days[month]))

day = dd;

else

throw invalid\_argument(

"Day is out of range for current month and year");

}//seat month, day, and year

bool Date::leapYear(int testYear) {

if (testYear % 400 == 0 ||

(testYear % 100 != 0 && testYear % 4 == 0))

return true;//a leap year

else

return false;//not a leap year

}//if the year is a leap year, return true;otherwise, return false

bool Date::endOfMonth(int testDay) const{

if (month == 2 && leapYear(year))

return testDay == 29;

else

return testDay == days[month];

}//determinr whether the day is the last day of the month

void Date::helpIncrement() {

//day is not end of month

if (!endOfMonth(day))

++day;//increment day

else

if (month < 12) {//day is end of month and month < 12

++month;//increment month

day = 1;//first day of new month

}

else {//last day of year

++year;//increment year

month = 1;//first month of new year

day = 1;//first day of new month

}

}//function to help increment th date

ostream& operator<<(ostream& output, const Date& d) {

static string monthName[13] = { "","January","Febuary",

"March","April","May","June","July","August",

"Sepetember","October","November","December" };

output << monthName[d.month] << ' ' << d.day << ", " << d.year;

return output;//enables cascading

}//overload output operator

bool Date::sameDay(const Date&d){

if (this->day == d.day)

return 1;

else

return 0;

}//return if they have the same day

bool Date::sameMonth(const Date& d) {

if (this->month == d.month)

return 1;

else

return 0;

}//return if they have the same month

////////////////////////////////////////////////////////////////////////////////////////////////////

//Employee.h

#ifndef Employee\_h

#define Employee\_h

#include"Date.h"

class Employee {

public:

Employee(const string&, const string&, const string&, const Date&);//constructor

virtual~Employee(){}//virtual destructor

void setFirstName(const string&);//set first name

string getFirstName() const;//return first name

void setLastName(const string&);//set last name

string getLastName() const;//return last name

void setSocialSecurityNumber(const string&);//set SSN

string getSocialSecurityNumber() const;//return SSN

virtual double earnings() const = 0;//pure virtual

virtual void print() const;//virtual

bool Birthday() const;//is this month the employee's birthday in?

private:

string firstName, lastName, socialSecurityNumber;

Date birthDate;

};

#endif // !Employee\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Employee.cpp

#include<iostream>

#include<string>

#include<Windows.h>

using namespace std;

#include"Employee.h"

Employee::Employee(const string& first, const string& last, const string& ssn, const Date& date)

:firstName(first), lastName(last), socialSecurityNumber(ssn), birthDate(date) {

//empty body

}//constructor

void Employee::setFirstName(const string& first) {

firstName = first;

}//set first name

string Employee::getFirstName() const {

return firstName;

}//return first name

void Employee::setLastName(const string& last) {

lastName = last;

}//set last name

string Employee::getLastName() const {

return lastName;

}//return last name

void Employee::setSocialSecurityNumber(const string& ssn) {

socialSecurityNumber = ssn;//should valid

}//set social security number

string Employee::getSocialSecurityNumber() const {

return socialSecurityNumber;

}//return first name

void Employee::print() const {

cout << getFirstName() << ' ' << getLastName()

<< "\nbirthday: " << birthDate

<< "\nsocial security number: " << getSocialSecurityNumber();

}//print Employee's information (virtual, but not pure virtual)

bool Employee::Birthday() const {

SYSTEMTIME systm;

GetLocalTime(&systm);

Date today(systm.wMonth);

if (today.sameMonth(birthDate)) {

cout << "\nHAPPY BIRTHDAY!";

return 1;

}

else

return 0;

}

////////////////////////////////////////////////////////////////////////////////////////////////////

//SalariedEmployee.h

#ifndef SalariedEmployee\_h

#define SalariedEmployee\_h

#include"Employee.h"

class SalariedEmployee :public Employee {

public:

SalariedEmployee(const string&, const string&,

const string&, const Date&, double = 0.0);//constructor

virtual ~SalariedEmployee();//virtual destructor

void setWeeklySalary(double);//set weekly salary

double getWeeklySalary() const;//return weekly salary

virtual double earnings() const override;//cauculate earnings

virtual void print() const override;//print object

private:

double weeklySalary;//salary per week

};

#endif // !SalariedEmployee\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//SalariedEmployee.cpp

#include<iostream>

#include<string>

using namespace std;

#include"SalariedEmployee.h"

SalariedEmployee::SalariedEmployee(

const string& first, const string& last, const string& ssn,

const Date& date, double salary)

:Employee(first, last, ssn, date) {

setWeeklySalary(salary);

}//constructor

SalariedEmployee::~SalariedEmployee() {

cout << "deleting object of class SalariedEmployee\n";

}//destructor

void SalariedEmployee::setWeeklySalary(double salary) {

if (salary >= 0)

weeklySalary = salary;

else

throw invalid\_argument("Weekly salary must be >= 0.0");

}//set salary

double SalariedEmployee::getWeeklySalary() const {

return weeklySalary;

}//return salary

double SalariedEmployee::earnings() const {

return getWeeklySalary();

}//calculate earnings

void SalariedEmployee::print() const {

cout << "salaried employee: ";

Employee::print();//reuse abstract base-class print function

cout << "\nweekly salary: " << getWeeklySalary();

}//print SalariedEmployee's information

////////////////////////////////////////////////////////////////////////////////////////////////////

//HourlyEmployee.h

#ifndef HourlyEmployee\_h

#define HourlyEmployee\_h

#include"Employee.h"

class HourlyEmployee :public Employee {

public:

HourlyEmployee(const string&, const string&,

const string&, const Date&, double = 0.0, double = 0.0);//constructor

virtual ~HourlyEmployee();//virtual destructor

void setHourlyWage(double);//set hourly wage

double getHourlyWage() const;//return hourly wage

void setWorkTime(double);//set hours worked

double getWorkTime() const;//return hours worked

virtual double earnings() const override;//calculate earniongs

virtual void print() const override;//print object

private:

double hourlyWage, workTime;//hourly wage and hours worked

};

#endif // !HourlyEmployee

////////////////////////////////////////////////////////////////////////////////////////////////////

//HourlyEmployee.cpp

#include<iostream>

#include<string>

using namespace std;

#include"HourlyEmployee.h"

HourlyEmployee::HourlyEmployee(

const string& first, const string& last, const string& ssn,

const Date& date, double wage, double hours)

:Employee(first, last, ssn, date) {

setHourlyWage(wage);

setWorkTime(hours);

}//constructor

HourlyEmployee::~HourlyEmployee() {

cout << "deleting object of class HourlyEmployee\n";

}//destructor

void HourlyEmployee::setHourlyWage(double wage) {

if (wage >= 0)

hourlyWage = wage;

else

throw invalid\_argument("Hourly wage must be >= 0.0");

}//set hourly wage

double HourlyEmployee::getHourlyWage() const {

return hourlyWage;

}//return hourly wage

void HourlyEmployee::setWorkTime(double hours) {

if (hours >= 0)

workTime = hours;

else

throw invalid\_argument("Hours employee worked must be >= 0.0");

}//set hours worked

double HourlyEmployee::getWorkTime() const {

return workTime;

}//return hours worked

double HourlyEmployee::earnings() const {

return getHourlyWage() \* getWorkTime();

}//calculate earnings; override pure virtual function earnings in Employee

void HourlyEmployee::print() const {

cout << "hourly employee: ";

Employee::print();//code reuse

cout << "\nhourly wage: " << getHourlyWage()

<< "; hours worked: " << getWorkTime();

}//print HourlyEmployee's information

////////////////////////////////////////////////////////////////////////////////////////////////////

//CommissionEmployee.h

#ifndef CommissionEmployee\_h

#define CommissionEmployee\_h

#include"Employee.h"

class CommissionEmployee :public Employee {

public:

CommissionEmployee(const string&, const string&,

const string&, const Date&, double = 0.0, double = 0.0);//constructor

virtual ~CommissionEmployee();//virtual destructor

void setCommissioRate(double);//set commission rate

double getCommissionRate() const;//return commission rate

void setGrossSales(double);//set gross sales amount

double getGrossSales() const;//return gross sales amount

virtual double earnings() const override;//calculate earniongs

virtual void print() const override;//print object

private:

double grossSales, commissionRate;//gross weekly sales and commission percentage

};

#endif // !CommissionEmployee

////////////////////////////////////////////////////////////////////////////////////////////////////

//CommissionEmployee.cpp

#include<iostream>

#include<string>

using namespace std;

#include"CommissionEmployee.h"

CommissionEmployee::CommissionEmployee(

const string& first, const string& last, const string& ssn,

const Date& date, double sales, double rate)

:Employee(first, last, ssn, date) {

setGrossSales(sales);

setCommissioRate(rate);

}//constructor

CommissionEmployee::~CommissionEmployee() {

cout << "deleting object of class CommissionEmployee\n";

}//destructor

void CommissionEmployee::setGrossSales(double sales) {

if (sales >= 0)

grossSales = sales;

else

throw invalid\_argument("Gross sales must be >= 0.0");

}//set gross sales amount

double CommissionEmployee::getGrossSales() const {

return grossSales;

}//return gross sales

void CommissionEmployee::setCommissioRate(double rate) {

if (rate > 0.0 && rate < 1.0)

commissionRate = rate;

else

throw invalid\_argument("Commission rate must be > 0.0 and <1.0");

}//set commission rate

double CommissionEmployee::getCommissionRate() const {

return commissionRate;

}//return commission rate

double CommissionEmployee::earnings() const {

return getCommissionRate() \* getGrossSales();

}//calculate earnings; override pure virtual function earnings in Employee

void CommissionEmployee::print() const {

cout << "commission employee: ";

Employee::print();//code reuse

cout << "\ngross sales: " << getGrossSales()

<< "; commission rate: " << getCommissionRate();

}//print CommissionEmployee's information

////////////////////////////////////////////////////////////////////////////////////////////////////

//BasePlusCommissionEmployee.h

#ifndef BasePlusCommissionEmployee\_h

#define BasePlusCommissionEmployee\_h

#include"CommissionEmployee.h"

class BasePlusCommissionEmployee :public CommissionEmployee {

public:

BasePlusCommissionEmployee(const string&, const string&,

const string&, const Date&, double = 0.0, double = 0.0, double = 0.0);//constructor

virtual ~BasePlusCommissionEmployee();//virtual destructor

void setBaseSalary(double);//set base salary

double getBaseSalary() const;//return base salary

virtual double earnings() const override;//calculate earniongs

virtual void print() const override;//print object

private:

double baseSalary;//base salary per week

};

#endif // !BasePlusCommissionEmployee

////////////////////////////////////////////////////////////////////////////////////////////////////

//BasePlusCommissionEmployee

#include<iostream>

#include<string>

using namespace std;

#include"BasePlusCommissionEmployee.h"

BasePlusCommissionEmployee::BasePlusCommissionEmployee(

const string& first, const string& last, const string& ssn,

const Date& date, double sales, double rate, double salary)

:CommissionEmployee(first, last, ssn, date, sales, rate) {

setBaseSalary(salary);

}//constructor

BasePlusCommissionEmployee::~BasePlusCommissionEmployee() {

cout << "deleting object of class BasePlusCommissionEmployee\n";

}//destructor

void BasePlusCommissionEmployee::setBaseSalary(double salary) {

if (salary >= 0)

baseSalary = salary;

else

throw invalid\_argument("Salary must be >= 0.0");

}//set base salary

double BasePlusCommissionEmployee::getBaseSalary() const {

return baseSalary;

}//return base salary

double BasePlusCommissionEmployee::earnings() const {

return getBaseSalary() + CommissionEmployee::earnings();

}//calculate earnings

void BasePlusCommissionEmployee::print() const {

cout << "base-salaried ";

CommissionEmployee::print();

cout << "; base salary: " << getBaseSalary();

}//print BasePlusCommissionEmployee's information

////////////////////////////////////////////////////////////////////////////////////////////////////

//main.cpp

#include<iostream>

#include<iomanip>

#include<vector>

#include<Windows.h>

using namespace std;

#include"Employee.h"

#include"SalariedEmployee.h"

#include"HourlyEmployee.h"

#include"CommissionEmployee.h"

#include"BasePlusCommissionEmployee.h"

void virtualViaPointer(const Employee\* const);//prototype

int main() {

//print system time

SYSTEMTIME now;

GetLocalTime(&now);

cout << "Time at this moment:\n" << now.wYear << '-'

<< now.wMonth << '-' << now.wDay << ' ' << now.wHour

<< ':' << now.wMinute << ':' << now.wSecond << "\n\n";

//set floatint-point out put formatting

cout << fixed << setprecision(2);

//creat Date objects for employees' birthdates

Date birthDate\_JS(6, 15, 1944);

Date birthDate\_KP(4, 29, 1960);//Karen Price's birthday is today: April,29

Date birthDate\_SJ(9, 8, 1954);

Date birthDate\_BL(3, 2, 1965);

//create derived-class objects

SalariedEmployee salariedEmployee(

"John", "Smith", "111-11-1111", birthDate\_JS, 800);

HourlyEmployee hourlyEmployee(

"Karen", "Price", "222-22-2222", birthDate\_KP, 16.75, 40);

CommissionEmployee commissionEmployee(

"Sue", "Jones", "333-33-3333", birthDate\_SJ, 10000, .06);

BasePlusCommissionEmployee basePlusCommissionEmployee(

"Bob", "Lewis", "444-44-4444", birthDate\_BL, 5000, .04, 300);

//create vector of three base-class pointers

vector<Employee\*>employees(4);

//initialize vector with pointers to Employees

employees[0] = &salariedEmployee;

employees[1] = &hourlyEmployee;

employees[2] = &commissionEmployee;

employees[3] = &basePlusCommissionEmployee;

cout << "Employees processed polymorphically via dynamic binding:\n\n";

//call virtualViaPointer to print each Employee's information and earnings using dynamic binding

cout << "Virtual function calls made off base-class pointers:\n\n";

for (const Employee\* employeePtr : employees)

virtualViaPointer(employeePtr);

system("pause");

return 0;

}//end main

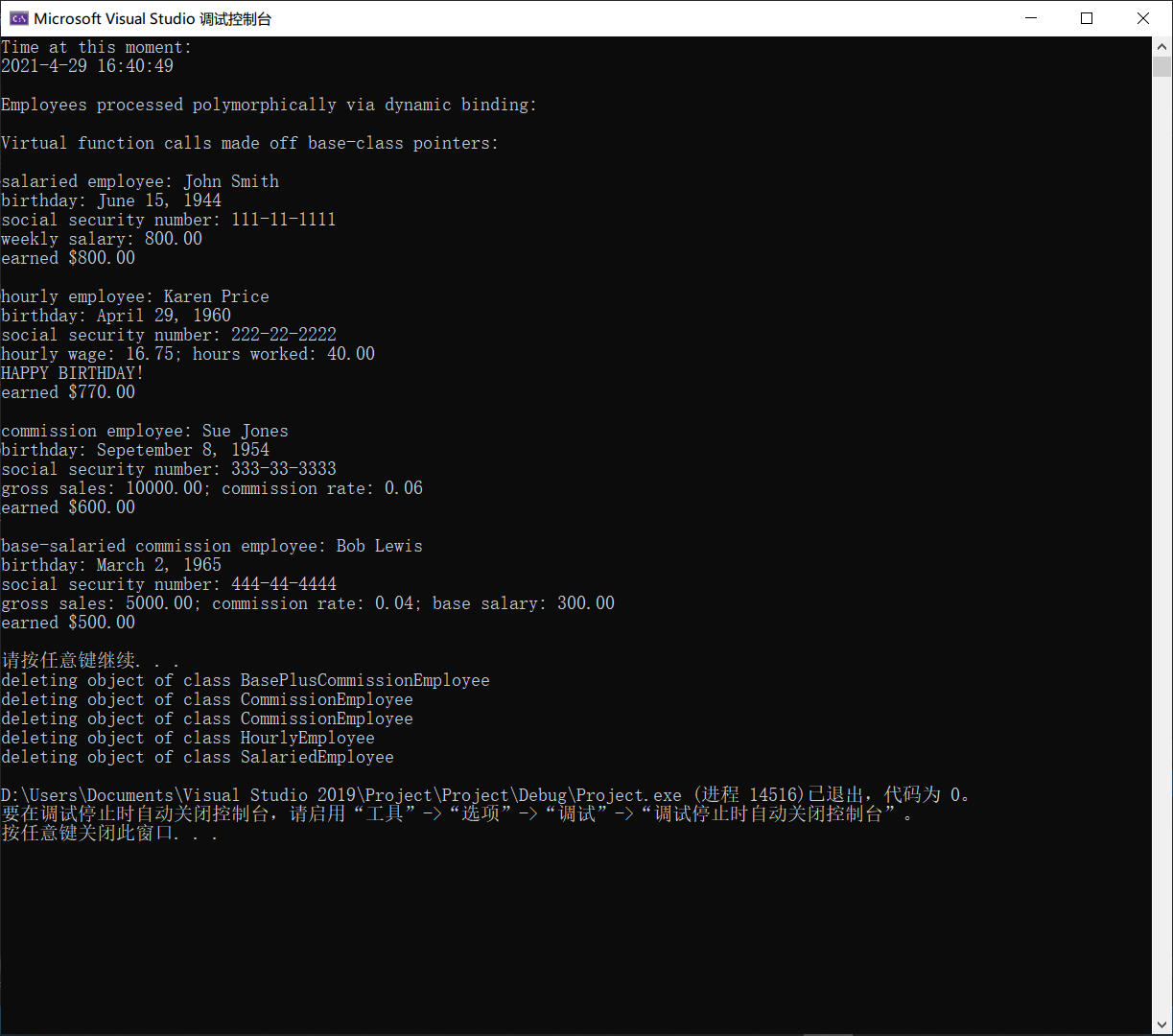
//call Employee virtual functions print and earnings off a base-class pointer using dynamic binding

void virtualViaPointer(const Employee\* const baseClassPtr) {

baseClassPtr->print();

cout << "\nearned $" << (baseClassPtr->Birthday() ? 100 : 0) + baseClassPtr->earnings() << "\n\n";

}



2.

(To make it easier to read, I didn’t write .cpp for each class)

Order:

1. Shape.h
2. Shape\_2D.h
3. Circle.h
4. Reactangle.h
5. Triangle.h
6. Shape\_3D.h
7. Sphere.h
8. Cuboid.h
9. Tetrahedron.h
10. Main.cpp

//Shape.h

#ifndef Shape\_h

#define Shape\_h

class Shape {

public:

Shape(const string& id = " ")

:identity(id) {}//constructor

virtual ~Shape() {}//virtual destructor

void virtual print() const = 0;//pure virtual function

double virtual getArea() const = 0;//pure virtual function

double virtual getVolumn() const = 0;//pure virtual function

protected:

string identity;

};

#endif // !Shape\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Shapes\_2D.h

#ifndef Shapes\_2D\_h

#define Shapes\_2D\_h

#include"Shape.h"

class Shapes\_2D :public Shape {

public:

Shapes\_2D(const string& id = " ", double S = 0.0)

:Shape(id) {

setArea(S);

}//constructor

virtual ~Shapes\_2D() {}//virtual destructor

protected:

double area;

void setArea(double S) {

area = (S > 0) ? S : 0;

}//set area

};

#endif // !Shapes\_2D\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Circle.h

#ifndef Circle\_h

#define Circle\_h

#include"Shapes\_2D.h"

class Circle :public Shapes\_2D {

public:

Circle(const string& id = " ", double r = 0.0)

:Shapes\_2D(id, 0.0){

setRadius(r);

}//constructor

virtual ~Circle() {}//virtual destructor

void setRadius(double r) {

radius = (r > 0) ? r : 0;

}//set radius

double getRadius() const{

return radius;

}//return radius

void print() const {

cout << "Circle: " << identity

<< "\n Radius: "

<< getRadius() << endl;

}//override: print

double getArea() const {

return PI \* pow(radius, 2);

}//override: return area

double getVolumn() const { return -1; }//override: empty

protected:

double radius;

};

#endif // !Circle\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Rectangle.h

#ifndef Rectangle\_h

#define Rectangle\_h

#include"Shapes\_2D.h"

class Rectangle :public Shapes\_2D {

public:

Rectangle(const string& id = " ",

double l = 0.0, double w = 0.0)

:Shapes\_2D(id, 0.0) {

setSides(l, w);

}//constructor

virtual ~Rectangle() {}//virtual destructor

void setSides(double l, double w) {

sides[0] = (l > 0) ? l : 0;

sides[1] = (w > 0) ? w : 0;

}//set 2 sides

const double\* getSides() const{

const double\* temp = sides;

return temp;

}//return 2 sides (pointer)

void print() const {

cout << "Rectangle: " << identity

<< "\n 2 sides: "

<< \*(getSides() + 0) << " , "

<< \*(getSides() + 1) << endl;

}//override: print

double getArea() const {

return sides[0] \* sides[1];

}//override: return area

double getVolumn() const { return -1; }//override: empty

protected:

double sides[2];

};

#endif // !Rectangle\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Triangle.h

#ifndef Triangle\_h

#define Triangle\_h

#include"Shapes\_2D.h"

class Triangle :public Shapes\_2D {

public:

Triangle(const string& id = " ",

double a = 0.0, double b = 0.0, double c = 0.0)

:Shapes\_2D(id, 0.0) {

setSides(a, b, c);

}//constructor

virtual ~Triangle() {}//virtual destructor

void setSides(double a, double b, double c) {

sides[0] = (a > 0) ? a : 0;

sides[1] = (b > 0) ? b : 0;

sides[2] = (c > 0) ? c : 0;

}//set 3 sides

const double\* getSides() const {

const double\* temp = sides;

return temp;

}//return 3 sides (pointer)

void print() const {

cout << "Triangle: " << identity

<< "\n 3 sides: "

<< \*(getSides() + 0) << " , "

<< \*(getSides() + 1) << " , "

<< \*(getSides() + 2) << endl;

}//override: print

double getArea() const {

double p = (sides[0] + sides[1] + sides[2]) / 2;//semi-perimeter

return sqrt(p \* (p - sides[0]) \* (p - sides[1]) \* (p - sides[2]));//Helen Formular

}//override: return area

double getVolumn() const { return -1; }//override: empty

protected:

double sides[3];

};

#endif // !Triangle\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Shapes\_3D.h

#ifndef Shapes\_3D\_h

#define Shapes\_3D\_h

#include"Shape.h"

class Shapes\_3D :public Shape {

public:

Shapes\_3D(const string& id = " ",

double S = 0.0, double V = 0.0)

:Shape(id) {

setSurfaceArea(S);

setVolumn(V);

}//constructor

virtual ~Shapes\_3D() {}//virtual destructor

protected:

double surfaceArea;

double volumn;

void setSurfaceArea(double S) {

surfaceArea = (S > 0) ? S : 0;

}//set surface area

void setVolumn(double V) {

volumn = (V > 0) ? V : 0;

}//set volumn

};

#endif // !Shapes\_2D\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Sphere.h

#ifndef Sphere\_h

#define Sphere\_h

#include"Shapes\_3D.h"

class Sphere :public Shapes\_3D {

public:

Sphere(const string& id = " ", double r = 0.0)

:Shapes\_3D(id, 0.0, 0.0) {

setRadius(r);

}//constructor

virtual ~Sphere() {}//virtual destructor

void setRadius(double r) {

radius = (r > 0) ? r : 0;

}//set radius

double getRadius() const {

return radius;

}//return radius

void print() const {

cout << "Sphere: " << identity

<< "\n Radius: "

<< getRadius() << endl;

}//override: print

double getArea() const {

return 4 \* PI \* pow(radius, 2);

}//override: return area

double getVolumn() const {

return 4 \* PI \* pow(radius, 3) / 3;//cautions: put "/3" to the end

}//override: return volumn

private:

double radius;

};

#endif // !Sphere\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Cuboid.h

#ifndef Cuboid\_h

#define Cuboid\_h

#include"Shapes\_3D.h"

class Cuboid :public Shapes\_3D {

public:

Cuboid(const string& id = " ",

double l = 0.0, double w = 0.0, double h = 0.0)

:Shapes\_3D(id, 0.0, 0.0) {

setSides(l, w, h);

}//constructor

virtual ~Cuboid() {}//virtual destructor

void setSides(double l, double w,double h) {

sides[0] = (l > 0) ? l : 0;

sides[1] = (w > 0) ? w : 0;

sides[2] = (h > 0) ? h : 0;

}//set 3 sides

const double\* getSides() const {

const double\* temp = sides;

return temp;

}//return 3 sides (pointer)

void print() const {

cout << "Cuboid: " << identity

<< "\n 3 sides: "

<< \*(getSides() + 0) << " , "

<< \*(getSides() + 1) << " , "

<< \*(getSides() + 2) << endl;

}//override: print

double getArea() const {

return 2 \* (sides[0] \* sides[1] + sides[0] \* sides[2] + sides[1] \* sides[2]);

}//override: return surface area

double getVolumn() const {

return sides[0] \* sides[1] \* sides[2];

}//override: return volumn

protected:

double sides[3];

};

#endif // !Rectangle\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Tetrahedron.h

#ifndef Tetrahedron\_h

#define Tetrahedron\_h

#include"Shapes\_3D.h"

class Tetrahedron :public Shapes\_3D {

//to make question easier, assume it is regular tetrahedrons

public:

Tetrahedron(const string& id = " ", double s = 0.0)

:Shapes\_3D(id, 0.0, 0.0) {

setSide(s);

}//constructor

virtual ~Tetrahedron() {}//virtual destructor

void setSide(double s) {

side = (s > 0) ? s : 0;

}//set side

double getSide() const {

return side;

}//return side

void print() const {

cout << "Tetrahedron: " << identity

<< "\n Radius: "

<< getSide() << endl;

}//override: print

double getArea() const {

return sqrt(3) \* pow(side, 2);

}//override: return surface area

double getVolumn() const {

return sqrt(2) \* pow(side, 3) / 12;

}//override: return volumn

private:

double side;

};

#endif // !Tetrahedron\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//main.cpp

#include<iostream>

#include<string>

#include<cmath>

#include<vector>

#include<iomanip>

using namespace std;

#define PI 3.1415

//two dimensional shapes:

#include"Circle.h"

#include"Rectangle.h"

#include"Triangle.h"

//three dimensional shapes:

#include"Sphere.h"

#include"Cuboid.h"

#include"Tetrahedron.h"

int main() {

vector<Shape\*>shapes(12);//creat vectors

cout << fixed << setprecision(2);//set floatint-point out put formatting

//two dimensional objects:

Circle cir\_1("No.1", -1.87), cir\_2("No.2", 14.52);

Rectangle rec\_1("No.3", 19.23, 2.95), rec\_2("No.4", 0.12, 28.04);

Triangle tri\_1("No.5", 3, 4, 5), tri\_2("No.6", 1.63, 8.31, 5.64);

shapes[0] = &cir\_1;//cir\_1's radius is less than 0

shapes[1] = &cir\_2;

shapes[2] = &rec\_1;

shapes[3] = &rec\_2;

shapes[4] = &tri\_1;

shapes[5] = &tri\_2;//tri\_2's sides can't form a triangle

//three dimensional objects:

Sphere sph\_1("No.7", 9.21), sph\_2("No.8", 3);

Cuboid cub\_1("No.9", 1, 2, 3), cub\_2("No.10", 5.23, 8.42, 13.34);

Tetrahedron tet\_1("No.11", -1), tet\_2("No.12", 7.93);

shapes[6] = &sph\_1;

shapes[7] = &sph\_2;

shapes[8] = &cub\_1;

shapes[9] = &cub\_2;

shapes[10] = &tet\_1;

shapes[11] = &tet\_2;//tet\_2's side is less than 0

for (int i = 0; i < 12; i++) {

shapes[i]->print();

if (shapes[i]->getVolumn() == -1)//2D shapes

cout << " Area: "

<< shapes[i]->getArea() << "\n\n";

else//3D shapes

cout << " Surface Area: "

<< shapes[i]->getArea()

<< "\n Volumn: "

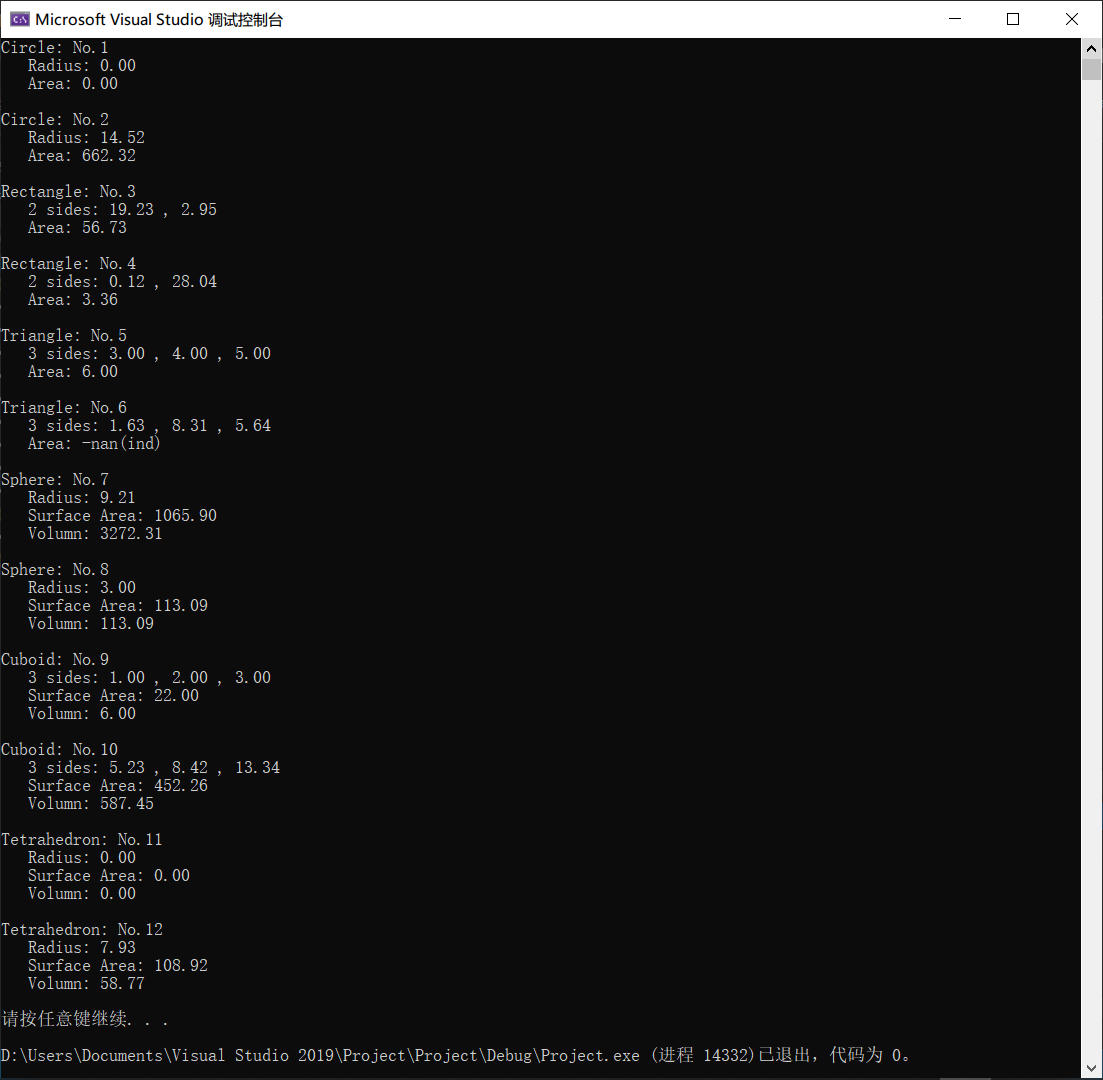
<< shapes[i]->getVolumn() << "\n\n";

}//print all

system("pause");

return 0;

}//end main



3.

//Account.h

#ifndef Account\_h

#define Account\_h

class Account {

public:

Account(double = 0.0);

virtual ~Account() {}

void virtual credit(double);

void virtual debit(double);

double getBalance();

double virtual calculateInterest() = 0;

protected:

double balance;

};

#endif // !Account\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//Account.cpp

#include<iostream>

using namespace std;

#include"Account.h"

Account::Account(double b){

if (b < 0.0) {

balance = 0.0;

cout << "Error\n"

<< "Account balance mustn't be less than 0.0.\n";

}

else

balance = b;

};

void Account::credit(double money) {

if (money <= 0)

cout << "Error\n"

<< "Credit amount must be greater than 0.0.\n";

else {

balance += money;

cout << money << "$ has been credited to your account.\n";

}

}

void Account::debit(double money) {

if (money <= 0 || money > getBalance())

cout << "Error\n"

<< "Debit amount exceeded account balance or is less than 0.0.\n";

else {

balance -= money;

cout << money << "$ has been debited to your account.\n";

}

}

double Account::getBalance(){

return balance;

}//Account.cpp

#include<iostream>

using namespace std;

#include"Account.h"

Account::Account(double b){

if (b < 0.0) {

balance = 0.0;

cout << "Error\n"

<< "Account balance mustn't be less than 0.0.\n";

}

else

balance = b;

};

void Account::credit(double money) {

if (money <= 0)

cout << "Error\n"

<< "Credit amount must be greater than 0.0.\n";

else {

balance += money;

cout << money << "$ has been credited to your account.\n";

}

}

void Account::debit(double money) {

if (money <= 0 || money > getBalance())

cout << "Error\n"

<< "Debit amount exceeded account balance or is less than 0.0.\n";

else {

balance -= money;

cout << money << "$ has been debited to your account.\n";

}

}

double Account::getBalance(){

return balance;

}

////////////////////////////////////////////////////////////////////////////////////////////////////

//SavingsAccount.h

#ifndef SavingsAccount\_h

#define SavingsAccount\_h

#include"Account.h"

class SavingsAccount :public Account {

public:

SavingsAccount(double = 0.0, double = 0.0);

virtual ~SavingsAccount() {}

double calculateInterest();

private:

double rate;

};

#endif // !SavingsAccount\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//SavingsAccount.cpp

#include<iostream>

using namespace std;

#include"SavingsAccount.h"

SavingsAccount::SavingsAccount(double b, double r)

:Account(b) {

if (r < 0.0 || r > 1.0) {

rate = 0.0;

cout << "Error\n"

<< "Interest rate mustn't be less than 0.0 or more than 1.0.\n";

}

else

rate = r;

}

double SavingsAccount::calculateInterest() {

double temp = balance \* rate;

balance += temp;

return temp;

}

////////////////////////////////////////////////////////////////////////////////////////////////////

//CheckingAccount.h

#ifndef CheckingAccount\_h

#define CheckingAccount\_h

#include"Account.h"

class CheckingAccount :public Account {

public:

CheckingAccount(double = 0.0, double = 0.0);

virtual ~CheckingAccount() {}

void credit(double);

void debit(double);

double calculateInterest() { return 0; }

private:

double fee;

};

#endif // !CheckingAccount\_h

////////////////////////////////////////////////////////////////////////////////////////////////////

//CheckingAccount.cpp

#include<iostream>

using namespace std;

#include"CheckingAccount.h"

CheckingAccount::CheckingAccount(double b, double f)

:Account(b) {

if (f < 0.0) {

fee = 0.0;

cout << "Error\n"

<< "Fee charged mustn't be less than 0.0 or more than 1.0.\n";

}

else

fee = f;

}

void CheckingAccount::credit(double money) {

if (money - fee > 0)

Account::credit(money - fee);

else {

balance += (money - fee);

cout << money << "$ has been credited to your account.\n";

}

cout << "(fee: " << fee << ")\n";

}

void CheckingAccount::debit(double money) {

if(money<0)

cout << "Error\n"

<< "Debit amount exceeded account balance or is less than 0.0.\n";

else {

if (money + fee > 0 || money + fee <= balance) {

Account::debit(money + fee);

}

else {

balance -= (money + fee);

cout << money << "$ has been debited to your account.\n";

}

}

cout << "(fee: " << fee << ")\n";

}

////////////////////////////////////////////////////////////////////////////////////////////////////

//main.cpp

#include<iostream>

#include<iomanip>

#include<vector>

#include<cstdlib>

#include<ctime>

using namespace std;

#include"SavingsAccount.h"

#include"CheckingAccount.h"

int main() {

cout << fixed << setprecision(2);

srand(time(0));

vector<Account\*>accounts(4);

accounts[0] = new SavingsAccount(double(rand()), double(rand()) / RAND\_MAX / 10);

accounts[1] = new SavingsAccount(double(rand()), double(rand()) / RAND\_MAX / 10);

accounts[2] = new CheckingAccount(double(rand()), double(rand()) / RAND\_MAX \* 5.0);

accounts[3] = new CheckingAccount(double(rand()), double(rand()) / RAND\_MAX \* 5.0);

for (int i = 0; i < 4; i++) {

cout << "Account\_" << i + 1 << ":\n";

cout << " Account balance: " << accounts[i]->getBalance();

if (i < 2) {

cout << "\n Interest: " << accounts[i]->calculateInterest();

cout << "\n Sum: " << accounts[i]->getBalance();

}

cout << "\n\n How much to withdraw from your account?\n ";

double temp1;

cin >> temp1;

cout << " ";

accounts[i]->debit(temp1);

cout << " Now your account balance is: " << accounts[i]->getBalance();

cout << "\n\n How much to deposit into your account?\n ";

double temp2;

cin >> temp2;

cout << " ";

accounts[i]->credit(temp2);

cout << " Now your account balance is: " << accounts[i]->getBalance() << endl << endl;

delete accounts[i];

}

system("pause");

//for (int i = 0; i < 4; i++)

//delete accounts[i];

return 0;

}//end main

