AERSP 424: Advanced Computer Programming

Submission Instructions: Homework 6
Due: 02/12/19

Submission Instructions:

- Submit the file with the .cpp extension containing your C++ source code.
- For questions 1, 2, 3 and 4 submit cpp and h files.
- Teams of up to 3 are allowed. Put the names of all team members on a single submission.
- Suggestion: use the Visual Studio compiler to check and/or verify your work.

Use the Circle.h, Cylinder.h, Point.h and Circle.cpp, Cylinder.cpp, Point.cpp files I posted on Canvas for this homework. Complete the following:

1. We have explored the creation of a **`Point, Circle, Cylinder`** class hierarchy. I have provided code for those classes on Canvas. Now create and add the following classes to the hierarchy: **`Triangle, Scalene, Isosceles, and Equilateral.`** The **`Triangle`** class is derived from the point class. The **`Scalene`** class is derived from the **`Triangle`** class. The **`Isosceles`** class is derived from the **`Scalene`** class. The **`Equilateral`** class is derived from the **`Isosceles`** class.

   See Q2 below.

2. Add data members and setters and getters that contain the properties inherent to these types of shapes. For example, a rectangle has a length and a width. A triangle has three side lengths and angles. Define equivalence operators for each type of object. Add inherited members functions called **`area`**() and **`perimeter`**().

```
#ifndef TRIANGLE_H_
#define TRIANGLE_H_

#include "Point.h"

class Triangle : public Point {

public:
	Triangle(double sOne, double sTwo, double sThree);

	bool operator==(const Triangle);

	double area();
	double perimeter();


	double getSideOne() const		{	return dSideOne;	}
	double getSideTwo() const		{	return dSideTwo;	}
	double getSideThree() const			{	return dSideThree;  }
	double getAngleOne() const		{	return dAngleOne;	}
	double getAngleTwo() const		{	return dAngleTwo;	}
	double getAngleThree() const		{	return dAngleThree; }

	void  setSideOne(double a)		{	dSideOne=a;			}
	void  setSideTwo(double a)		{	dSideTwo=a;			}
	void  setSideThree(double a)		{	dSideThree=a;		}
	void  setAngleOne(double a)			{	dAngleOne=a;		}
	void  setAngleTwo(double a)			{	dAngleTwo=a;		}
	void  setAngleThree(double a)	{	dAngleThree=a;			}

protected:
	double dSideOne;
	double dSideTwo;
	double dSideThree;

	double dAngleOne;
	double dAngleTwo;
	double dAngleThree;

	const double epsilon;

private:
	void calculateAngles();

};
#endif /* TRIANGLE_H_ */
```

Triangle.cpp

```
#include "Triangle.h"
#include <math.h>	/* fabs */

Triangle::Triangle(double sOne, double sTwo, double sThree):Point(-1, -1),
epsilon(0.1)
{
```

```
	dSideOne=sOne;
	dSideTwo=sTwo;
	dSideThree=sThree;
	dAngleThree=0;
	dAngleOne=0;
	dAngleTwo=0;

	calculateAngles();
}

bool Triangle::operator==(const Triangle rhs) {
	if(fabs(this->dSideOne - rhs.getSideOne())>epsilon)
		return false;
	else if(fabs(this->dSideTwo - rhs.getSideTwo())>epsilon)
		return false;
	else if(fabs(this->dSideThree - rhs.getSideThree())>epsilon)
		return false;
	else
		return true;
}

void Triangle::calculateAngles() {

	double num;
	double den;

	if(dSideOne==0 || dSideTwo==0 || dSideThree==0)
		return;

	num = pow(dSideOne,2)+pow(dSideTwo,2)-pow(dSideThree,2);
	den = 2*dSideOne*dSideTwo;
	dAngleThree=acos(num / den)*(180/M_PI);

	num = pow(dSideTwo,2)+pow(dSideThree,2)-pow(dSideOne,2);
	den = 2*dSideTwo*dSideThree;
	dAngleOne=acos(num / den)*(180/M_PI);


	num = pow(dSideThree,2)+pow(dSideOne,2)-pow(dSideTwo,2);
	den = 2*dSideThree*dSideOne;
	dAngleTwo=acos(num / den)*(180/M_PI);
}

double Triangle::area() {

	double p = this->perimeter()/2;
	double temp = (p*(p-this->dSideOne)*(p-this->dSideTwo)*(p-this->dSideThree));

	return sqrt(temp);
}

double Triangle::perimeter() {
	return (dSideOne+dSideTwo+dSideThree);
}
```

```
#ifndef SCALENE_H_
#define SCALENE_H_

#include "Triangle.h"

class Scalene : public Triangle {

public:
	Scalene(double sOne, double sTwo, double sThree);

	bool operator==(const Scalene);

	void  setSideOne(double a);
	void  setSideTwo(double a);
	void  setSideThree(double a);


private:
	bool CheckScalene(double a, double b, double c);
};

#endif /* SCALENE_H_ */
```

Scalene.cpp

```
#include "Scalene.h"
#include "Triangle.h"
#include <math.h>	/* fabs */

Scalene::Scalene(double sOne, double sTwo, double sThree):Triangle(sOne,sTwo,sThree)
{

}

//You could have just created an opertor for triangle.
bool Scalene::operator==(const Scalene rhs){

	if(fabs(this->dSideOne - rhs.getSideOne())>epsilon)
		return false;
	else if(fabs(this->dSideTwo - rhs.getSideTwo())>epsilon)
		return false;
	else if(fabs(this->dSideThree - rhs.getSideThree())>epsilon)
		return false;
	else
		return true;
}


void  Scalene::setSideOne(double a)    {
	if(CheckScalene(a,dSideTwo,dSideThree)==true)
		dSideOne=a;
}

void  Scalene::setSideTwo(double a) {
	if(CheckScalene(dSideOne,a,dSideThree)==true)
```

```
            dSideTwo=a;
}

void  Scalene::setSideThree(double a) {
      if(CheckScalene(dSideOne,dSideTwo,a)==true)
            dSideThree=a;
}

bool Scalene::CheckScalene(double a, double b, double c) {
      if(a==b || a==c || b==c) {
            return false;
      }
      return true;
}
```

```
#ifndef ISOSCELES_H_
#define ISOSCELES_H_

#include "Scalene.h"

class Isosceles : public Scalene {

public:
	Isosceles(double sOne, double sTwo, double sThree);

	bool operator==(const Isosceles);

	void  setSideOne(double a);
	void  setSideTwo(double a);
	void  setSideThree(double a);


private:
	bool CheckIsosceles(double a, double b, double c);
};
#endif /* ISOSCELES_H_ */
```

Isosceles.cpp

```
Isosceles::Isosceles(double sOne, double sTwo, double
sThree):Scalene(sOne,sTwo,sThree)
{

}

bool Isosceles::operator==(const Isosceles rhs){

	if(fabs(this->dSideOne - rhs.getSideOne())>epsilon)
		return false;
	else if(fabs(this->dSideTwo - rhs.getSideTwo())>epsilon)
		return false;
	else if(fabs(this->dSideThree - rhs.getSideThree())>epsilon)
		return false;
	else
		return true;
}

void  Isosceles::setSideOne(double a)  {
	if(CheckIsosceles(a,dSideTwo,dSideThree)==true)
		dSideOne=a;
}

void  Isosceles::setSideTwo(double a) {
	if(CheckIsosceles(dSideOne,a,dSideThree)==true)
		dSideTwo=a;
}

void  Isosceles::setSideThree(double a) {
```

```
	if(CheckIsosceles(dSideOne,dSideTwo,a)==true)
		dSideThree=a;
}

bool Isosceles::CheckIsosceles(double a, double b, double c) {
	if(a==b && a==c && b==c)
		return false;

	if(a==b || a==c || b==c)
		return true;

	return false;
}
```

```
#ifndef EQUILATERAL_H_

#define EQUILATERAL_H_


#include "Isosceles.h"

class Equilateral : public Isosceles {

public:
	Equilateral(double sOne);

	bool operator==(const Equilateral);

	void  setSideOne(double a);
	void  setSideTwo(double a);
	void  setSideThree(double a);


private:
	bool CheckEquilateral(double a, double b, double c);
};

#endif /* EQUILATERAL_H_ */
```

Equilateral.cpp

```
#include "Equilateral.h"
#include "Triangle.h"
#include <math.h>	/* fabs */

Equilateral::Equilateral(double sOne):Isosceles(sOne,sOne,sOne)
{

}

bool Equilateral::operator==(const Equilateral rhs){
	if(fabs(this->dSideOne - rhs.getSideOne())>epsilon)
		return false;
	else
		return true;
}

void  Equilateral::setSideOne(double a){
	if(CheckEquilateral(a,dSideTwo,dSideThree)==true)
		dSideOne=a;
}

void  Equilateral::setSideTwo(double a) {
	if(CheckEquilateral(dSideOne,a,dSideThree)==true)
		dSideTwo=a;
}

void  Equilateral::setSideThree(double a) {
```

```
	if(CheckEquilateral(dSideOne,dSideTwo,a)==true)
		dSideThree=a;
}

bool Equilateral::CheckEquilateral(double a, double b, double c) {
	if(a==b && a==c && b==c)
		return true;
	else
		return false;
}
```

3. Create an abstract base class called **AbstractTriangle** from which **Triangle** is inherited. Create a virtual function called **area**. Create a pure virtual function called **printName**() which prints the shape's name.

```
#ifndef ABSTRACTTRIANGLE_H_
#define ABSTRACTTRIANGLE_H_

class AbstractTriangle {

public:
   virtual void printName() =0;
   virtual double area()           {     return -1;   }
   virtual double perimeter()      {     return -1;   }
   virtual int getX()              {     return -1;   }
   virtual int getY()              {     return -1;   }

   virtual ~AbstractTriangle()     {};


};
```

You also need to add a printName function for each derived class.

4. In main, create an array of **AbstractTriangle** pointers. Create a randomly valued **Triangle, Scalene, Isosceles, and Equilateral** objects. Set each value in the array to point to a different randomly created object. Use a loop to iterate through the array and print the shape's name, area, perimeter, and x, y location, if these member functions exist.

In main

```
Triangle myTriangle(8,6,7);
Scalene myScalene(10,10,3);
Isosceles myIsosceles(1,5,5);
Equilateral myEquilateral(2);

AbstractTriangle* myArray[4];
myArray[0]=&myScalene;
myArray[1]=&myIsosceles;
myArray[2]=&myTriangle;
myArray[3]=&myEquilateral;
```

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

	myArray[i]->printName();
	cout << "AREA: "<<myArray[i]->area()<<endl;
	cout << "PERIMETER: "<<myArray[i]->perimeter()<<endl;
	cout << "[ "<<myArray[i]->getX()<<","<<myArray[i]->getY()<<" ]"<<endl;
	cout << endl;
}
```