|  |
| --- |
| **Polygon** |
| - numOfEdge: int  - length : int |
| +Polygon()  +setNumOfEdge(ne: int)  + getNumOfEdge():int  + setLength (ln:int)  + getLength():int  + Perimeter():int  +CalculateInternalAngel() |

Write a program (int main()) that calculates perimeter and total internal angel values of the given polygon ( define the Polygon class first) . Polygon has number of edges and length of one edge. Length of each edges of this polygon is equal.

**Polygon.h**

class Polygon

{

private:

int numOfEdge;

int length;

public:

Polygon(){}; //default constructor

void setNumOfEdge(int ne);

int getNumOfEdge();

void setLength(int ln);

int getLength();

int Perimeter();

void CalculateInternalAngel();

};

**Polygon.cpp**

#include "Polygon.h"

#include <iostream>

using namespace std;

void Polygon::setNumOfEdge(int ne)

{

if (ne >= 3)

{

numOfEdge = ne;

}

}

int Polygon::getNumOfEdge()

{

return numOfEdge;

}

void Polygon::setLength(int ln)

{

if (ln>0)

{

length = ln;

}

}

int Polygon::getLength()

{

return length;

}

int Polygon::Perimeter()

{

return numOfEdge \* length;

}

void Polygon::CalculateInternalAngel()

{

cout << "Internal angels total value: " << (numOfEdge - 2) \* 180 << endl;

}

**Solution.cpp**

#include <iostream>

#include "Polygon.h"

using namespace std;

int main()

{

int noe, lgth;

cout << "Enter number of edge: ";

cin >> noe;

cout << "Enter length: ";

cin >> lgth;

Polygon p1;

p1.setNumOfEdge(noe);

p1.setLength(lgth);

cout << "Perimeter: " << p1.Perimeter() << endl;

p1.CalculateInternalAngel();

system("pause");

return 0;

}

//DYNAMIC CAST

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() = 0;

};

class Rectangle : public Shape {

public:

void draw() {

cout << "Drawing a rectangle" << endl;

}

};

class Circle : public Shape {

public:

void draw() {

cout << "Drawing a circle" << endl;

}

};

int main() {

Shape\* shape1 = new Rectangle();

Shape\* shape2 = new Circle();

Rectangle\* rect = dynamic\_cast<Rectangle\*>(shape1);

if (rect) {

cout << "Shape1 is a rectangle" << endl;

rect->draw();

}

else {

cout << "Shape1 is not a rectangle" << endl;

}

Circle\* circle = dynamic\_cast<Circle\*>(shape2);

if (circle) {

cout << "Shape2 is a circle" << endl;

circle->draw();

}

else {

cout << "Shape2 is not a circle" << endl;

}

delete shape1;

delete shape2;

system("pause");

return 0;

}

// Abstract base class

#include <iostream>

using namespace std;

// Abstract base class

class Shape {

public:

virtual double area() = 0;

virtual double perimeter() = 0;

};

// Derived class

class Rectangle : public Shape {

private:

double width;

double height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() {

return width \* height;

}

double perimeter() {

return 2 \* (width + height);

}

};

// Derived class

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() {

return 3.14 \* radius \* radius;

}

double perimeter() {

return 2 \* 3.14 \* radius;

}

};

int main() {

Shape\* shape1 = new Rectangle(4, 5);

Shape\* shape2 = new Circle(6);

cout << "Area of Rectangle: " << shape1->area() << endl;

cout << "Perimeter of Rectangle: " << shape1->perimeter() << endl;

cout << "Area of Circle: " << shape2->area() << endl;

cout << "Perimeter of Circle: " << shape2->perimeter() << endl;

delete shape1;

delete shape2;

system("pause");

return 0;

}

//EXCEPTION

#include <iostream>

#include <string>

using namespace std;

class MyException : public exception {

public:

const char\* what() const throw() {

return "My custom exception";

}

};

int main() {

try {

string s;

cout << "Enter a string : \n ";

cin >> s;

if (s.length() < 5) {

throw MyException();

}

else {

cout << "Your string is: " << s << endl;

}

}

catch (MyException& e) {

cerr << "Error: " << e.what() << endl;

}

system("pause");

return 0;

}

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<int> numbers = { 1, 2, 3 };

try {

int index = 5;

//int index = 2;

if (index < 0 || index >= numbers.size())

{

throw out\_of\_range("Index out of range!");

}

cout << "Value at index " << index << ": " << numbers.at(index) << endl;

}

catch (const out\_of\_range& error)

{

cerr << "Error: " << error.what() << endl;

}

system("pause");

return 0;

}

#include <iostream>

//#include <stdexcept>

using namespace std;

void doSomething() {

try {

throw runtime\_error("Error in doSomething()");

}

catch (exception& e) {

cerr << "Caught an exception in doSomething(): " << e.what() << endl;

throw; // rethrow the same exception

}

}

int main() {

try {

doSomething();

}

catch (exception& e) {

cerr << "Caught an exception in main(): " << e.what() << endl;

}

system("pause");

return 0;

}