Has-a, Destructors

ITP 165 – Fall 2015

Week 12, Lecture 1

A Point Class (in Point.h)

```
#pragma once

class Point
{
public:
   // Constructors
   Point();
   Point(double x, double y);

   // Getters
   double getX();
   double getY();

   // Setters
   void setX(double inX);
   void setY(double inY);

   // Display
   void print();

private:
   double mXCoord;
   double mYCoord;
};
```

A Point Class (in Point.cpp)

```
#include "Point.h"
#include <iostream>

Point::Point()
{
   mXCoord = 0.0;
   mYCoord = 0.0;
}

Point::Point(double x, double y)
{
   mXCoord = x;
   mYCoord = y;
}

void Point::setX(double inX)
{
   mXCoord = inX;
}

void Point::setY(double inY)
{
   mYCoord = inY;
}
```

```
void Point::set(double inX, double inY)
{
   mXCoord = inX;
   mYCoord = inY;
}

double Point::getX()
{
   return mXCoord;
}

double Point::getY()
{
   return mYCoord;
}

void Point::print()
{
   std::cout << "(" << mXCoord << ","
      << mYCoord << ")";
}
```

Circle

- A circle has a center and a radius
- The center can be expressed as a point.
- So we could say the circle ***has-a*** point!! Or in other words, the circle should have an instance of a point as a member variable.
- This is a perfect opportunity to use a has-a relationship (aka composition)

A Circle class (in Circle.h)

```
#pragma once
#include "Point.h"

class Circle {
private:
   Point mCenter;
   double mRadius;
public:
   // Constructor with parameters (Default constructor not allowed!)
   Circle(double x, double y, double rad);

   // Getters/setter
   Point getCenter();
   double getRadius();
   void set(double x, double y, double rad);

   // Calculates area of circle
   double calcArea();
};
```

A Circle class (in Circle.cpp)

```
#include "Circle.h"
#include <iostream>

// Constructor
Circle::Circle(double x, double y, double rad)
{
   set(x, y, rad);
}

// Getters/setters
void Circle::set(double x, double y, double rad)
{
   mCenter.setX(x);
   mCenter.setY(y);
   mRadius = rad;
}
```

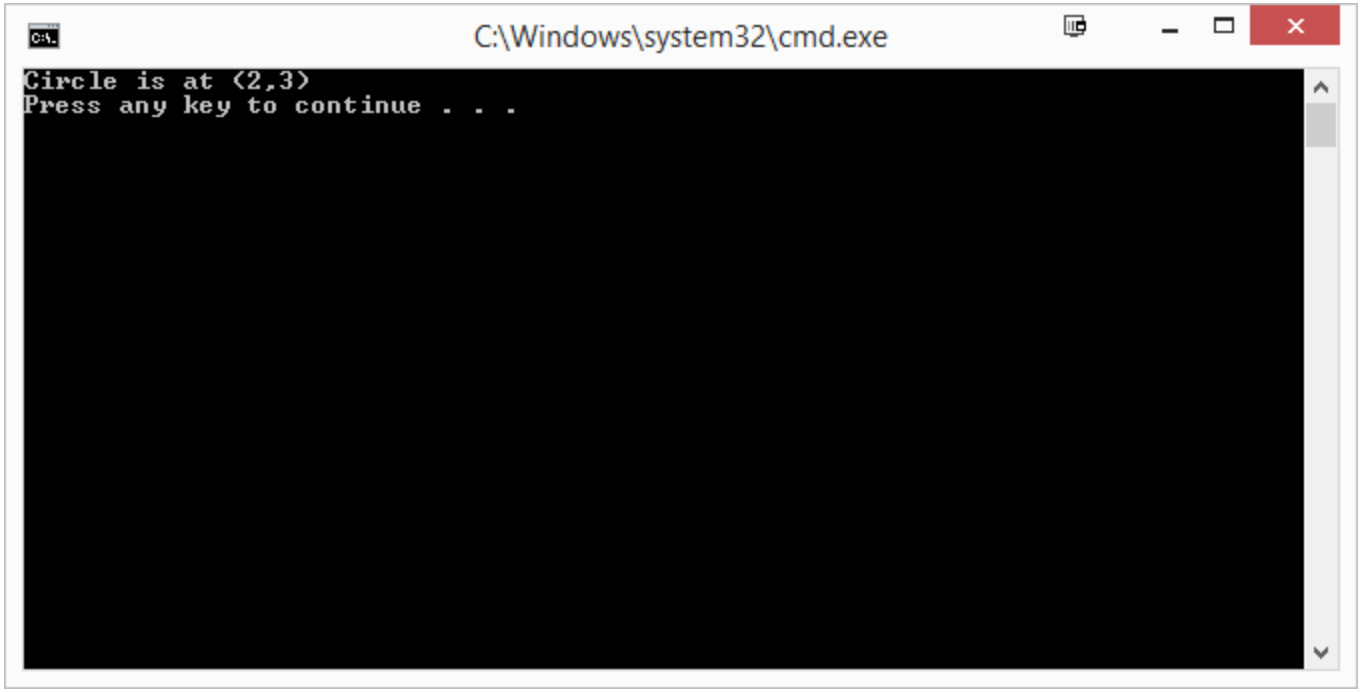
```
Point Circle::getCenter() {
   return mCenter;
}

double Circle::getRadius() {
   return mRadius;
}

// Calculate area of circle
double Circle::calcArea() {
   return mRadius * mRadius * 3.141592;
}
```

Circle in Action

```
// main.cpp
#include "Circle.h"
#include <iostream>

int main() {
   // Create a circle
   Circle testCirc(2, 3, 5);
   // Grab the center
   Point center = testCirc.getCenter();

   std::cout << "Circle is at (";
   std::cout << center.getX() << ",";
   std::cout << center.getY() << ")" << std::endl;
   return 0;
}
```

Circle in Action

- The order of the Point outputs vs. the Circle outputs may not be entirely in the order you might expect…



Rectangle

- A rectangle can be represented by ***two*** points – a bottom left and a top right point.

- So we can use composition again!!

- This time, it will need two point member variables

A Rect class (in Rect.h)

```
#pragma once
#include "Point.h"

class Rect
{
private:
   Point mBotLeft;
   Point mTopRight;
public:
   // Constructor w/ parameters
   Rect(double botX, double botY, double topX, double topY);

   // Getters/setter
   Point getBotLeft();
   Point getTopRight();
   void set(double botX, double botY, double topX, double topY);

   // Calculate area of rectangle
   double calcArea();
};
```

A Rect Class (in Rect.cpp)

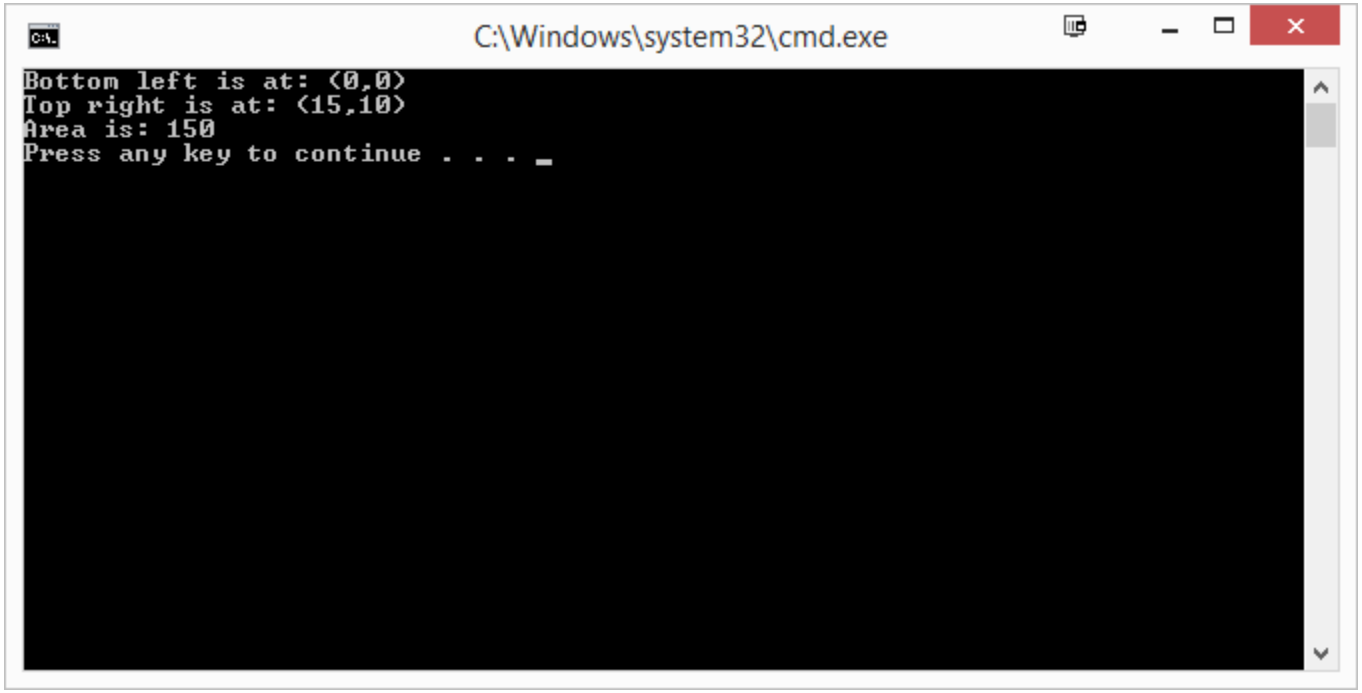
```
#include "Rect.h"
#include <iostream>

Rect::Rect(double botX, double botY,
   double topX, double topY)
{
   set(botX, botY, topX, topY);
}

// Getters/setter
Point Rect::getBotLeft()
{
   return mBotLeft;
}

Point Rect::getTopRight()
{
   return mTopRight;
}
```

```
void Rect::set(double botX, double botY,
   double topX, double topY)
{
   mBotLeft.set(botX, botY);
   mTopRight.set(topX, topY);
}

// Calculate area of rectangle
double Rect::calcArea()
{
   double length = mTopRight.getX() -
      mBotLeft.getX();
   double width = mTopRight.getY() -
      mBotLeft.getY();
   return length * width;
}
```

Rect in Action

```
// main.cpp
#include "Shapes.h"
#include <iostream>

int main() {
   // Create a circle
   Rect testRect(0, 0, 15, 10);
   // Grab the bottom left
   Point botLeft = testRect.getBotLeft();
   // Grab top right
   Point topRight = testRect.getTopRight();

   std::cout << "Bottom left is at: ";
   std::cout << "(" << botLeft.getX() << ",";
   std::cout << botLeft.getY() << ")" << std::endl;

   std::cout << "Top right is at: ";
   std::cout << "(" << topRight.getX() << ",";
   std::cout << topRight.getY() << ")" << std::endl;

   std::cout << "Area is: " << testRect.calcArea();
   std::cout << std::endl;
   return 0;
}
```

Rect in Action



Triangle

- A triangle can be represented by its three points (one for each corner)

- So we can use points again, this time in an array form!

A Tri class (in Tri.h)

```
#pragma once
#include "Point.h"

class Tri
{
private:
   Point mCorners[3];
public:
   // Constructor w/ parameters
   Tri(double x0, double y0, double x1, double y1, double x2, double y2);

   // Getters/setter
   Point getCorner(unsigned int index);
   void set(double x0, double y0, double x1, double y1, double x2, double y2);
   // Calculate the area of the triangle
   double calcArea();
};
```

A Tri Class (in Tri.cpp)

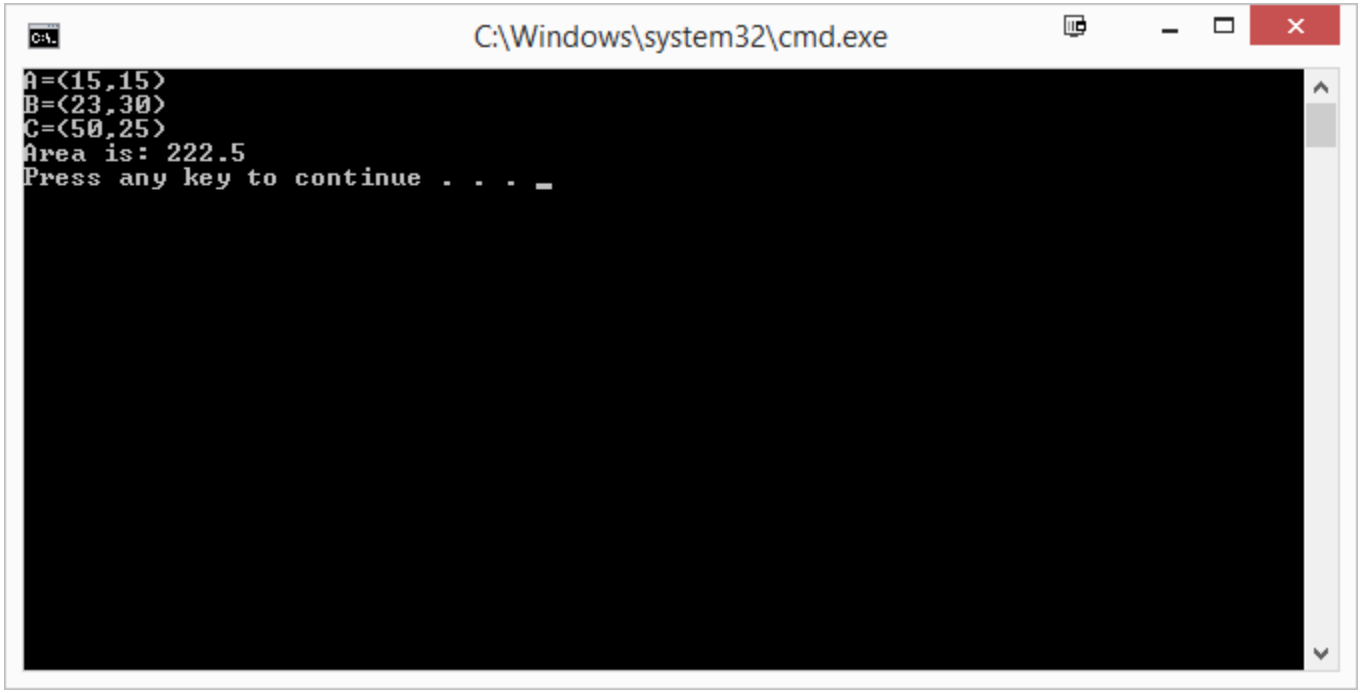
```
// In Shapes.cpp, after Rect implementations
Tri::Tri(double x0, double y0,
   double x1, double y1,
   double x2, double y2)
{
   set(x0, y0, x1, y1, x2, y2);
}

// Getter/setter
Point Tri::getCorner(unsigned int index)
{
   // Note: we should validate index
   return mCorners[index];
}

void Tri::set(double x0, double y0,
   double x1, double y1,
   double x2, double y2)
{
   mCorners[0].set(x0, y0);
   mCorners[1].set(x1, y1);
   mCorners[2].set(x2, y2);
}
```

```
// Calculate area of triangle
double Tri::calcArea()
{
   // Formula for area (thanks Internet):
   // | (Ax(By-Cy) + Bx(Cy-Ay) + Cx(Ay-By)) |
   // | ----------------------------------- |
   // |                  2                  |

   Point A = mCorners[0];
   Point B = mCorners[1];
   Point C = mCorners[2];

   // i = Ax(By-Cy)
   double i = A.getX() * (B.getY() - C.getY());
   // j = Bx(Cy-Ay)
   double j = B.getX() * (C.getY() - A.getY());
   // k = Cx(Ay - By)
   double k = C.getX() * (A.getY() - B.getY());

   double area = (i + j + k) / 2;

   // std::abs is absolute value in <cmath>
   area = std::abs(area);
   return area;
}
```

Tri in Action

```
// main.cpp
#include "Shapes.h"
#include <iostream>

int main() {
   // Create a triangle
   Tri test(15, 15, 23, 30, 50, 25);
   Point A = test.getCorner(0);
   Point B = test.getCorner(1);
   Point C = test.getCorner(2);

   std::cout << "A=(" << A.getX() << ",";
   std::cout << A.getY() << ")" << std::endl;
   std::cout << "B=(" << B.getX() << ",";
   std::cout << B.getY() << ")" << std::endl;
   std::cout << "C=(" << C.getX() << ",";
   std::cout << C.getY() << ")" << std::endl;

   std::cout << "Area is: " << test.calcArea() << std::endl;
   return 0;
}
```

Tri in Action



A further modification

- We could change the constructors for `Circle`/`Rect`/`Tri` so that they actually take in `Point` classes (by reference) instead of separate x/y coordinates.

- This is probably how you'd ***really*** want to do it, but I didn't for sake of simplicity

What if?

- Triangle has-a constant sized array, what if it was non-constant sized?

- ***Dynamic memory to the rescue!!***

- In the Triangle constructor, we dynamically create the array for Points!

Problem…

- Remember when we talked about **new**, we said we should always have a corresponding **delete**

- Otherwise we get a memory leak

- Where is the corresponding delete in this case?

Destructor

- A ***destructor*** is a special type of member function that automatically gets called when the object goes out of scope

- We'll write a destructor today (together) on today's lab…

Lab practical #20

- Let’s write an address book!