Is-a

ITP 165 – Fall 2015

Week 13, Lecture 1

Last week…

- In lecture, we made the following classes:
  - `Point`
  - `Circle`
  - `Rect`
  - `Tri`

Using all of the shapes

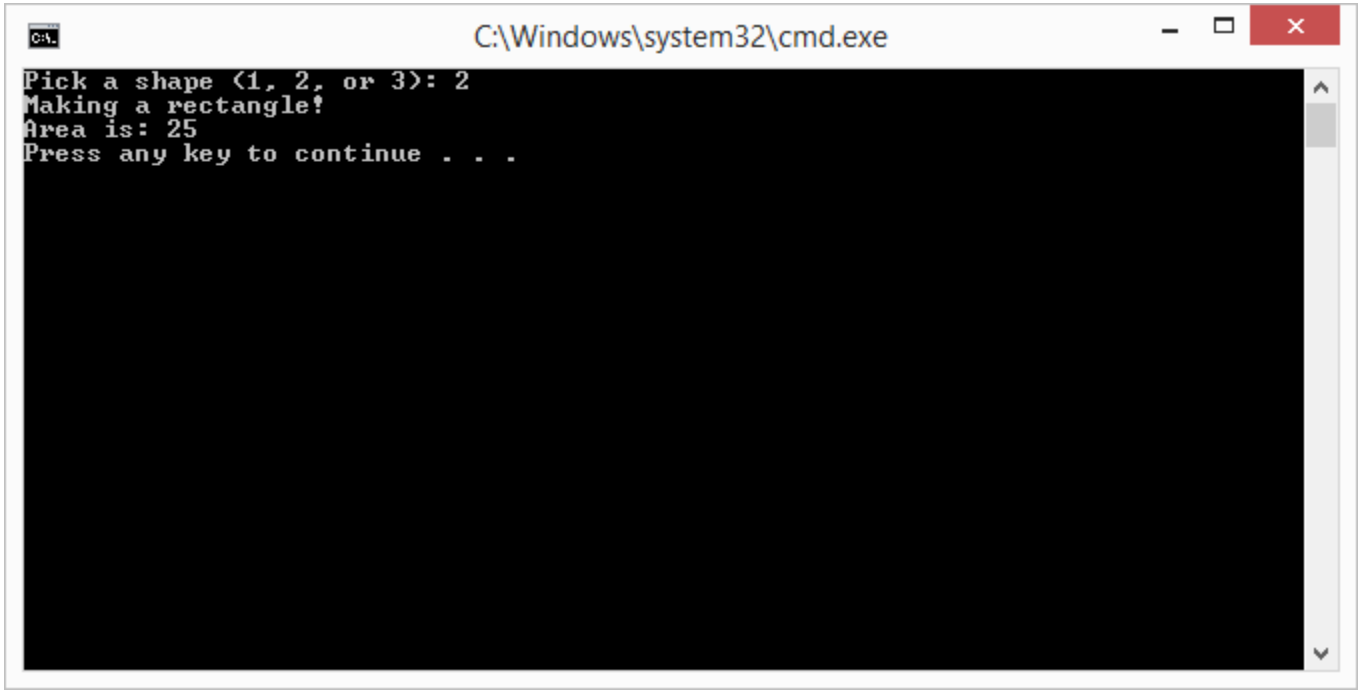
```
// main.cpp

int main() {
   std::cout << "Pick a shape (1, 2, or 3): ";
   int option = 0;
   std::cin >> option;

   if (option == 1) {
      std::cout << "Making a circle!" << std::endl;
      Circle myCircle(0, 0, 5);
      std::cout << "Area is: " << myCircle.calcArea() << std::endl;
   }
   else if (option == 2) {
      std::cout << "Making a rectangle!" << std::endl;
      Rect myRect(0, 0, 5, 5);
      std::cout << "Area is: " << myRect.calcArea() << std::endl;
   }
   else {
      std::cout << "Making a triangle!" << std::endl;
      Tri myTri(0, 0, 0, 5, 5, 5);
      std::cout << "Area is: " << myTri.calcArea() << std::endl;
   }

   return 0;
}
```

Shapes In Action



Let's use the heap just because!

```
// main.cpp

int main() {
   std::cout << "Pick a shape (1, 2, or 3):";
   int option = 0;
   std::cin >> option;

   if (option == 1) {
      std::cout << "Making a circle!" << std::endl;
      Circle* myCircle = new Circle(0, 0, 5);
      std::cout << "Area is: " << myCircle->calcArea() << std::endl;
   }
   else if (option == 2) {
      std::cout << "Making a rectangle!" << std::endl;
      Rect* myRect = new Rect(0, 0, 5, 5);
      std::cout << "Area is: " << myRect->calcArea() << std::endl;
   }
   else {
      std::cout << "Making a triangle!" << std::endl;
      Tri* myTri = new Tri(0, 0, 0, 5, 5, 5);
      std::cout << "Area is: " << myTri->calcArea() << std::endl;
   }

   return 0;
}
```

What's different?

```
// main.cpp

int main() {
   std::cout << "Pick a shape (1, 2, or 3):";
   int option = 0;
   std::cin >> option;

   if (option == 1) {
      std::cout << "Making a circle!" << std::endl;
      Circle* myCircle = new Circle(0, 0, 5);
      std::cout << "Area is: " << myCircle->calcArea() << std::endl;
   }
   else if (option == 2) {
      std::cout << "Making a rectangle!" << std::endl;
      Rect* myRect = new Rect(0, 0, 5, 5);
      std::cout << "Area is: " << myRect->calcArea() << std::endl;
   }
   else {
      std::cout << "Making a triangle!" << std::endl;
      Tri* myTri = new Tri(0, 0, 0, 5, 5, 5);
      std::cout << "Area is: " << myTri->calcArea() << std::endl;
   }

   return 0;
}
```

What's the bug in this code?

```
// main.cpp

int main() {
   std::cout << "Pick a shape (1, 2, or 3):";
   int option = 0;
   std::cin >> option;

   if (option == 1) {
      std::cout << "Making a circle!" << std::endl;
      Circle* myCircle = new Circle(0, 0, 5);
      std::cout << "Area is: " << myCircle->calcArea() << std::endl;
   }
   else if (option == 2) {
      std::cout << "Making a rectangle!" << std::endl;
      Rect* myRect = new Rect(0, 0, 5, 5);
      std::cout << "Area is: " << myRect->calcArea() << std::endl;
   }
   else {
      std::cout << "Making a triangle!" << std::endl;
      Tri* myTri = new Tri(0, 0, 0, 5, 5, 5);
      std::cout << "Area is: " << myTri->calcArea() << std::endl;
   }

   return 0;
}
```

Bug fixed!

```
// main.cpp

int main() {
   std::cout << "Pick a shape (1, 2, or 3):";
   int option = 0;
   std::cin >> option;

   if (option == 1) {
      std::cout << "Making a circle!" << std::endl;
      Circle* myCircle = new Circle(0, 0, 5);
      std::cout << "Area is: " << myCircle->calcArea() << std::endl;
      delete myCircle;
   }
   else if (option == 2) {
      std::cout << "Making a rectangle!" << std::endl;
      Rect* myRect = new Rect(0, 0, 5, 5);
      std::cout << "Area is: " << myRect->calcArea() << std::endl;
      delete myRect;
   }
   else {
      std::cout << "Making a triangle!" << std::endl;
      Tri* myTri = new Tri(0, 0, 0, 5, 5, 5);
      std::cout << "Area is: " << myTri->calcArea() << std::endl;
      delete myTri;
   }

   return 0;
}
```

Let's look a little more closely…

```
if (option == 1) {
   std::cout << "Making a circle!" << std::endl;
   Circle* myCircle = new Circle(0, 0, 5);
   std::cout << "Area is: " << myCircle->calcArea() << std::endl;
   delete myCircle;
}
else if (option == 2) {
   std::cout << "Making a rectangle!" << std::endl;
   Rect* myRect = new Rect(0, 0, 5, 5);
   std::cout << "Area is: " << myRect->calcArea() << std::endl;
   delete myRect;
}
else {
   std::cout << "Making a triangle!" << std::endl;
   Tri* myTri = new Tri(0, 0, 0, 5, 5, 5);
   std::cout << "Area is: " << myTri->calcArea() << std::endl;
   delete myTri;
}
```

Defining a “Shape”

- In this example, we have three different types of shapes:
  - A circle
  - A rectangle
  - A triangle

- All three of these shapes support the idea of calculating the area of that shape

- However, our code as written, does not formally define this relationship

Is-a relationship

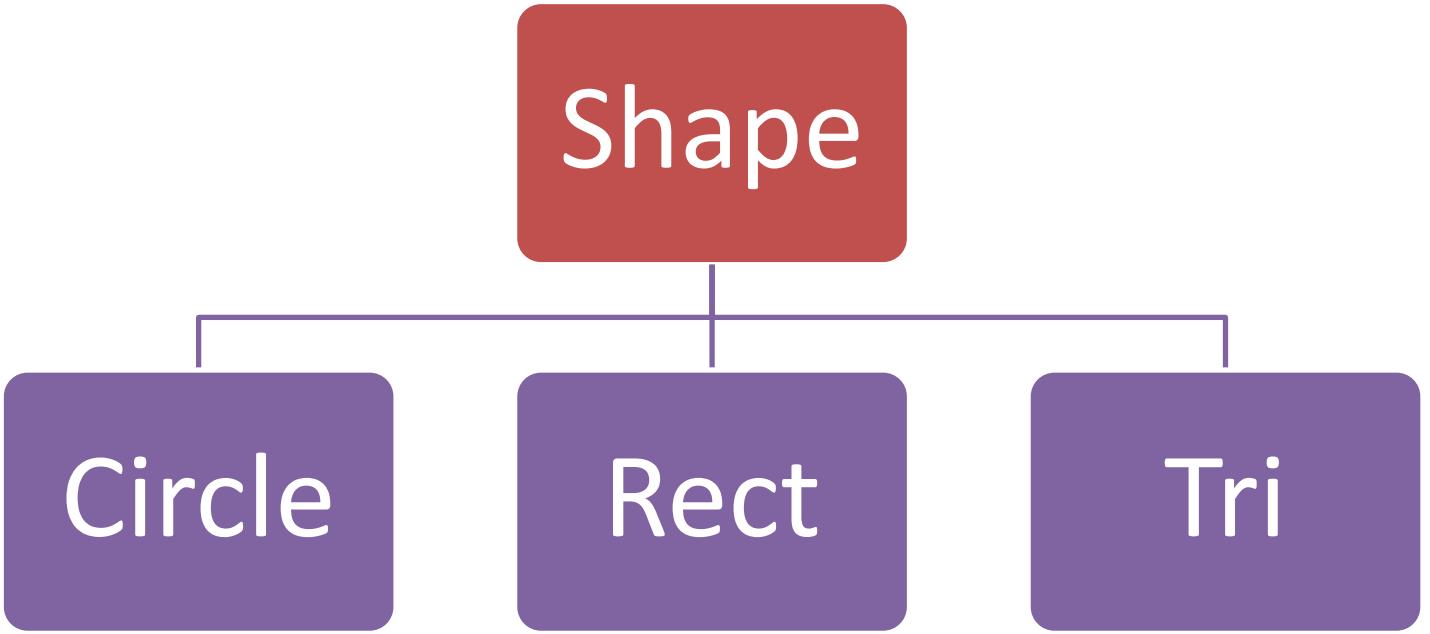
- In an ***is-a relationship***, we say that one class is-a type of another class

- So conceptually, we could make a `Shape` class and say:
  - A `Circle` ***is-a*** type of `Shape`
  - A `Rect` ***is-a*** type of `Shape`
  - A `Tri` ***is-a*** type of `Shape`

- C++ and other object-oriented languages allow us to formally define such a relationship

A family tree of shapes!

- Another way to look at this might be with a hierarchy that looks kind of like a family tree…



The Base Class

- So on the preceding hierarchy, we would say that `Shape` is the ***base class*** (or the class that all the other classes in the hierarchy are descendants of)

- We could also say that `Circle`, `Rect`, and `Tri` are ***derived from*** `Shape` (aka they are children of `Shape`)

Defining what a Shape supports

- Before we write any code, we must decide what every Shape has to support…

- For example, do all Shapes have a radius?
- ***No***. So not all shapes would have a `getRadius` function

- Do all Shapes support calculating an area?
- ***Yes***. So all shapes should have a `calcArea` function.

- In this case, it turns out that the only common functionality between the three shapes is `calcArea` – everything else is different

Object-Oriented Programming

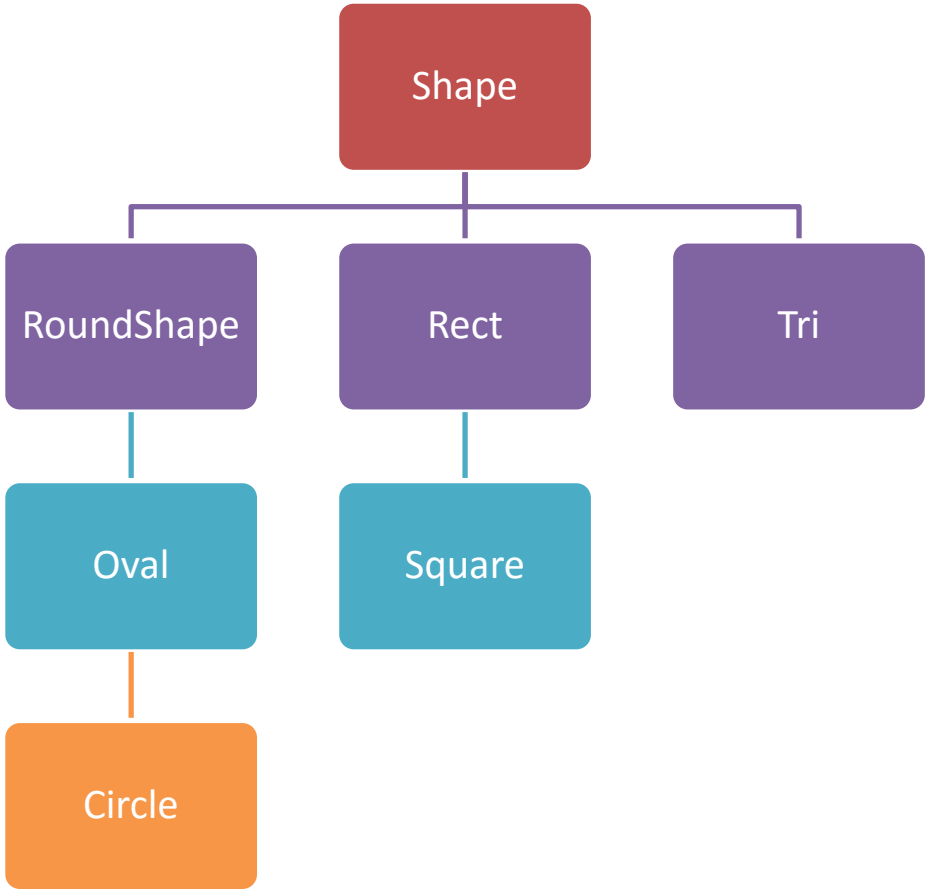
- Using ***is-a*** is a cornerstone of ***Object-Oriented Programming*** (OOP), where the entire program is modeled as different classes.

- So for example, Windows/Mac software programmed in this way might have:
  - A window class instance
  - A menu class instance
  - Several button class instances
  - And so on…

- However, OOP is just one way of programming. It is not the end-all be-all. It's very popular, but that doesn't mean it's always best to use.

More Complex Hierarchy

- Just like a family tree, it's possible to have a multi-level hierarchy:



Watch out!

Poorly Defined Base Classes

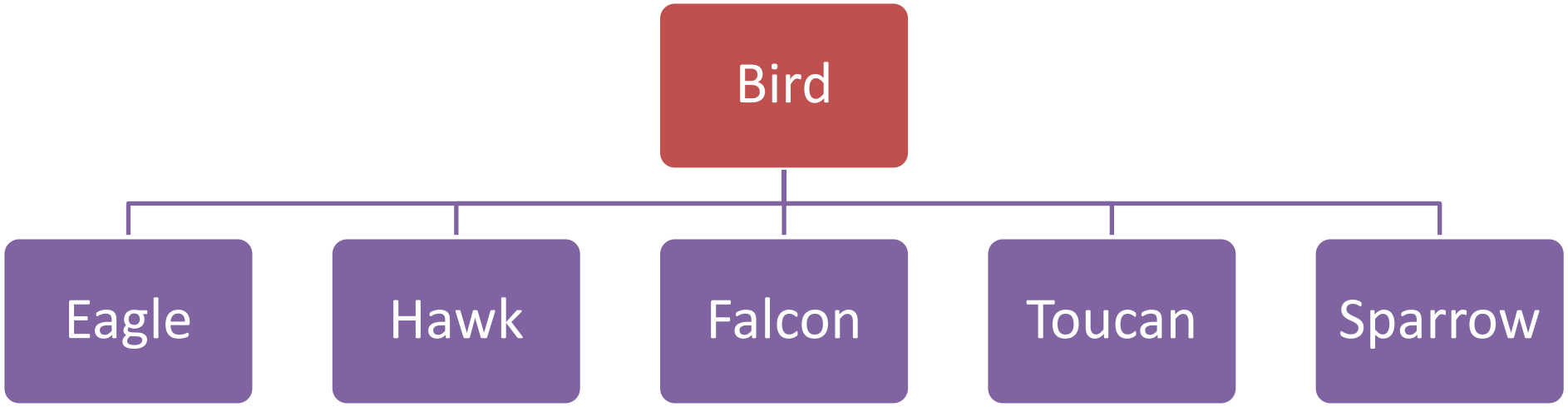
- One big trap to watch out for is adding functions to the base class that all children do not support.

- For example, let's say you make a `Bird` class…

- "Well, birds can fly, so we should say that all children of `Bird` should support a `fly()` function…"

Bird Class

- So maybe you make a hierarchy like this…



- These all fly, so it works well!

What about these?

Lab Practical #22