CIT237
Chapter 15: Inheritance (Part 1)

November 4, 2019

Reminder

- Quiz 5 will be held at the start of class on Wednesday, November 6.
- The material covered on Quiz 5 will be:
  - Lectures of October 21 through October 30.
  - Chapters 13 and 14.

Recall Programming Project 1

- In Programming Project 1, we created a program that simulated some attributes and behaviors of a car:
- Attributes:
  - Current Speed: 35 MPH
  - Current Position (distance from starting point): 750 ft
  - Current State: **Stopped**, **Accelerating**, **Braking**, or **Cruising**
- Behaviors:
  - Accelerate
  - Cruise
  - Brake

Approximating the "Real World"

- There are many other Attributes and Behaviors of a *real* car, which we did not include in our system:
- Other Attributes:
  - Type of engine: gasoline, diesel, electric
  - Type of transmission: automatic, manual
  - Maximum number of passengers: 2, 4, 5, etc.
- Other Behaviors:
  - Steering: straight ahead, turning left, turning right

Generalization and Specialization

- In fact, many of these attributes and behaviors could also apply to other types of “vehicles”, such as trucks, buses, etc.
- We could think of a “Car” as a particular type of “Vehicle”.
- So, if we were interested in designing a C++ class to represent a “Car”, we might also think about designing a more general class called a “Vehicle”

What Is Inheritance?

- Provides a way to create a new class from an existing class.
- The new class is a specialized version of the existing class:
  - A class called “Vehicle” could represent attributes and behaviors which are common to all vehicles.
  - A class called “Car” could be considered a *specialization* of “Vehicle”: it has all attributes and behaviors of “Vehicle”, plus other attributes and behaviors which may be unique to “Car”.

Another Example: Insects

- Our textbook describes the real-world example of Insects.
- Two different types of insects:
  - Bumblebee
  - Grasshopper
- Bumblebees and grasshoppers both have characteristics associated with insects, but each also has their own characteristics:
  - a Bumblebee has the ability to sting.
  - a Grasshopper has the ability to jump.

The "is a" Relationship

- Inheritance establishes an "is a" relationship between classes:
  - A poodle **is a** dog
  - A car **is a** vehicle
  - A flower **is a** plant
  - A football player **is an** athlete
- An object of a specialized class “is a(n)” object of the general class.
- For Example:
  - an `UnderGrad` **is a** `Student`
  - a `Mammal` **is an** `Animal`
- A specialized object has all of the characteristics of the general class, plus its own characteristics.

Different from Aggregation

- With Aggregation, an object of one class is a member of another class.
- The enclosing class "has a" member which is an instance of the enclosed class:
  - A Student "has a" GPA
  - A Car "has a" transmission.

Inheritance: Terminology and Syntax

- Base class: this is the class which is inherited from.
  - Sometimes called the "Parent" class.
  - Sometimes called the "Superclass"
- Derived class: inherits from the base class
  - Sometimes called the "Child" class.
  - Sometimes called the "Subclass"
- Syntax:

```
class Student                    // base class
{
  . . .
};
class UnderGrad : public Student
{                                // derived class
  . . .
};
```

What Does a Child Object Have?

An object of the derived class has:

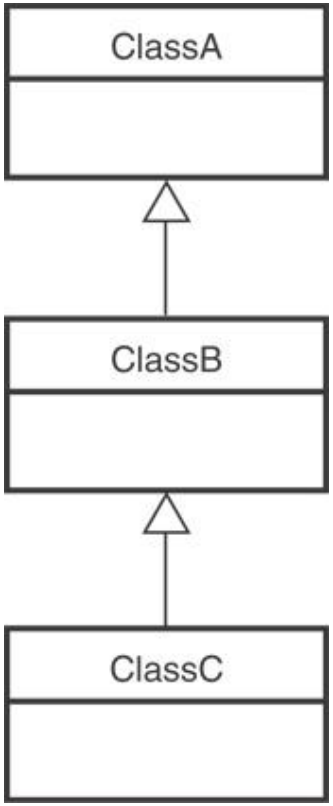
- all members defined in child class
- all members declared in parent class

An object of the derived class can use:

- all `public` members defined in child class
- all `public` members defined in parent class

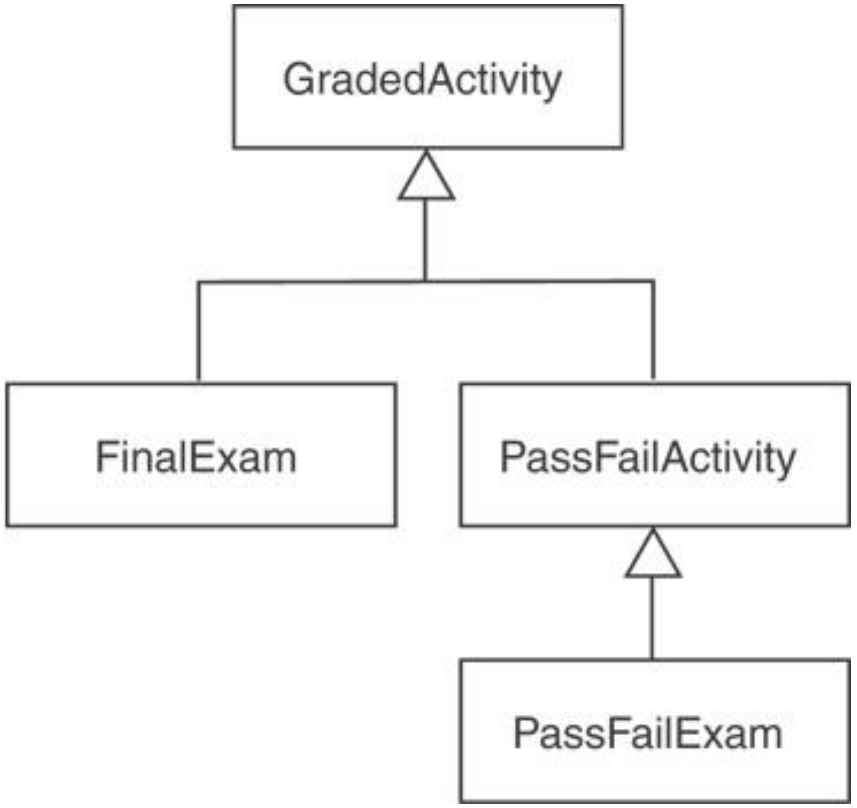
Class Hierarchies

- A base class can be derived from another base class.



Class Hierarchies Example

- Consider the `GradedActivity`, `FinalExam`, `PassFailActivity`, `PassFailExam` hierarchy in Chapter 15:



Recall "`private`" vs. "`public`"

- We said that "`private`" members are accessible only from within the class, but "`public`" members are accessible from outside the class.
- A common design technique is to make data members "`private`" and member functions "`public`".

Protected Members and Class Access

- `protected` member access specification: like `private`, but accessible by objects of derived class
- Class access specification: determines how `private`, `protected`, and `public` members of base class are inherited by the derived class.

Class Access Specifiers

1) `public` – object of derived class can be treated as object of base class (not vice-versa)
2) `protected` – more restrictive than `public`, but allows derived classes to know details of parents
3) `private` – prevents objects of derived class from being treated as objects of base class. (This is the default.)

The textbook discusses several examples regarding Class Access, and whether a object of the child-class can access a member of the parent class.

(See also page 904 in Chapter 15.)

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class

Sample program: SpecialFinalExam

- **GradedActivity** class:
  - Member variable: `double score;`
- **FinalExam** class extends GradedActivity.
  - A FinalExam object ***is a*** GradedActivity object.
  - Member variables:

```
int numQuestions;
double pointsEach;
int numMissed;
```

- The constructors and destructors contain ouput statements, so you can see *when* they run.

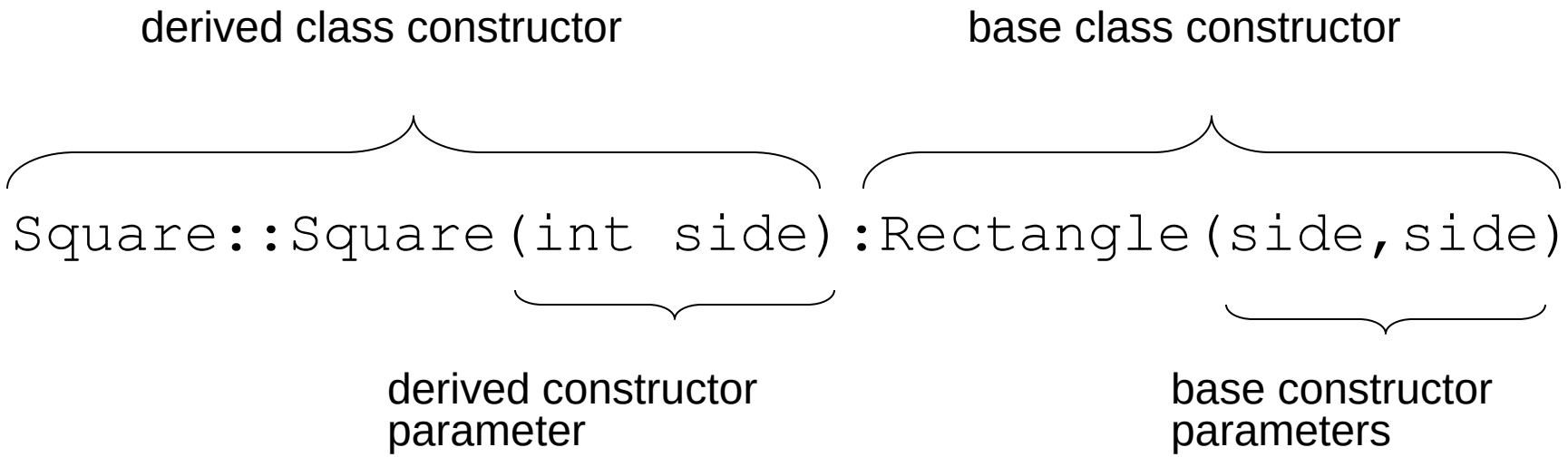
Passing Arguments to Base Class Constructor

- Allows selection between multiple base class constructors
- Specify arguments to base constructor on derived constructor heading:

```
Square::Square(int side) :
                    Rectangle(side, side)
```

- Can also be done with inline constructors
- Must be done if base class has no default constructor

Passing Arguments to Base Class Constructor



Sample program: CubeDemo

- **Rectangle** class:
  - Member variables:

```
double width;
double length;
```

- **Cube** class extends Rectangle.
  - A Cube object ***is a*** Rectangle object.
  - Member variables:

```
double height;
double volume;
```

CubeDemo has two Constructors

- One of the Cube constructors passes arguments to the Rectangle constructor:

```
// Default constructor
Cube() : Rectangle()  {
    height = 0.0; volume = 0.0;
}

// Constructor #2
Cube(double w, double len, double h) :
     Rectangle(w, len) {
     height = h;
     volume = getArea() * h;
}
```

Inheritance: Summary

- Different from Aggregation:
  - Aggregation: "HAS A" relationship
  - Inheritance: "IS A" relationship
- Access Specifiers:

  public

  protected

  private
- Constructors:
  - Order of execution.
  - Passing arguments to base class constructor.