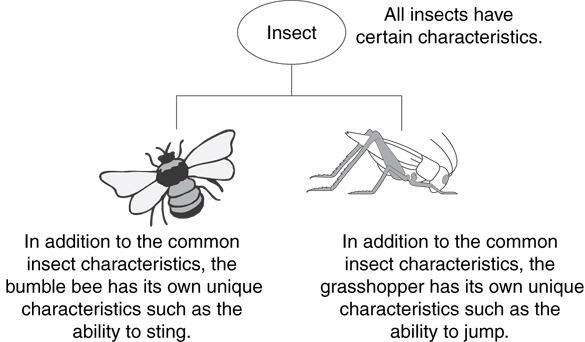
**Inheritance**

**Overview**

Provides a way to create a new class from an existing class

The new class is a **specialized** version of the existing class



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

**Declaring Base/Sub Classes - protected data fields**

* Base class (or parent) – inherited from
* Derived class (or child) – inherits from the base class
* Notation:

class Student // base class

{

Always should use public inheritance

. . .

};

class UnderGrad : public student

{ // derived class

. . .

};

* The UnderGrad class shown above can access all the public members of Student because it inherits them (It cannot access the private data fields of Student)
* Usually want a child class object to be able to access data members that originated with the parent
* What happens if we set the datafields private in the Base Classes (**private data fields are not inherited (accessible) by the children)**

Consider the following inheritance hierarchy:

All getters/setters

void draw();

void print();

x, y,

Circle

radius

All getters/setters

void draw();

r,

ColorCircle

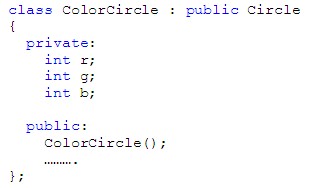
g,

b

void draw();

void print();

ColorFilledCircle



The classes are **initially** declared as follows:

class Circle

{

private:

int x;

int y;

int radius;

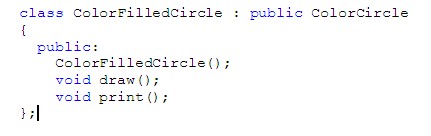
const static double PI;

public:

Circle();

………………

};



Can x,y,radius be accessed in ColorCircle’s methods as shown below?

void ColorCircle::draw()

{

int obj\_no = drawCircle(radius,x,y);

setColor(obj\_no,r,g,b);

}

Syntax errors results. **While having private members is generally a good thing, it creates problems with inheritance!**

We want ColorCircle’s methods to inherit (i.e., access) all of the parent’s (Circle) data fields.

The **protected** specifier is utilized for this purpose. Declaring data fields as protected in the Base Class (instead of private) allows the data fields to be accessed

By all methods of the Base Class

By all methods of the Sub Class (and it’s Sub Classes too!)

Change the access specifier to **protected** in Circle and try to recompile ColorCircle.

ColorFilledCircle has the same problem because ColorCircle has r,g,b as private. ColorFilledCircle cannot access these datafields as shown below:

void ColorFilledCircle::draw()

{

int i = 0;

int obj\_no = 0;

for (i = 0; i < radius; i++)

{

obj\_no = drawCircle(i,x,y);

setColor(obj\_no,r,g,b); //Error!

}

}

Change the declaration for ColorCircle to **protected** and recompile.

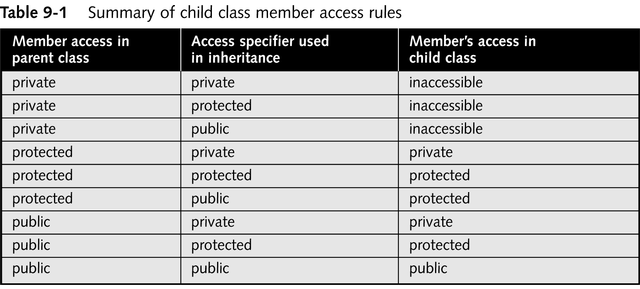
**The Inheritance Specifier**

C++ programmers usually use the public access specifier for inheritance (public access specifier indicates a true isA relationship - Only access specifier we’ll use in this class)

If a derived class uses the public access specifier, then the following statements are true:

* + Base class members that are public remain public in the derived class
  + Base class members that are protected remain protected in the derived class
  + Base class members that are private are inaccessible in the derived class

Can also use protected or private for the inheritance specifier (rarely used). The following table shows the relationship between base/sub class and data fields based on the inheritance specifier:



to be used by class member functions and by derived classes, but not by other parts o

**Overriding Parent’s Methods**

The child class also can have its own methods.

Methods in the child class can have names/signatures that are **identical** to the methods in the parent class. This is called **overriding**. Examples would be **draw** found in ColorCircle.

Precedence is given to the method in the child class. If not found, the parent class is searched for the method. Consider draw and print in ColorCircle. Which one is overridden? Is print implemented in ColorCircle? What happens when print is invoked on a ColorCircle object?

**Calling a Base Class Method within a Sub Class method (same name/signature)**

Use the following notation:

ParentClass::methodName()

When calling a Base class method within a subclass method of the same name. Consider the following example in ColorFilledCircle:

void ColorFilledCircle::print()

{

Circle::print();

gout << setPos(x-radius,y) << "Filled Circle" << endg;

}

The first line invokes the print method of Circle. What happens if I leave off the Circle:: ?