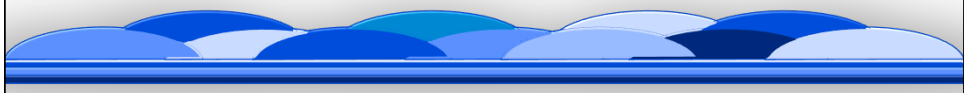
Topic 15 – Inheritance (OOP)



Inheritance

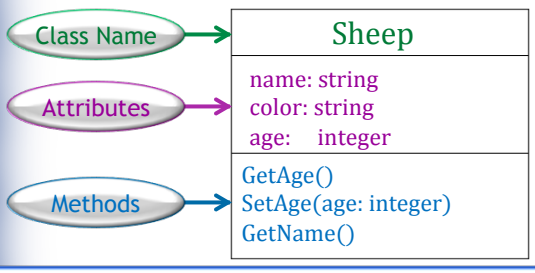
Relationship between Object Types

- Many object types have some form of relationship
  - complex object types can be built by combining simpler object types
    - A car *is composed of* engine, transmission, wheels, doors, seats, etc.
    - A computer *is composed of* external storage(hard drive), memory, motherboard, etc...
  - Also, some object types share many common features
    - They have a large subset of features that are shared and a few that differentiate them
    - Pigs, sheep, dogs, cats are all animals
      - **They have weight, height, value, etc…**
      - **Some have fur some don't**

Designing OOP programs using UML

UML = Unified Modeling Language

- Tool used to diagram or design OOP programs
- UML consists of many diagramming tools
  - *Class Diagrams* allow us to generally define our relationships between objects
  - Fill in the details as we go along
- We represent a class using:



We also use different lines between them to better define those relationships

We'll discuss those as we go along

Class Diagrams help us in 2 ways
1 - to think about our OOP design - classes & relationships
2 - to communicate our ideas with others

Relationships between objects

- We find that many classes are related
  - They can be composed of different classes
  - They can share common behavior
- We define different relationships between objects to reduce complexity of our code
- We will discuss
  - Composition
  - Inheritance
  - Polymorphism

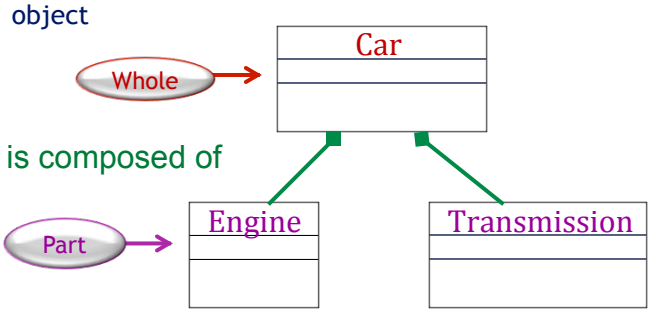
Composition (or Aggregation)

Aggregation

- how we show the "*is composed of*" relationship
- We use this line
- Parts are joined to create a whole

We use Aggregation when we can say an object of this class

- "has a", "contains a", "a part of", "is composed of" another object



Composition

- In composition, one or more attribute(s) of a class are objects of another class type

For example

engine and transmission are objects of *engineType* and *transmType* class types

```
class Car
{
    public:
        …                          // car methods
    private:
        engineType engine;         // car's engine - class object
        transmType transmission;   // car's transmission - class object
        string color;              // car's color
};
```

Composition (cont'd)

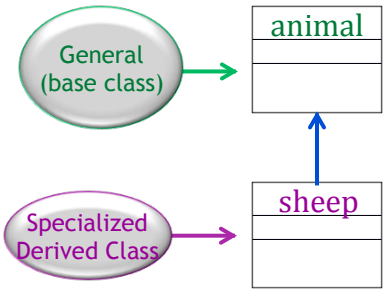
- *attribute-objects* of a class are constructed (that is, initialized) in the order they are declared
  - *Default constructor* for each attribute-object
    - automatically executed when a class object are constructed
    - constructed first (before the containing class objects are constructed)
  - For the Car class, objects will be created in the following order:
    - engine (object of engineType class)
    - transmission (object of transmType class)
    - instance of Car class

Inheritance

- The inheritance relationship allows us to define subclasses that are extensions of a superclass
- With inheritance a class can acquire the properties of another class
  - For example,
    - Shape can be a superclass (base class)
    - Circle or Rectangle can be subclassses (derived class)
  - Shape will have some methods and attributes that apply to both
  - Circle and Rectangle can be extended
    - By adding methods or attributes or overriding methods

Inheritance

- *Inheritance* is a relationship between a general class (base class) and a specialized class (derived class)
- *Derived class* (child) *inherits* features from a base class (parent) and can add extra features
- Object types feature relationships such as
  - "is a", "is a kind of", "is like a"



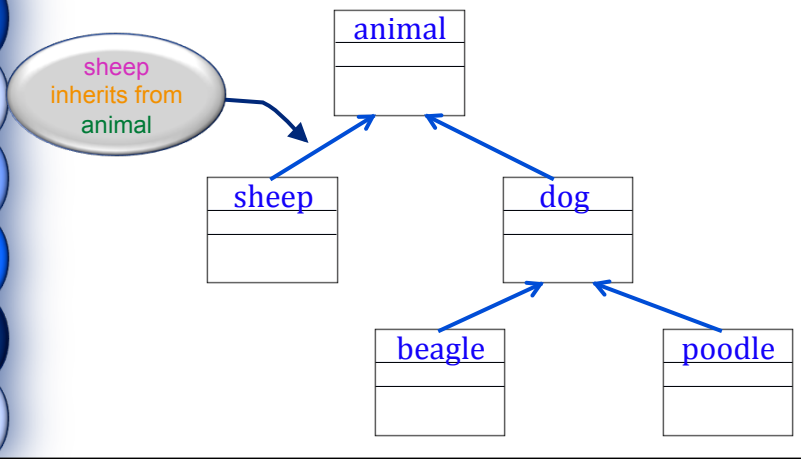
Inheritance

- Example of Inheritance relationship
  - Every sheep and pig is an animal
  - Every student and teacher is a person
- Inheritance allows creation of new classes from existing classes
  - *Derived classes:* new classes created from the existing class
  - *Base class:* the original class
- Derived class inherits the properties of its base classes
- Inheritance helps reduce software complexity

Defining these relationships allows us to share code between classes and provide services that can be used in multiple classes

Inheritance

- Inheritance can be viewed as a tree-like, or hierarchical, structure between the base class and its derived classes



Defining a class (review)

```
Syntax
class className
{
        public:
            public parts go here
        protected:
            protected parts go here
        private:
            private parts go here
};
```

- For attributes

```
Syntax
datatype attributeName;
```

- Note: the data type can also be a class
- For methods

```
Syntax
returnDatatype MethodName (datatype name, datatype name) const;
```

Reviewing Class Data Categories

3 categories for defining data within a class

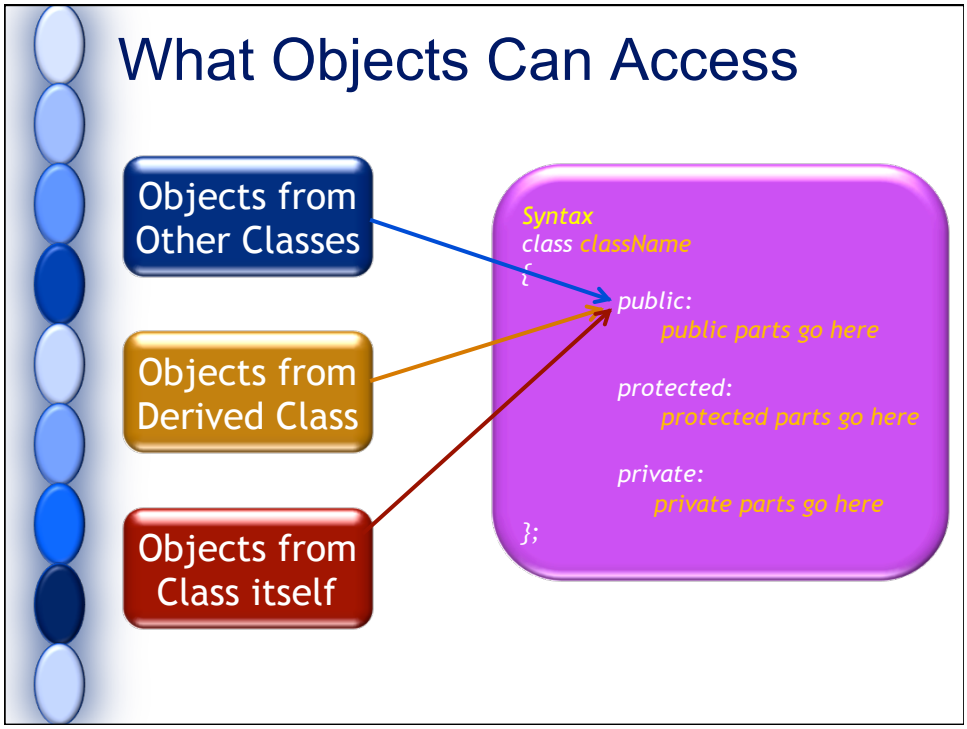
So far we have talked about 2

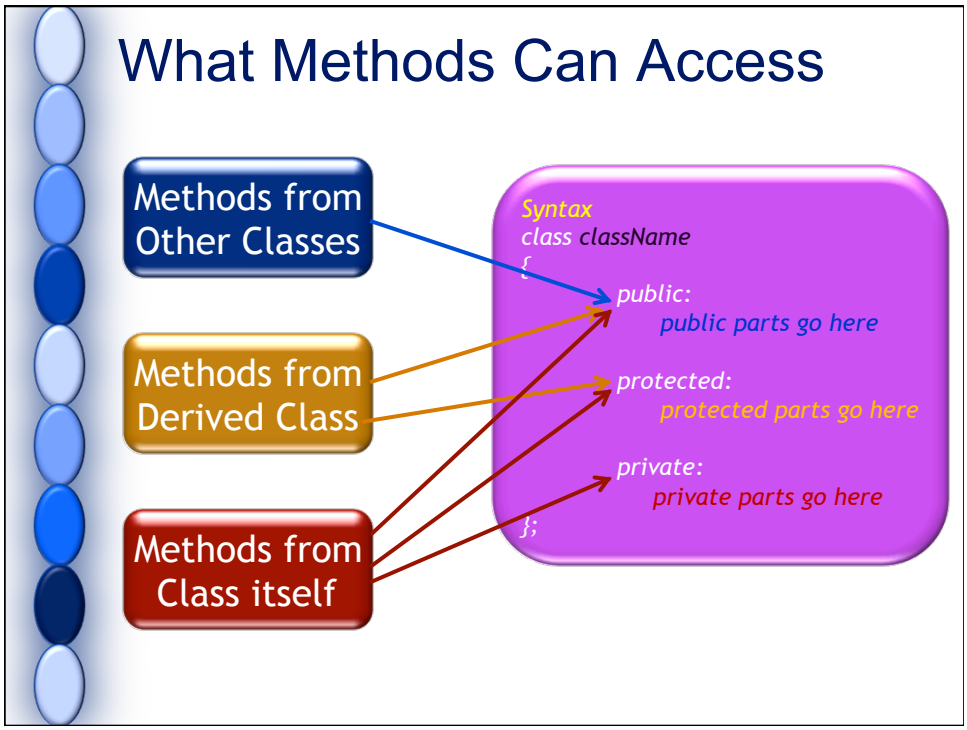
- *Public Data* → available to any **object** that uses the class and the class itself
  - Anyone using this class sees *public data* including any derived class
- *Private Data* → available only to class methods within that class
  - Derived classes can not see *private data*

- To support inheritance a ***third category*** is necessary
  - *Protected Data* is available to any derived class methods and to the class itself → but not to available to any **object**

What can access what?

- **Objects**
  - Can access anything defined as PUBLIC within its own class
  - Can access anything defined as PUBLIC in the base class (if it has a base class)
  - Can't access anything defined as **PRIVATE** or **PROTECTED** (in any class)
- **Methods**
  - Can access anything defined within its own class
  - Can access anything defined as PUBLIC or **PROTECTED** in the base class (if it has a base class)
  - Can't access anything defined as **PRIVATE** (in the base class)


What Objects Can Access
Objects from Other Classes
Objects from Derived Class
Objects from Class itself
Syntax
class className
{
public:
public parts go here
protected:
protected parts go here
private:
private parts go here
};


What Methods Can Access
Methods from Other Classes
Methods from Derived Class
Methods from Class itself
Syntax
class className
{
public:
public parts go here
protected:
protected parts go here
private:
private parts go here
};

Using Inheritance

- Defining a derived class

Syntax

```
class className: memberAccessSpecifier baseClassName
{
        public:
            New and changed member functions
        protected:
            protected parts go here
        private:
            Additional data members
};
```

- *memberAccessSpecifier* can be:
  - public, protected or private (default)
  - We'll talk about this later FOR NOW JUST USE PUBLIC!
- Assuming an animal base class, a derived pig class could be defined as

```
class Pig: public Animal { … };
```

Inheritance Rules

If it is defined in the base class as:

public

- Derived class methods can access it
- Base class and Derived class objects can access it

protected

- Derived class methods can access it
- No objects can access it!

private ➔ It is *private* to the base class

- Only methods in the base class can access it
- Derived class methods and objects cannot access it
- No objects can access it!

Inheritance Rules (cont'd)

- All attributes of the base class are also attributes of the derived class
- Derived class can include additional *attributes* and/or *methods*
  - thus extending the base class
- Derived class can redefine *public methods* of the base class
  - Provide a different set of code for the same method
  - Applies only to the objects of the derived class
  - We'll get to this later!

Constructor in Derived Class

- Recall: *constructor* is a special method that allows an instance of the class to be created
- A Derived class *constructor* cannot directly access private attributes of the base class
  - It can directly initialize only public attributes of the base class
- When a derived object is declared, it must execute the base class constructor
  - The *default constructor* of the base class is automatically executed when derived class *default constructor* is executed

Destructor in Derived Class

- Recall: *destructor* is a special method that allows an instance of the class to be terminated
  - it is used for releasing resources (de-allocating memory, closing files, etc.)
- When a derived class object goes out of scope
  - Automatically invokes its destructor
- When the *destructor* of the derived class executes
  - Automatically invokes the *destructor* of the base class

Example

Lets say I have a base class called shape

```
class Shape
{
   public:
      Shape();
      ~Shape();
      void setWidth(int w);
      void setHeight(int h);

   protected:
      int width;
      int height;
};

// constructor and destructor here
void Shape::setWidth(int w)
{
   width = w;
}


void Shape::setHeight(int h)
{
   height = h;
}
```

These can be protected or private in the derived class

- Now I can derive a class - say Rectangle

```
class Rectangle: public Shape
{
   public:
      int getArea() const;
};

int Rectangle::getArea()
{
  return (width * height);
}
```

Base Class

Rectangle still has SetWidth & SetHeight, But also has GetArea

- And here's how I can use it

```
int main()
{
   Rectangle Rect;
   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;
   return 0;
}
```

Exercise

- Let's assume an Animal class as our base class. We need to use inheritance to create a derived Sheep class that in addition to the basic animal features defined in the base class should include:
  - sheep wool type
    - There are four different types of wool that we want to represent: long, medium, fine and carpet
  - sheep wool color
- How would you create the Sheep class?

```
class Animal                        // base class
{
public:

        Animal();                   // constructor
        ~Animal();                  // destructor
         /******************
          ***  MUTATORS  ***
          ******************/
         void SetInitialValues(string sheepName,
                               int    sheepAge);
         void ChangeAge(int animalAge);
        /******************
         ***  ACCESSORS  ***
         ******************/
         string GetName() const;          // return the animal's name
         int    GetAge () const;          // return the animal's age
         void   Display () const;         // print all animal's details

private:
         string name;    // animal's name
         int    age;     // animal's age in years
```

Derived Sheep Class

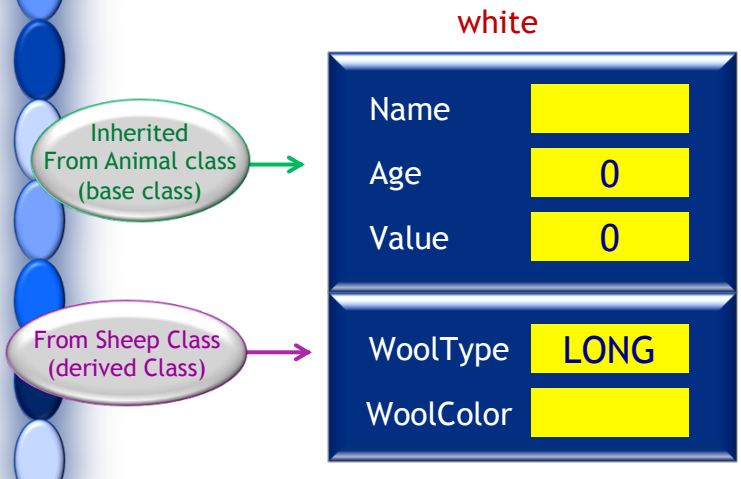
- We will use public inheritance allowing instances of Sheep class to access methods from the Animal class
- Derived Sheep class require two attributes representing the sheep wool type and color
  - We need to create an enumerated type variable to represent the different wool types

    enum WoolType {LONG, MEDIUM, FINE, CARPET};
  - Wool color can be represented by a string
- We need to create methods to handle the new attributes

Sheep Instance

- Creating a Sheep instance named white

  Sheep white;



Redefining Base Class Methods

In some cases, to better design the derived class we need to redefine the original *method* in the base class

- It is replaced for a new *method* implemented in the derived class

- To redefine a base class public *method*
  - We define a method in the derived class that has the same signature as a method in the base class:
    - same method name and
    - the same number/types of parameters

Write a method in the sheep class to redefine Display in the animal class

- If derived class redefines a public *method* of the base class, then the derived object will call the redefined method

Syntax
baseClassName::MethodName(name, name, …); or
returnvalue = baseClassName::MethodName(name, name, …);

Updating the Sheep Class

- Assume that we want to redefine the Display() method in the Animal Class

Back to Member Access Specifiers

- Defining a derived class

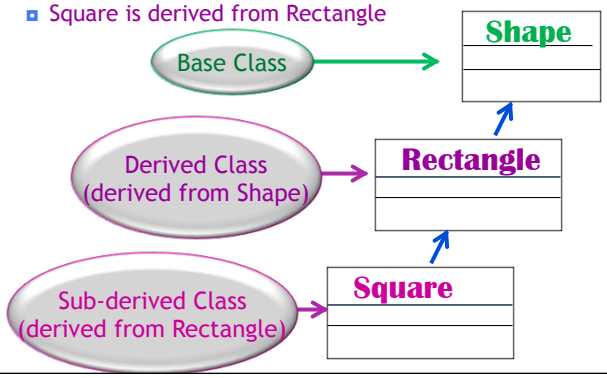
```
Syntax
class className: memberAccessSpecifier baseClassName
{
        public:
            New and changed member functions
        protected:
            protected parts go here
        private:
            Additional data members
};
```

- *memberAccessSpecifier* can be:
  - public, protected or private (default)
- Assuming an animal base class, a derived pig class could be defined as

```
class Pig: public    Animal { … };
class Pig: protected Animal { … };
class Pig: private   Animal { … };
```

Member Access Specifiers

- The Member Access Specifiers affect
  - What derived (and sub-derived) objects can access in a base class
  - What methods from a sub-derived class can access from the base class
- Let's say we have a shape as a base class
  - Rectangle is derive from Shape and
    - Square is derived from Rectangle



RECALL: What can access what?

Objects

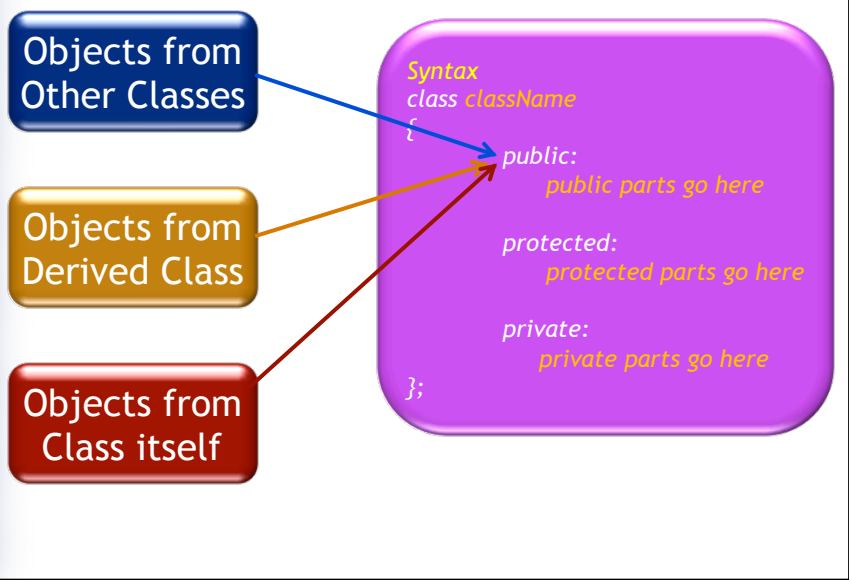
- Can access anything defined as PUBLIC within its own class
- Can access anything defined as PUBLIC in the base class (if it has a base class)
- Can't access anything defined as **PRIVATE** or **PROTECTED** (in any class)

Methods

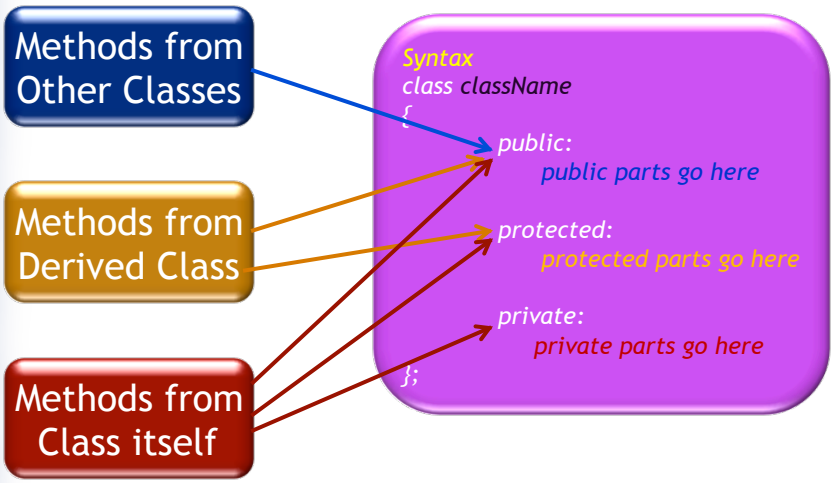
- Can access anything in defined within its own class
- Can access anything defined as PUBLIC or **PROTECTED** in the base class (if it has a base class)
- Can't access anything defined as **PRIVATE** (in the base class)

NOTE: *Member Access Specifiers do not change these*

What Objects Can Access



What Methods Can Access



Access Specifiers – Affect on Objects

The Member Access Specifiers affects what derived (and sub-derived) objects can access in a base class

Public:

```
class RectanglePublic: public Shape;
```

Objects of the Derived class can access:

- Anything defined as public in the base class
- (i.e. An object of type RectanglePublic can access anything public in Shape)

Protected:

```
class RectangleProtected: protected Shape;
```

Objects of the Derived class can't access anything in the base class:

- (i.e. An object of type RectangleProtected can't access anything in Shape)

Private:

```
class RectanglePrivate: private Shape;
```

Objects of the Derived class can't access anything in the base class:

- (i.e. An object of type RectanglePrivate can't access anything in Shape)

NOTE: *With respect to an object of a derived class protected and private are the same*

```
#include <string>
using namespace std;
class Shape
{
public:
   Shape();
   ~Shape();
   void   SetShapeWidthPub(int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;


protected:
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;


private:
   int shapeWidthPriv;
};
```

Public Methods/Attributes:
Accessible to:
- Any Shape Method
- Any Shape Object

- Any Derived Method
- Derived Objects can only access this if the access specifier is PUBLIC

Protected Methods/Attributes:
Accessible to:
- Any Shape Method
- Any Derived Method

NOT ACCESSIBLE to ANY OBJECTS

Private Methods/Attributes:
Accessible to:
- Any Shape Method

NOT ACCESSIBLE to ANY OBJECTS

Rectangle Class

```
#include "Shape.h"

class RectangleXXX: public Shape
{
public:
  RectanglePub();
 ~RectanglePub();

 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

private:
   int rectAreaPriv;


};
```

The Member Access Specifier doesn't affect the derived classes - it affects the objects and sub-derived classes.

Let's create 3 rectangle classes to demonstrate inheritance using the 3 different Member Access Specifiers

All 3 classes will be identical except the names:

RectanglePublic
RectangleProtected
RectanglePrivate

The implementations of the methods will also be identical.

What can methods from a derived class access?

```
class Shape
{
public:
   Shape();
   ~Shape();
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;


protected:
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;


private:
   int shapeWidthPriv;
};
```

```
class RectanglePublic   :public    Shape
class RectangleProtected: protected Shape
class RectanglePrivate  : private   Shape
```

The Access Specifiers **don't have any effect** the methods that any of the rectangles can access

RectangleXXX Methods can access:

What can methods from a derived class access?

```
class Shape
{
public:


   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;


protected:
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;




};
```

```
class RectanglePublic   :public    Shape
class RectangleProtected: protected Shape
class RectanglePrivate  : private   Shape
```

The Access Specifiers **don't have any effect** the methods that any of the rectangles can access

RectangleXXX Methods can access:

- anything Public in Shape
- anything Protected in Shape

Public Access Specifier

```
class Shape
{
public:
   Shape();
   ~Shape();
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;



protected:
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;



private:
   int shapeWidthPriv;
};
```

class RectanglePublic: public Shape

*Whatever is **PUBLIC** in the base*
*is **PUBLIC** to the derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

A *RectanglePublic* Object can access:

Public Access Specifier - Objects

```
class Shape
{
public:



   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;









};
```

class RectanglePublic: public Shape

*Whatever is **PUBLIC** in the base*
*is **PUBLIC** to the derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

A *RectanglePublic* Object can access:
- anything **Public** in Shape

For example:

```
RectanglePublic myPubRect;

myPubRect.SetShapeWidthPub(2);
myPubRect.shapeNamePub = "My Awesome"
                         " Rectangle";
```

Public Access Specifier

*Whatever is **PUBLIC** in the base is **PUBLIC** to the derived*

```
class RectanglePublic: public Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Public Access Specifier

*Whatever is **PROTECTED** in the base is **PROTECTED** to the derived*

```
class RectanglePublic: public Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Public Access Specifier

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

```
class RectanglePublic: public Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor
 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

protected:
 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Remember:
Objects can access anything Public in their own class
Methods can access anything in their own class

Protected & Private Access Specifier

```
class Shape
{
public:
  Shape();
  ~Shape();
  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;

protected:
  int GetShapeWidthProt()  const;
  int GetShapeHeightProt() const;
  int shapeHeightProt;

private:
  int shapeWidthPriv;
};
```

class RectangleProtected : protected Shape

*Whatever is **PUBLIC** in the base is **PROTECTED** to the derived*

*Whatever is **PROTECTED** in the base is **PROTECTED** to the derived*

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

class RectanglePrivate : private Shape

*Whatever is **PUBLIC** in the base is **PRIVATE** to the derived*

*Whatever is **PROTECTED** in the base is **PRIVATE** to the derived*

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

A *RectangleProtected* or *RectanglePrivate* Object

Protected & Private Access Specifier

```
class Shape
{




};
```

class RectangleProtected : protected Shape

*Whatever is **PUBLIC** in the base is **PROTECTED** to the derived*

*Whatever is **PROTECTED** in the base is **PROTECTED** to the derived*

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

class RectanglePrivate : private Shape

*Whatever is **PUBLIC** in the base is **PRIVATE** to the derived*

*Whatever is **PROTECTED** in the base is **PRIVATE** to the derived*

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

A *RectangleProtected* or *RectanglePrivate* **Object** can't access *anything* in Shape

Protected Access Speci

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

```
class RectangleProtected: protected Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);




private:
   int rectAreaPriv;
};
```

```
class Shape
{




private:
   int shapeWidthPriv;
};
```

Remember:

***Objects** can access anything Public in their own class*

***Methods** can access anything in their own class*

Protected Access Specifier

```
class RectangleProtected: protected Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Protected Access Specifier

*Whatever is **PUBLIC** in the base is **PROTECTED** to the derived*

```
class RectangleProtected: protected Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

void   SetShapeWidthPub (int newWidth);
void   SetShapeHeightPub(int newHeight);
string shapeNamePub;

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor

void   SetShapeWidthPub (int newWidth);
void   SetShapeHeightPub(int newHeight);
string shapeNamePub;

protected:

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Protected Access Specifier

*Whatever is **PROTECTED** in the base is **PROTECTED** to the derived*

```
class RectangleProtected: protected Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

protected:

 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Protected Access Specifier

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

```
class RectangleProtected: protected Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

```
class Shape
{
public:
 // constructor & destructor

 void   SetShapeWidthPub (int newWidth);
 void   SetShapeHeightPub(int newHeight);
 string shapeNamePub;

protected:

 int GetShapeWidthProt()  const;
 int GetShapeHeightProt() const;
 int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Private Access Specifier

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

```
class RectanglePrivate: private Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;




};
```

```
class Shape
{




private:
   int shapeWidthPriv;
};
```

Remember:

Objects can access anything Public in their own class

Methods can access anything in their own class

Private Access Specifier

```
class RectanglePrivate: private Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;









};
```

```
class Shape
{
public:
 // constructor & destructor

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;


protected:

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;



private:
   int shapeWidthPriv;
};
```

Private Access Specifier

*Whatever is **PUBLIC** in the base is **PRIVATE** to the derived*

```
class RectanglePrivate: private Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

};
```

```
class Shape
{
public:
 // constructor & destructor
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Remember:
Objects can access anything Public in their own class
Methods can access anything in their own class

Private Access Specifier

*Whatever is **PROTECTED** in the base is **PRIVATE** to the derived*

```
class RectanglePrivate: private Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

};
```

```
class Shape
{
public:
 // constructor & destructor
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Remember:
Objects can access anything Public in their own class
Methods can access anything in their own class

Private Access Specifier

*Whatever is **PRIVATE** in the base is **UNACCESSIBLE** to the derived*

```
class RectanglePrivate: private Shape
{
public:
 // constructor & destructor
 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;

protected:
 void SetRectAreaProt(int rectHeight,
                      int rectWidth);

private:
   int rectAreaPriv;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

};
```

```
class Shape
{
public:
 // constructor & destructor
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int shapeWidthPriv;
};
```

Remember:

Objects can access anything Public in their own class

Methods can access anything in their own class

Rectangle Class & Square Class

```
#include "Shape.h"

class RectangleXXX: public Shape
{
public:
  RectanglePub();
 ~RectanglePub();

 void SetRectDimensionsPub(int rectHeight
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                  int &rectHeight,
                  int &rectWidth) const;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

private:
   int rectAreaPriv;

};
```

Let's say we declare 3 classes that inherit from each of our rectangle classes

class SquarePublic : public RectanglePublic

class SquareProtected : public RectangleProtected

class SquarePrivate : public RectanglePrivate

The SquareXXX methods can access:

Rectangle Class & Square Class

```
#include "Shape.h"

class RectangleXXX: public Shape
{
public:



 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                 int &rectHeight,
                 int &rectWidth) const;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);




};
```

Let's say we declare 3 classes that inherit from each of our rectangle classes

```
class SquarePublic    : public RectanglePublic
class SquareProtected : public RectangleProtected
class SquarePrivate   : public RectanglePrivate
```

The SquareXXX methods can access:

Anything PUBLIC or PROTECTED in the respective RectangleXXX classes because the access specifier for rectangle is public

A SquareXXX object can access:

Rectangle Class & Square Class

```
#include "Shape.h"

class RectangleXXX: public Shape
{
public:



 void SetRectDimensionsPub(int rectHeight,
                           int rectWidth);
 int  GetRectAreaPub();
 void GetRectDimensionsPub(
                 int &rectHeight,
                 int &rectWidth) const;
```

Let's say we declare 3 classes that inherit from each of our rectangle classes

```
class SquarePublic    : public RectanglePublic
class SquareProtected : public RectangleProtected
class SquarePrivate   : public RectanglePrivate
```

The SquareXXX methods can access:

Anything PUBLIC or PROTECTED in the respective RectangleXXX classes because the access specifier for rectangle is public

A SquareXXX object can access:

Anything PUBLIC in the respective RectangleXXX class because the access specifier for rectangle is public

This relationship between the derived class and sub-derived classes is not affected by Access Specifiers definded for the base class in the sub-derived class.

NOTE: *The Access Specifiers defined for Shape when inherited by Rectangle affect how Square's methods and objects can access the Shape's methods.*

Base Class & Sub-derived Class – PUBLIC Member Access Specifier

```
class Shape
{
public:
  Shape();
  ~Shape();
  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;



protected:
  int GetShapeWidthProt()  const;
  int GetShapeHeightProt() const;
  int shapeHeightProt;



private:
  int shapeWidthPriv;
};
```

class RectanglePublic : public Shape
class SquarePublic : public RectanglePublic

*Whatever is **PUBLIC** in the base*
*is **PUBLIC** to the derived*
*is **PUBLIC** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquarePublic **method*** can access:

Base Class & Sub-derived Class – PUBLIC Member Access Specifier

```
class Shape
{
public:


  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;



protected:
  int GetShapeWidthProt()  const;
  int GetShapeHeightProt() const;
  int shapeHeightProt;









};
```

class RectanglePublic : public Shape
class SquarePublic : public RectanglePublic

*Whatever is **PUBLIC** in the base*
*is **PUBLIC** to the derived*
*is **PUBLIC** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquarePublic **method*** can access:
- anything **Public** in Shape
- anything **Protected** in Shape

A *SquarePublic **object*** can access:

Base Class & Sub-derived Class – **PUBLIC** Member Access Specifier

```
class Shape
{
public:


  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;









};
```

class RectanglePublic : public Shape
class SquarePublic : public RectanglePublic

*Whatever is **PUBLIC** in the base*
*is **PUBLIC** to the derived*
*is **PUBLIC** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquarePublic **method*** can access:
- anything **Public** in Shape
- anything **Protected** in Shape

A *SquarePublic **object*** can access:
- anything **Public** in Shape

Public Access Specifier

```
class SquarePublic : public RectanglePublic
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:




private:
};
```

*Whatever is **PUBLIC** in the base*
*is **PUBLIC** to the derived*
*is **PUBLIC** to the sub-derived*

```
class RectanglePublic: public Shape
{


   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:

   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

Public Access Specifier

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

```
class SquarePublic : public RectanglePublic
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
};
```

```
class RectanglePublic: public Shape
{
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

Public Access Specifier

*Whatever is **PRIVATE** in the derived*
*is **UNACCESSIBLE** to the sub-derived*

```
class SquarePublic : public RectanglePublic
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
};
```

```
class RectanglePublic: public Shape
{
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;

   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;

protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

private:
   int rectAreaPriv;
};
```

Base Class & Sub-derived Class – PROTECTED Member Access Specifier

```
class Shape
{
public:
  Shape();
  ~Shape();
  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;



protected:
  int GetShapeWidthProt()  const;
  int GetShapeHeightProt() const;
  int shapeHeightProt;



private:
  int shapeWidthPriv;
};
```

class RectangleProtected : protected Shape
class SquareProtected : public RectangleProtected

*Whatever is **PUBLIC** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquareProtected **method*** can access:

Base Class & Sub-derived Class – PROTECTED Member Access Specifier

```
class Shape
{
public:


  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;



protected:
  int GetShapeWidthProt()  const;
  int GetShapeHeightProt() const;
  int shapeHeightProt;









};
```

class RectangleProtected : protected Shape
class SquareProtected : public RectangleProtected

*Whatever is **PUBLIC** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquareProtected **method*** can access:
- anything Public in Shape
- anything Protected in Shape

A *SquareProtected **object*** can access:

Base Class & Sub-derived Class – PROTECTED Member Access Specifier

```
class Shape
{




};
```

class RectangleProtected : protected Shape
class SquareProtected : public RectangleProtected

*Whatever is **PUBLIC** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquareProtected **method*** can access:
- anything Public in Shape
- anything Protected in Shape

A *SquareProtected **object*** can access:
- nothing in Shape

Protected Access Specifier

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;




protected:




private:
};
```

```
class RectangleProtected: protected Shape
{
 public:

    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

 protected:

    void SetRectAreaProt(int rectHeight,
                         int rectWidth);
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

 private:
   int rectAreaPriv;
};
```

Protected Access Specifi

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;

protected:

private:
};
```

This was public in Shape
But it is protected in Rectangle

*Whatever is **PUBLIC** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

```
class RectangleProtected: protected Shape
{
 public:
    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
    void SetRectAreaProt(int rectHeight,
                         int rectWidth);
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

 private:
   int rectAreaPriv;
};
```


Protected Access Specifi

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;

protected:

private:
};
```


This is protected in Shape
It is also protected in Rectangle

*Whatever is **PROTECTED** in the base*
*is **PROTECTED** to the derived*
*is **PROTECTED** to the sub-derived*

```
class RectangleProtected: protected Shape
{
 public:
    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
    void SetRectAreaProt(int rectHeight,
                         int rectWidth);
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

 private:
   int rectAreaPriv;
};
```

Protected Access Specifi

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;

protected:

private:
};
```

Anything Private in Shape
is UNACCESSIBLE to Rectangle
** so it won't be shown here **

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

```
class RectangleProtected: protected Shape
{
 public:
    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
    void SetRectAreaProt(int rectHeight,
                         int rectWidth);
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

 private:
   int rectAreaPriv;
};
```

Protected Access Specifi

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;
protected:

private:
};
```

The access specifier for Rectangle is Public SO
*whatever is **PUBLIC** in Rectangle*
*is **PUBLIC** in Square*

```
class RectangleProtected: protected Shape
{
 public:
    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
    void SetRectAreaProt(int rectHeight,
                         int rectWidth);
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

 private:
   int rectAreaPriv;
};
```

Protected Access Specifi

The access specifier for Rectangle is Public SO
- *whatever is **PROTECTED** in Rectangle*
- *is **PROTECTED** in Square*

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;
protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
private:
};
```

```
class RectangleProtected: protected Shape
{
 public:
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;
 protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
 private:
   int rectAreaPriv;
};
```

Protected Access Specifi

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

```
class SquareProtected
: public RectangleProtected
{
public:
  // constructor & destructor
   void SetSquSideLengthPub(int sideLength);
   int  GetSquAreaPub();
   int  GetSquSideLengthPub() const;
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;
protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
private:
};
```

```
class RectangleProtected: protected Shape
{
 public:
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                     int &rectHeight,
                     int &rectWidth) const;
 protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
 private:
   int rectAreaPriv;
};
```

Base Class & Sub-derived Class – PRIVATE Member Access Specifier

```
class Shape
{
public:
  Shape();
  ~Shape();
  void   SetShapeWidthPub (int newWidth);
  void   SetShapeHeightPub(int newHeight);
  string shapeNamePub;



protected:
  int GetShapeWidthProt()  const;
  int GetShapeHeightProt() const;
  int shapeHeightProt;



private:
  int shapeWidthPriv;
};
```

class RectanglePrivate : **private** Shape
class SquarePrivate : public RectanglePrivate

*Whatever is **PUBLIC** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquarePrivate **method*** can access:

Base Class & Sub-derived Class – PRIVATE Member Access Specifier

```
class Shape
{



};
```

class RectanglePrivate : **private** Shape
class SquarePrivate : public RectanglePrivate

*Whatever is **PUBLIC** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquarePrivate **method*** can access:
- nothing in **Shape**

A *SquarePrivate **object*** can access:

Base Class & Sub-derived Class – PRIVATE Member Access Specifier

```
class Shape
{




};
```

class RectanglePrivate : **private** Shape
class SquarePrivate : public RectanglePrivate

*Whatever is **PUBLIC** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

*Whatever is **PROTECTED** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*

*Whatever is **UNACCESSIBLE** the derived*
*is **UNACCESSIBLE** to the sub-derived*

A *SquarePrivate **method*** can access:
- nothing in **Shape**

A *SquarePrivate **object*** can access:
- nothing in **Shape**

Private Access Specifier

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;




protected:




private:
};
```

```
class RectanglePrivate: private Shape
{
 public:

    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

 protected:

    void SetRectAreaProt(int rectHeight,
                         int rectWidth);
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

 private:
   int rectAreaPriv;
};
```

Private Access Specifi

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;



protected:


private:
};
```

*Whatever is **PUBLIC** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

```
class RectanglePrivate: private Shape
{
 public:

    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                        int &rectHeight,
                        int &rectWidth) const;

 protected:

    void SetRectAreaProt(int rectHeight,
                         int rectWidth);

    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

 private:

    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

   int rectAreaPriv;
};
```

This was public in Shape
But it is private in Rectangle

Private Access Specifi

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;



protected:


private:
};
```

*Whatever is **PROTECTED** in the base*
*is **PRIVATE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

```
class RectanglePrivate: private Shape
{
 public:

    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                        int &rectHeight,
                        int &rectWidth) const;

 protected:

    void SetRectAreaProt(int rectHeight,
                         int rectWidth);


 private:

    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;

    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;

   int rectAreaPriv;
};
```

This is protected in Shape
It is also private in Rectangle

Private Access Specifi

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;

protected:
```

Anything Private in Shape
is UNACCESSIBLE to Rectangle
** so it won't be shown here **

```
private:
};
```

*Whatever is **PRIVATE** in the base*
*is **UNACCESSIBLE** to the derived*
*is **UNACCESSIBLE** to the sub-derived*

```
class RectanglePrivate: private Shape
{
 public:
    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
    void SetRectAreaProt(int rectHeight,
                         int rectWidth);

 private:
    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;
   int rectAreaPriv;
};
```

Private Access Specifie

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                      int &rectHeight,
                      int &rectWidth) const;
protected:

private:
```

The access specifier for Rectangle is Public SO
*whatever is **PUBLIC** in Rectangle*
*is **PUBLIC** in Square*

```
class RectanglePrivate: private Shape
{
 public:
    void SetRectDimensionsPub(int rectHeight,
                              int rectWidth);
    int  GetRectAreaPub();
    void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
    void SetRectAreaProt(int rectHeight,
                         int rectWidth);

 private:
    void   SetShapeWidthPub (int newWidth);
    void   SetShapeHeightPub(int newHeight);
    string shapeNamePub;
    int GetShapeWidthProt()  const;
    int GetShapeHeightProt() const;
    int shapeHeightProt;
   int rectAreaPriv;
};
```

Private Access Specifie

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

private:
};
```

The access specifier for Rectangle is Public SO
- *whatever is PROTECTED in Rectangle*
- *is PROTECTED in Square*

```
class RectanglePrivate: private Shape
{
 public:
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

 private:
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

   int rectAreaPriv;
};
```


Private Access Specifie

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

private:
};
```

The access specifier for Rectangle is Public SO
- *whatever is PROTECTED in Rectangle*
- *is PROTECTED in Square*

```
class RectanglePrivate: private Shape
{
 public:
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                       int &rectHeight,
                       int &rectWidth) const;
 protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

 private:
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;

   int rectAreaPriv;
};
```

Private Access Specifie

```
class SquarePrivate
: public RectanglePrivate
{
public:
  // constructor & destructor
   void SetSquareSideLength(int sideLength);
   int  GetSquareArea();
   int  GetSquareSideLength() const;

   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                        int &rectHeight,
                        int &rectWidth) const;

protected:

   void SetRectAreaProt(int rectHeight,
                        int rectWidth);

private:
};
```

The access specifier for Rectangle affects Square's relationship with Shape not Square's relationship with Rectangle

The access specifier for Rectangle is Public SO
- *whatever is **PRIVATE** in Rectangle*
- *is **UNACCESSIBLE** to Square*

```
class RectanglePrivate: private Shape
{
 public:
   void SetRectDimensionsPub(int rectHeight,
                             int rectWidth);
   int  GetRectAreaPub();
   void GetRectDimensionsPub(
                        int &rectHeight,
                        int &rectWidth) const;
 protected:
   void SetRectAreaProt(int rectHeight,
                        int rectWidth);
 private:
   void   SetShapeWidthPub (int newWidth);
   void   SetShapeHeightPub(int newHeight);
   string shapeNamePub;
   int GetShapeWidthProt()  const;
   int GetShapeHeightProt() const;
   int shapeHeightProt;
   int rectAreaPriv;
};
```

What has Access to Shape?

```
class Rectangle: public Shape
class Square   : public Rectangle
```

Shape's Interface	Rectangle OBJECTS	Square OBJECTS	Square Methods
PUBLIC			
PROTECTED			
PRIVATE			

```
class Rectangle: protected Shape
class Square   : public Rectangle
```

Shape's Interface	Rectangle OBJECTS	Square OBJECTS	Square Methods
PUBLIC			
PROTECTED			
PRIVATE			

```
class Rectangle: private Shape
class Square   : public Rectangle
```

Shape's Interface	Rectangle OBJECTS	Square OBJECTS	Square Methods
PUBLIC			
PROTECTED			
PRIVATE			

ACCESS to BASE Methods/Attributes by Derived OBJECTS & Sub-derived METHODS and OBJECTS are impacted by the *Member Access Specifier*

```
class Derived    : public Base
class SubDerived : public Derived
```

ACCESSIBILITY CHART	Derived OBJECTS	Sub-Derived OBJECTS	Sub-Derived METHODS
	Base	Base	Base
PUBLIC	YES	YES	YES
PROTECTED	NO	NO	YES
PRIVATE	NO	NO	NO

```
class Derived    : protected Base
class SubDerived : public Derived
```

ACCESSIBILITY CHART	Derived OBJECTS	Sub-Derived OBJECTS	Sub-Derived METHODS
	Base	Base	Base
PUBLIC	NO	NO	YES
PROTECTED	NO	NO	YES
PRIVATE	NO	NO	NO

```
class Derived    : private Base
class SubDerived : public Derived
```

ACCESSIBILITY CHART	Derived OBJECTS	Sub-Derived OBJECTS	Sub-Derived METHODS
	Base	Base	Base
PUBLIC	NO	NO	NO
PROTECTED	NO	NO	NO
PRIVATE	NO	NO	NO

Object Oriented Design (review)

- The fundamental principles of Object Oriented Design (OOD) are:
  - *Encapsulation*: combines data and operations on data in a single unit
  - *Inheritance*: creates new classes (derived classes) from existing classes (base class)
  - *Polymorphism*: the ability to use the same name to invoke different operations
    - We will learn this next

Exercise

- You have been asked to create a derived class from the ClockType class to implement an AlarmClock
  - The class should include the necessary attributes to support the Alarm Clock
  - The class should include methods to support the following functions
    - Turn Alarm ON/OFF
    - Check Alarm State
    - Set Alarm Time
    - Get Alarm Time
    - Display Alarm Time