# Another way to define a class

## Inheritance..!!

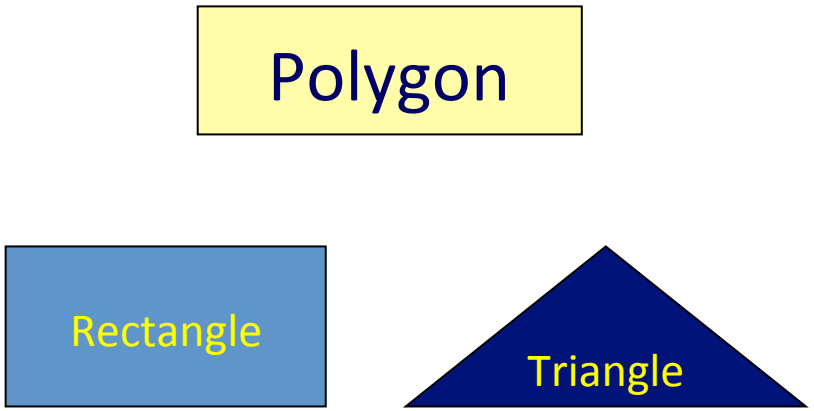
# Why Inheritance ?

Inheritance is a mechanism for

- building class types from existing class types
- defining new class types to be a
  - specialization
  - augmentation

  of existing types

# Inheritance Concept

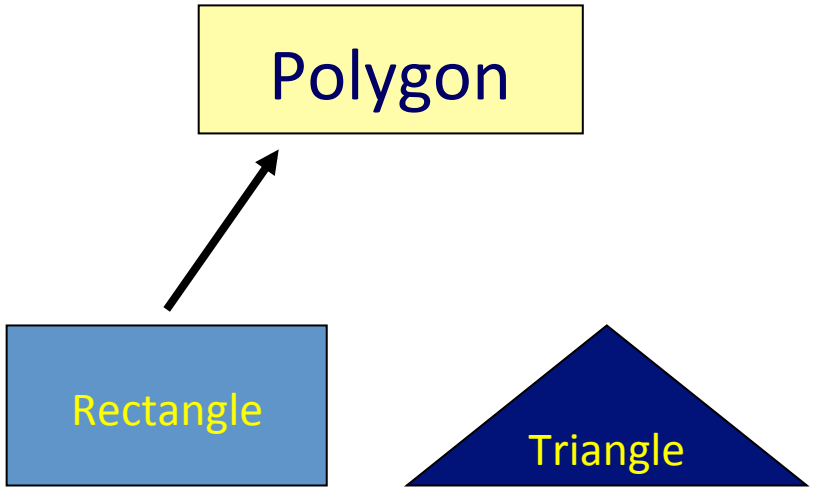



```
class Rectangle{
  private:
    int width, length;
  public:
    void set(int w, int l);
    int area();
}
```

```
class Polygon
{
  private:
    int width, length;
  public:
    void set(int w, int l);
}
```

```
class Triangle{
  private:
    int width, length;
  public:
    void set(int w, int l);
    int area();
}
```

# Inheritance Concept

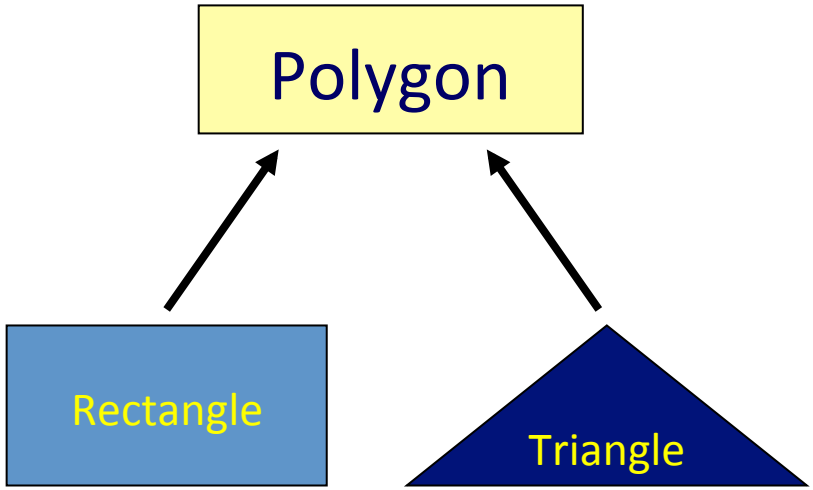



```
class Polygon
{
    protected:
      int width, length;
    public:
      void set(int w, int l);
}
```

```
class Rectangle : public Polygon
{
    public: int area();
}
```

```
class Rectangle{
    protected:
      int width, length;
    public:
      void set(int w, int l);
      int area();
}
```

# Inheritance Concept

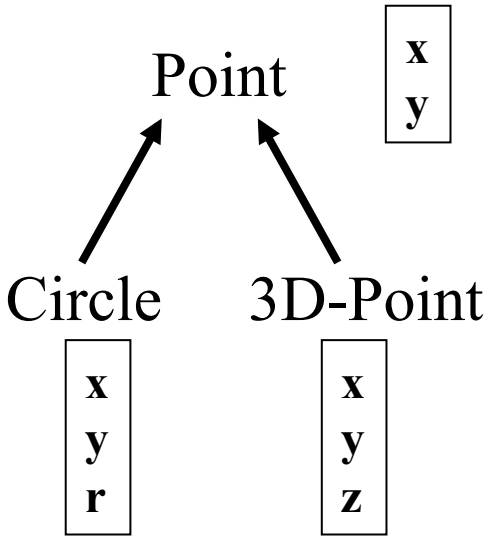



```
class Polygon
{
    protected:
      int width, length;
    public:
      void set(int w, int l);
}
```

```
class Triangle : public Polygon
{
    public: int area();
}
```

```
class Triangle{
    protected:
      int width, length;
    public:
      void set(int w, int l);
      int area();
}
```

# Inheritance Concept



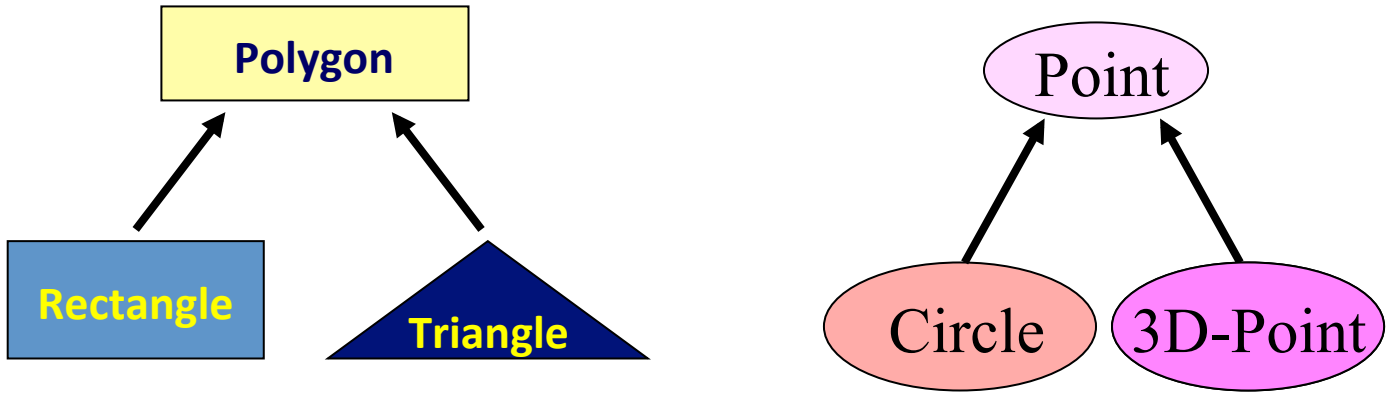

```
class Point
{
    protected:
      int x, y;
    public:
      void set(int a, int b);
}
```

```
class Circle : public Point
{
    private:
      double r;
}
```

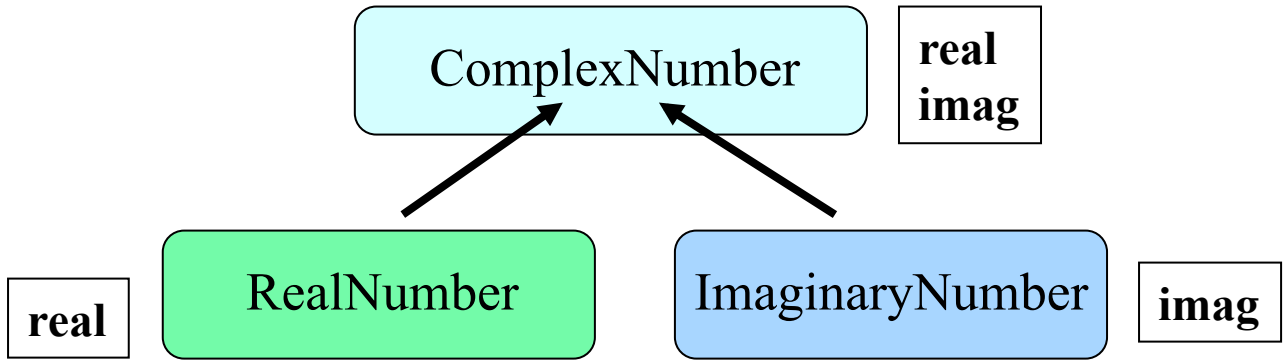
```
class 3D-Point: public Point
{
    private:
      int z;
}
```

# Inheritance Concept

- Augmenting the original class




- Specializing the original class



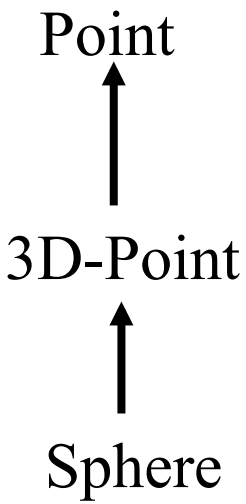
# Define a Class Hierarchy

- Syntax:

  class *DerivedClassName* : access-level *BaseClassName*

  where
  - access-level specifies the type of derivation
    - private by default, or
    - public
- Any class can serve as a base class
  - Thus a derived class can also be a base class

# Class Derivation




```
class Point{
    protected:
      int x, y;
    public:
      void set(int a, int b);
}
```

```
class 3D-Point : public Point{
    private:  double z;
    … …
}
```

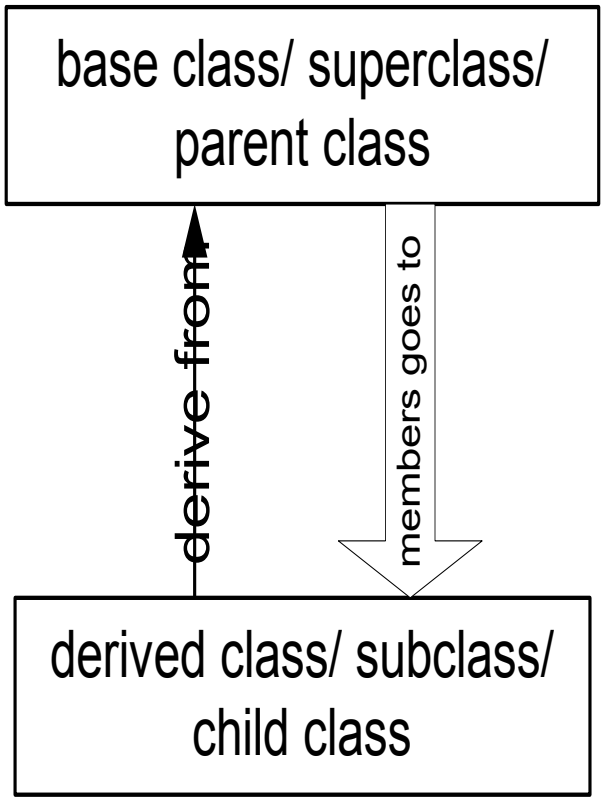
```
class Sphere : public 3D-Point{
    private:  double r;
    … …
}
```

Point is the base class of 3D-Point, while 3D-Point is the base class of Sphere

# What to inherit?

- In principle, every member of a base class is inherited by a derived class
  - just with different access permission

# Access Control Over the Members




- Two levels of access control over class members
  - class definition
  - inheritance type

```
class Point{
    protected: int x, y;
    public: void set(int a, int b);
}
```

```
class Circle : public Point{
    … …
}
```

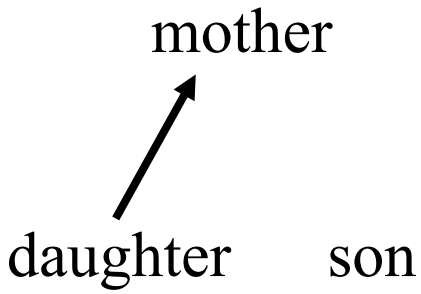
# Access Rights of Derived Classes

Type of Inheritance

| Access Control for Members | private | protected | public |
|---|---|---|---|
| private | private | private | private |
| protected | private | protected | protected |
| public | private | protected | public |

- The type of inheritance defines the minimum access level for the members of derived class that are inherited from the base class
- With public inheritance, the derived class follow the same access permission as in the base class
- With protected inheritance, the public and the protected members inherited from the base class can be accessed in the derived class as protected members
- With private inheritance, none of the members of base class is accessible by the derived class

# Class Derivation



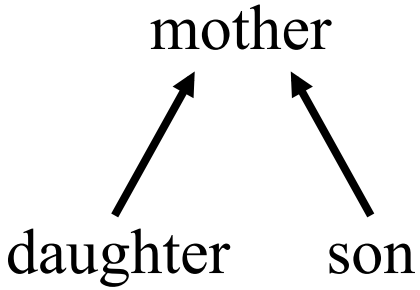

```
class daughter : public mother{
    private:
      double a;
    public:
      void foo ( );
}
```

```
class mother{
    protected:
      int x, y;
    public:
      void set(int a, int b);
    private:
      int z;
}
```

```
void daughter :: foo ( ){
    x = y = 20;
    set(5, 10);
    cout<<"value of a "<<a<<endl;
    z = 100;    // error, a private
    member
}
```

daughter can access 3 of the 4 inherited members

# Class Derivation




```
class son : protected mother{
    private:
      double b;
    public:
      void foo ( );
}
```

```
class mother{
    protected:
      int x, y;
    public:
      void set(int a, int b);
    private:
      int z;
}
```

```
void son :: foo ( ){
    x = y = 20;
    set(5, 10); //it becomes a protect member
    cout<<"value of b "<<b<<endl;
    z = 100;    // error, not a public member
}
```

# What to inherit?

- In principle, every member of a base class is inherited by a derived class
  - just with different access permission
- However, there are exceptions for
  - constructor and destructor
  - operator=() member
  - friends

  Since all these functions are class-specific

# Constructor Rules for Derived Classes

The default constructor and the destructor of the base class are always called when a new object of a derived class is created or destroyed.

```
class A {
  public:
    A ( )
     {cout<< "A:default"<<endl;}
    A (int a)
     {cout<<"A:parameter"<<endl;}
}
```

```
class B : public A
{
  public:
    B (int a)
       {cout<<"B"<<endl;}
}
```

```
B test(1);
```

output:

```
A:default
B
```

# Constructor Rules for Derived Classes

You can also specify an constructor of the base class other than the default constructor

**DerivedClassCon ( derivedClass args ) : BaseClassCon ( baseClass args )**
**{ DerivedClass constructor body }**

```
class A {
  public:
    A ( )
     {cout<< "A:default"<<endl;}
    A (int a)
     {cout<<"A:parameter"<<endl;}
}
```

```
class C : public A
{
  public:
    C (int a) : A(a)
       {cout<<"C"<<endl;}
}
```

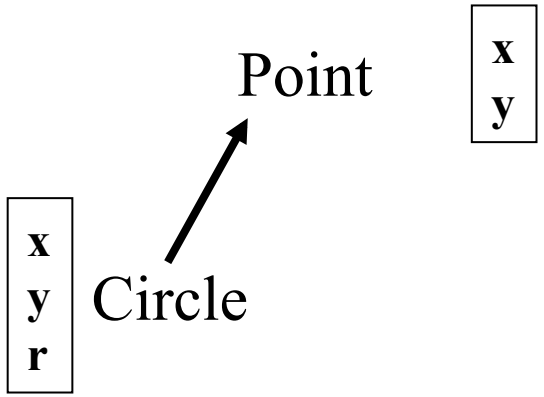
```
C test(1);
```

output:

```
A:parameter
C
```

# Define its Own Members

The derived class can also define its own members, in addition to the members inherited from the base class




```
class Circle : public Point{
    private:
      double r;
    public:
      void set_r(double c);
}
```

```
class Point{
    protected:
      int x, y;
    public:
      void set(int a, int b);
}
```

```
    protected:
      int x, y;
    private:
      double r;
    public:
      void set(int a, int b);
      void set_r(double c);
```

# Even more …

- A derived class can override methods defined in its parent class. With overriding,
  - the method in the subclass has the identical signature to the method in the base class.
  - a subclass implements its own version of a base class method.

```
class A {
  protected:
     int x, y;
   public:
     void print ()
      {cout<<“From A”<<endl;}
}
```

```
class B : public A
{
   public:
     void print ()
         {cout<<“From B”<<endl;}
}
```

# Access a Method

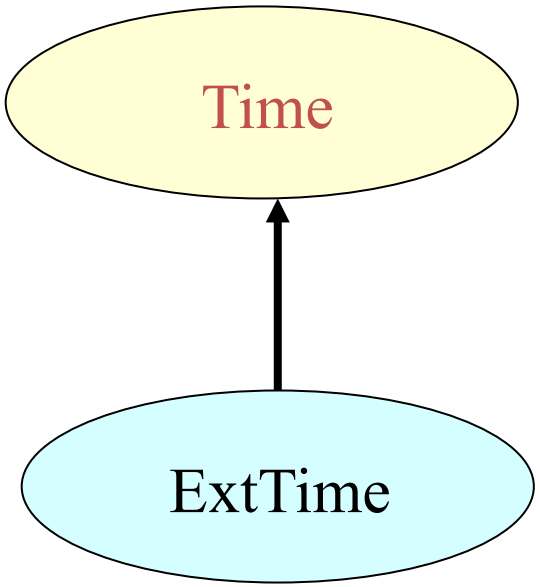
```
class Point
{
    protected:
      int x, y;
    public:
      void set(int a, int b)
     {x=a; y=b;}
      void foo ();
      void print();
}
```

```
class Circle : public Point{
  private:  double r;
  public:
     void set (int a, int b, double c) {
         Point :: set(a, b); //same name function call
         r = c;
     }
     void print();  }
```

```
Point A;
A.set(30,50);  // from base class Point
A.print(); // from base class Point
```

```
Circle C;
C.set(10,10,100);   // from class Circle
C.foo ();  // from base class Point
C.print(); // from class Circle
```

# Putting Them Together

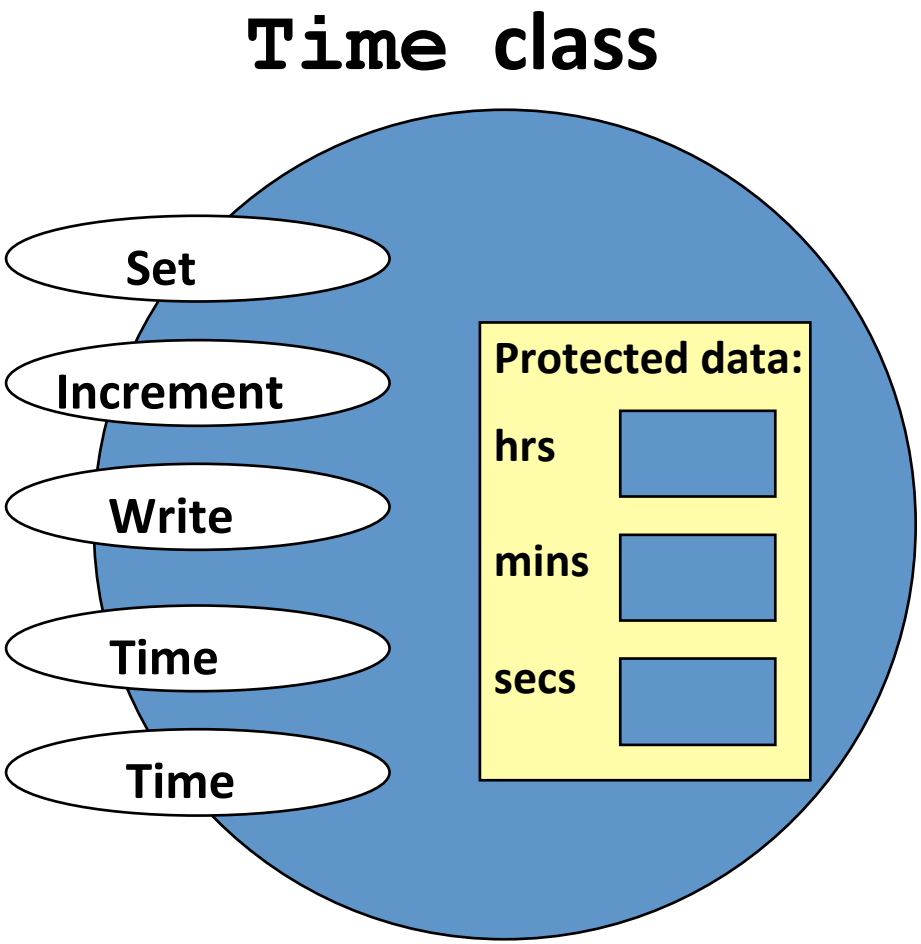



- Time is the base class
- ExtTime is the derived class with public inheritance
- The derived class can
  - inherit all members from the base class, except the constructor
  - access all public and protected members of the base class
  - define its private data member
  - provide its own constructor
  - define its public member functions
  - override functions inherited from the base class

# class **`Time`** Specification

```
// SPECIFICATION   FILE                ( time.h)

class  Time
{
  public :

     void     Set ( int h, int m, int s ) ;
     void     Increment ( ) ;
     void     Write ( )  const ;
     Time     ( int initH, int initM, int initS ) ;   //  constructor
     Time     ( ) ;                                   //  default constructor

  protected :

     int           hrs ;
     int           mins ;
     int           secs ;
} ;
```

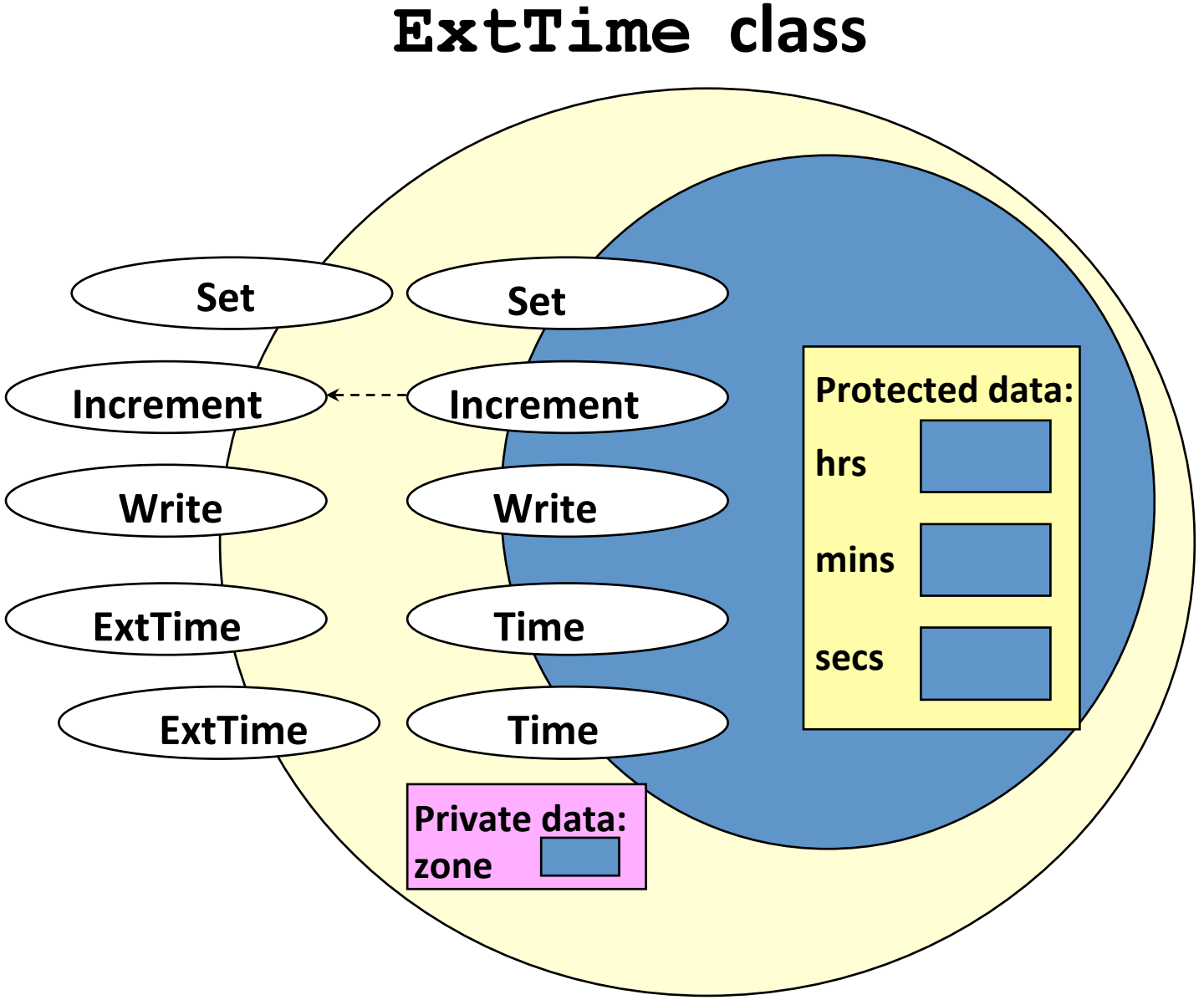
# Class Interface Diagram



# Derived Class **`ExtTime`**

```
// SPECIFICATION   FILE                    ( exttime.h)

#include   "time.h"
enum  ZoneType {EST, CST, MST, PST, EDT, CDT, MDT, PDT } ;

class  ExtTime  :  public  Time
      // Time is the base class and use public inheritance
{
  public :
     void        Set ( int h, int m, int s, ZoneType timeZone ) ;
     void        Write ( )  const;    //overridden
     ExtTime     (int initH, int initM, int initS, ZoneType initZone ) ;
     ExtTime     ();   // default constructor

private :
     ZoneType  zone ;  //  added data member
} ;
```

# Class Interface Diagram



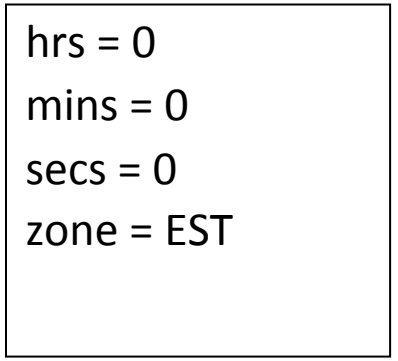
# Implementation of **`ExtTime`**

Default Constructor

```
ExtTime :: ExtTime ( )
{
      zone  =  EST ;
}
```

```
ExtTime et1;
```

et1



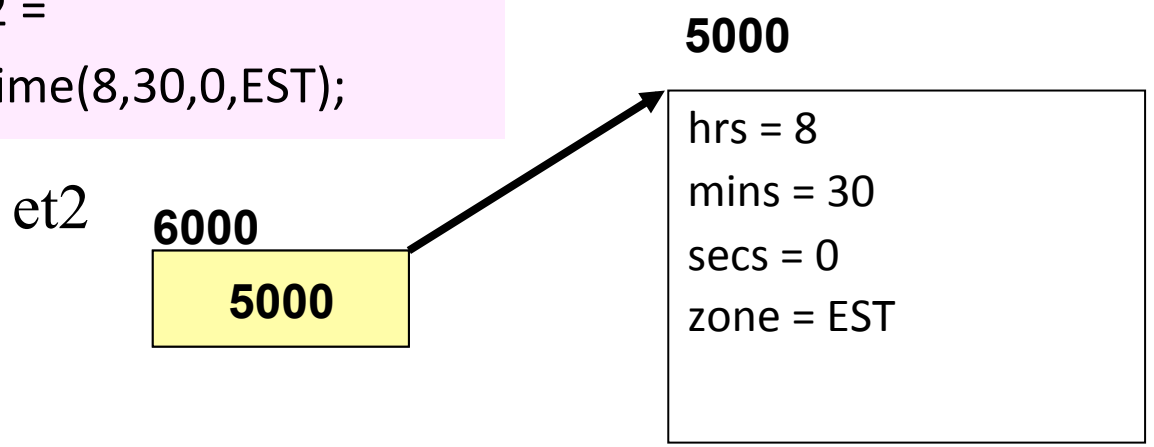

The default constructor of base class, Time(), is automatically called, when an ExtTime object is created.

# Implementation of **`ExtTime`**

Another Constructor

```
ExtTime :: ExtTime (int initH, int initM, int initS, ZoneType initZone)
        : Time (initH, initM, initS)
        // constructor initializer
{
          zone  = initZone ;
}
```

```
ExtTime *et2 =
     new ExtTime(8,30,0,EST);
```



# Implementation of `ExtTime`

```
void  ExtTime :: Set (int h, int m, int s, ZoneType timeZone)
{
     Time :: Set (hours, minutes, seconds);  // same name function call
     zone  = timeZone ;
}
```

```
void  ExtTime :: Write ( )   const  // function overriding
{
  string  zoneString[8] =
      {"EST", "CST", MST", "PST", "EDT", "CDT", "MDT", "PDT"} ;

 Time :: Write ( ) ;
 cout  <<' '<<zoneString[zone]<<endl;
}
```

# Working with **`ExtTime`**

```
#include  “exttime.h”
… …
int main()
{
  ExtTime    thisTime ( 8, 35, 0, PST ) ;
  ExtTime    thatTime ;                    // default constructor called

  thatTime.Write( ) ;              // outputs 00:00:00 EST

  thatTime.Set (16, 49, 23, CDT) ;
  thatTime.Write( ) ;              // outputs 16:49:23 CDT

  thisTime.Increment ( ) ;
  thisTime.Increment ( ) ;
  thisTime.Write ( ) ;             // outputs 08:35:02  PST
}
```

# Take Home Message

- Inheritance is a mechanism for defining new class types to be a specialization or an augmentation of existing types.

- In principle, every member of a base class is inherited by a derived class with different access permissions, except for the constructors