CS 1B Review Part 2

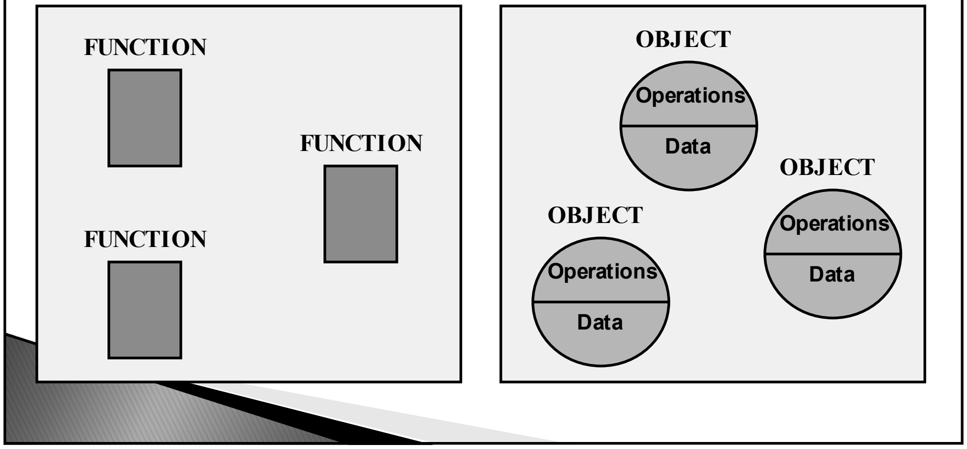
Topics

- Structured Programming vs. Object-Oriented Programming
- Using Inheritance to Create a New C++ class Type
- Using Composition (Containment) to Create a New C++ class Type
- Static vs. Dynamic Binding of Operations to Objects

Two Programming Paradigms

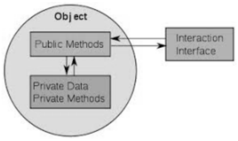
Structural (Procedural) PROGRAM

Object-Oriented PROGRAM (OOP)



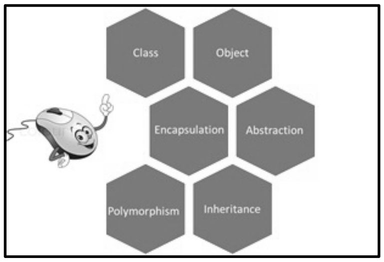
OOP vs. Structured Programming

- In OOP an object is a fundamental entity, while in structured programming a function is a fundamental entity
- In OOP objects are debugged, while in structured programming functions are debugged
- In structured programming a program is a collection of interacting functions, while in OOP a program is a collection of interacting objects
- In structured programming the programmer is action oriented, while in OOP the programmer is object oriented
- The object-oriented programming (OOP) implements Object Oriented Design (OOD)



The Fundamentals of Object Oriented Design (OOD)

- Encapsulation
  - Combine data and operations on data in a single unit.
- Inheritance
  - Create new objects from existing objects
- Polymorphism
  - The ability to use the same expression to denote different operations

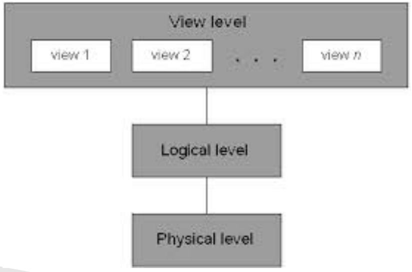


Object-Oriented Programming Language Features

1. Data abstraction
   – Separates the logical properties of a data type from its implementation

2. Inheritance of properties

3. Dynamic binding of operations to objects

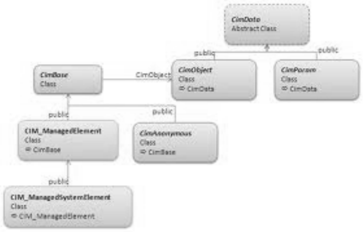


Terms

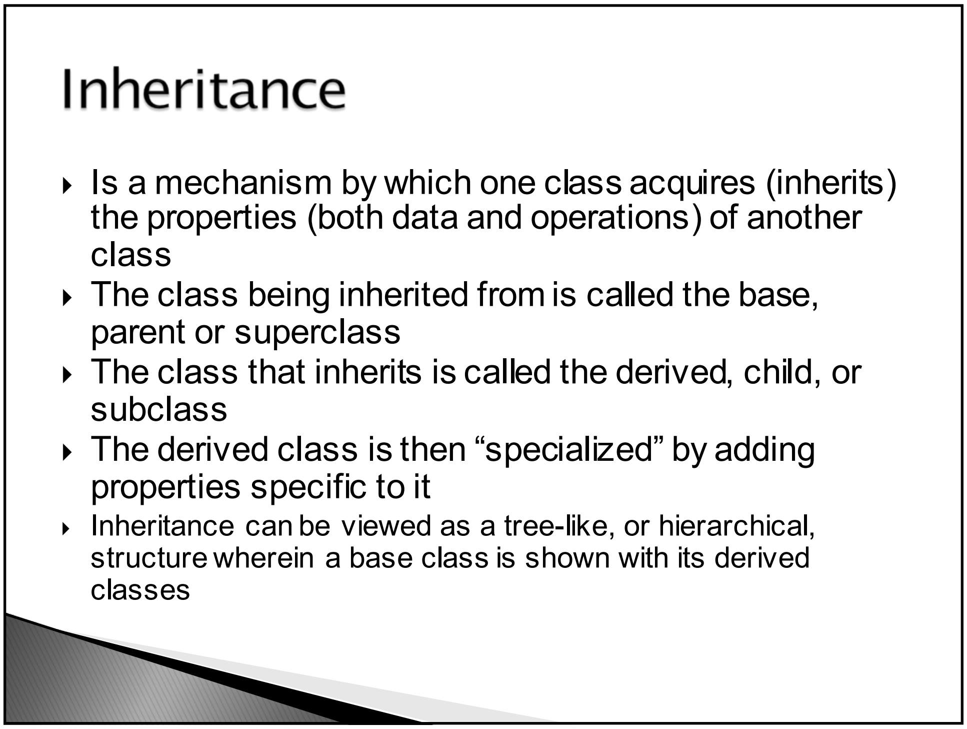
OOP Terms	C++ Equivalents
Object	Class object or class instance
Instance variable	Private data member
Method	Public member function
Message passing	Function call (to a public member function)

Relationship between C++ classes

- C++ classes can be related to each other in various ways

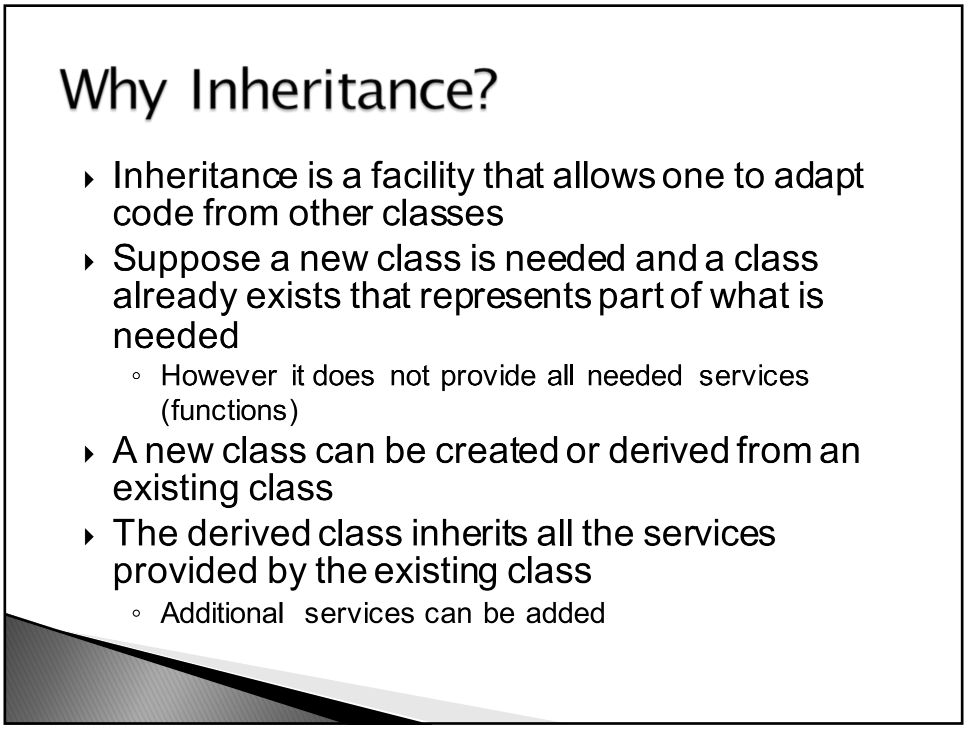


- The three most common ways
  - Two classes are independent (nothing in common)
  - Two classes are related by inheritance
  - Two classes are related by composition



Inheritance

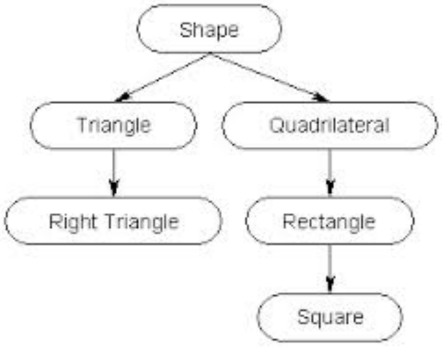
- Is a mechanism by which one class acquires (inherits) the properties (both data and operations) of another class
- The class being inherited from is called the base, parent or superclass
- The class that inherits is called the derived, child, or subclass
- The derived class is then “specialized” by adding properties specific to it
- Inheritance can be viewed as a tree-like, or hierarchical, structure wherein a base class is shown with its derived classes



Why Inheritance?

- Inheritance is a facility that allows one to adapt code from other classes
- Suppose a new class is needed and a class already exists that represents part of what is needed
  - However it does not provide all needed services (functions)
- A new class can be created or derived from an existing class
- The derived class inherits all the services provided by the existing class
  - Additional services can be added

Inheritance Example



Shape

Triangle

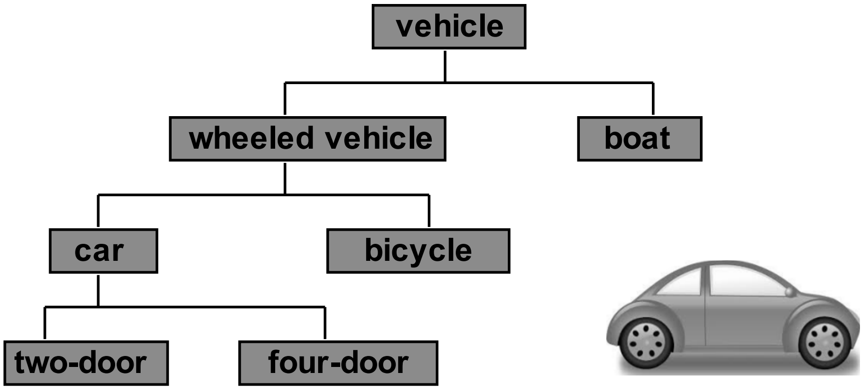
Quadrilateral

Right Triangle

Rectangle

Square

Inheritance Hierarchy Among Vehicles



vehicle

wheeled vehicle

boat

car

bicycle

two-door

four-door

Every car is a wheeled vehicle

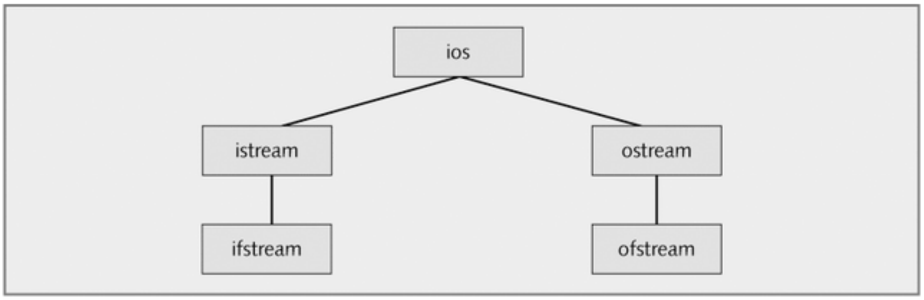
"is a" Relationship

- The inheritance relationship can be viewed as a "is a" relationship
- For example
  - Every car "is a" vehicle (a car inherits properties of a vehicle)
  - Every two door car "is a" car (a two-door car inherits properties of a car)

Vehicle Inheritance

```
class vehicle {/* . . . */};
class wheeledVehicle : public vehicle { /* . . . */};
class boat : public vehicle { /* . . . */};
class car : public  wheeledVehicle {/* . . . */ };
class bicycle : public  wheeledVehicle {/* . . . */ };
class two-door : public car {/* . . .*/};
class four-door : public car {/* . . .*/};
```

C++ Stream Classes



C++ stream classes hierarchy

ios class

- The class ios is the base class for all stream classes
- Classes istream and ostream are directly derived from the class ios
- The class ifstream is derived from the class istream, and the class ofstream is derived from the class ostream
- The class ios contains formatting flags and member functions to access and/or modify the setting of these flags
- To identify the I/O status, the class ios contains an integer status word
  - This integer status word provides a continuous update reporting the status of the stream
- The classes istream and ostream are responsible for providing the operations for the data transfer between memory and devices

ios Derived Classes

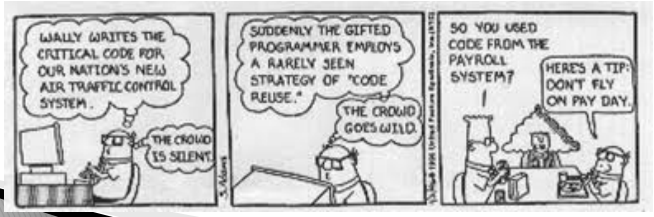
- The class istream defines the extraction operator, >>, and functions such as get and ignore
- The class ostream defines the insertion operator, <<, which is used by the object cout
- The class ifstream is derived from the class istream to provide the file input operations
- The class ofstream is derived from the class ostream to provide the file output operations
- Objects of the type ifstream are used for file output; objects of the type ofstream are used for file output
- The header file fstream contains the definition of the classes ifstream and ofstream

Inheritance Allows One to Reuse Code

- A "child" or derived class inherits members from one or more base or "parent" classes

Code Reuse

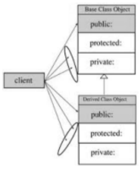
- Inherited members literally become part of the derived class without having to be rewritten or copied


WALLY WRITES THE CRITICAL CODE FOR OUR NATION'S NEW AIR TRAFFIC CONTROL SYSTEM.
THE CROWD IS SILENT.
SUDDENLY THE GIFTED PROGRAMMER EMPLOYS A RARELY SEEN STRATEGY OF "CODE REUSE."
THE CROWD GOES WILD.
SO YOU USED CODE FROM THE PAYROLL SYSTEM?
HERE'S A TIP: DON'T FLY ON PAY DAY.
S. Adams

Inheritance Allows for Adaptation of Code

- Inheritance allows one to create specialized classes that add to or modify the basic concept or behavior of a more generalized class
  - For example: A square is a special type of rectangle
- The derived (child) class inherits all the properties of the base (parent) class (except for the private members)
  - The data and operations defined in the base class are also defined in the derived class
- Specific properties are added to the derived class to make it unique
- Inheritance allows the creation of extensible data abstractions
  - The derived classes extends the base class by adding private data and public operations

Access Specifiers (Expanded View)



- public: (public interface)
  - Provides the interface between the client code and the class objects
  - Are accessible by both client code and derived classes
  - Function members (methods) are generally declared public
- private: (members are inaccessible to clients)
  - Not accessible by any client nor are they accessible by derived classes
  - Default
  - Data members are generally declare private
- protected:
  - Not accessible by client code but are accessible by derived classes

Access Method for Inheritance – 1

- **Public Inheritance**
  - **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class
  - A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class

Access Method for Inheritance – 2

- **Protected Inheritance**
  - **public** and **protected** members of the base class become **protected** members of the derived class
- **Private Inheritance**
  - **public** and **protected** members of the base class become **private** members of the derived class

Base Initialization list

- Data members can be initialized in a constructor using a “base initialization list”
- Data members are initialized after the parenthesis that ends the parameter list
  - On the function definition statement after a :
  - Not within the body of the function
- The invocation of a base class constructor within a child class constructor requires this syntax

Using a Base Initialization list

```
class  Time
{
public :
    Time ( int  initHrs, int  initMins,  int  initSecs ) ;
    Time ( ) ;
 private :
    int           hrs ;
    int           mins ;
    int              secs ;
} ;
Time :: Time (  ) : hrs(0), mins(0), secs(0)
{     // empty body
}
Time :: Time (int   initHrs,  int   initMins,  int   initSecs ):
               hrs (initHrs),
               mins (initMins),
               secs  (initSecs)
{ // empty body }
```

Time Specification

```
class  Time
{
public :
    void Set ( int  hours , int  minutes , int  seconds ) ;
    void Increment ( ) ;
    void Write  ( )  const ;
    Time  ( int  initHrs, int  initMins,  int  initSecs ) ;
    Time  ( ) ;
 private :
    int             hrs ;
    int             mins ;
    int              secs ;
} ;
```

Adding a data member

- Desire: Add time zone member
- Ways to accomplish this
  - Change the time class specification and implementation files
    - Not always possible since source code is sometimes proprietary
    - This method would also violate the encapsulation paradigm
  - Use inheritance principal
    - Create a new class, called ExtTime, that inherits the properties of the time class

The General Syntax of a Derived Class

```
class className: memberAccessSpecifier
  baseClassName
{
  member list
};
```

where memberAccessSpecifier is public, protected, or private

- When no memberAccessSpecifier is specified, it is assumed to be a private inheritance
- Example specifying inheritance

```
class   ExtTime   :  public   Time  // Time is a public base
   class
```

memberAccessSpecifier

- public
  - All the public members of the base class (except for the constructors) are also public members of child class
  - Clients can invoke the public members (except for the constructors) of the base class for the derived class objects
- private
  - Public members of the base class are not public members of the derived class
  - Clients of the derived class cannot invoke the base class methods on the derived class objects
- A derived class cannot access the private members of its base class
  - Would violate the encapsulation paradigm
- All data members of the base class are also data members of the derived class
  - Similarly, the member functions of the base class (unless redefined) are also the member functions of the derived class

Steps Needed to Create a Child Class

- Procedure to create a child class
  - Add new data member(s)
  - Write new constructor(s) (required)
  - Add or overwrite member functions if necessary
- Constructor rules for derived classes
  - At run time, the base class constructor is implicitly called first, before the body of the derived class's constructor executes
  - If the base class constructor requires parameters, they must be passed by the derived class's constructor

Inherit the ExtTime Class from the Time Class

- For the ExtTime class
  - New data member zone is added
  - Member functions Set and Write are overridden
  - The increment function for the Time class can be invoked for ExtTime class objects (not overridden)
- The private members of ExtTime are hrs, mins, secs, (inherited from Time), and zone
- Note: every ExtTime object is a Time object
  - Time is the base class and ExtTime is the derived class

ExtTime Specification

```
enum  ZoneType {EST, CST, MST, PST, EDT, CDT, MDT, PDT } ;
class  ExtTime  :  public  Time
{
public :
   void  Set ( int  hours,  int  minutes,  int   seconds ,
            ZoneType  timeZone ) ;
   void Write  ( )  const ;
   ExtTime ( int  initHrs ,  int  initMins ,  int  initSecs ,
           ZoneType  initZone ) ;
   ExtTime ( ) ;
 private :
   ZoneType  zone ;      //  added data member
} ;
```

ExtTime Constructors

```
ExtTime :: ExtTime (  ) : Time( )

{
          zone  =  EST ;
}
//*******************************************************

ExtTime :: ExtTime(int   initHrs,
                 int    initMins,
                      int    initSecs,
                 ZoneType   initZone )
                :  Time (initHrs, initMins, initSecs)
{
        zone  = initZone ;
}
```

Instantiating ExtTime objects

- ExtTime thisTime(8,35,0,PST) ;
  - The first three parameters are passed to the Time constructor before zone is set in the ExtTime constructor

- ExtTime thatTime ;
  - The default constructor of the Time class is called before the zone is set to EST

ExtTime Set function

```
void  ExtTime :: Set  (int              hours,
                       int              minutes,
                       int              seconds,
                       ZoneType  timeZone )
{
   Time :: Set (hours, minutes, seconds);  // calls base function
         zone  = timeZone ;
}
```

- There are two distinct Set( ) functions
  - Time::Set(...) and ExtTime::Set(...)
  - The ExtTime::Set() calls the Time::Set() to set the hrs, mins, secs
  - It cannot set hrs, mins, and secs since they are private members of the Time class
  - The ExtTime::Set( ) sets the zone

Redefinition of Member Functions

- To redefine a member function, the redefinition must have
  - Same name
  - Same signature as the function it replaces
- Otherwise there are two distinct functions
  - Original function (which is inherited)
  - The new function with a different signature
- (See example inherit1.cpp)

Avoiding Multiple Inclusion of Header Files

- Often several program files use the same header file containing typedef statements, constants, or class type declarations--but, it is a compile-time error to define the same identifier twice
- This preprocessor directive syntax is used to avoid the compilation error that would otherwise occur from multiple uses of #include for the same header file

```
#ifndef   Preprocessor_Identifier
#define  Preprocessor_Identifier
      .
      .
      .
#endif
```

- (See example inherit2.cpp)

Composition (or Containment)

- Is a mechanism by which the internal data (the state) of one class includes an object of another class
- One object is contained within a class
- There is no special syntax
- An object is declared to be one of the data members of another class
  - Like a struct within a struct
- Composition is a “has a” relationship
  - For example, a Timecard object “has a” Time object

TimeCard Class

- A typical timecard class would need
  - Employee id
  - Time the employees “punches in or punches out”

- The TimeCard class can use the Time class

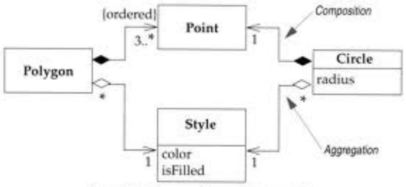


Figure 6-6: *Aggregation and Composition*

TimeCard Specification

```
class TimeCard
   {
   public:
     void  Punch (int   hours, int   minutes, int   seconds ) ;
     void  Print (  )  const ;
     TimeCard  (long   idNum, int  initHrs, int   initMins, int initSecs )
  ;
     TimeCard (  ) ;
    private:
      long    id ;
      Time   timeStamp  ;
   } ;
```

TimeCard Constructors

```
TimeCard :: TimeCard( )
{
                    id  =  0 ;
}
//*******************************************************
TimeCard :: TimeCard  (long   idNum, int     initHrs, int     initMins,  int
   initSecs ):  timeStamp (initHrs, initMins, initSecs)
{
                    id  =  idNum ;
}
```

Difference between Inheritance and Composition Constructors

- When using inheritance

ExtTime :: ExtTime(int initHrs, int initMins, int initSecs, ZoneType initZone) : Time (initHrs, initMins, initSecs) // base class specified

- When using composition

TimeCard :: TimeCard (long idNum, int initHrs, int initMins, int initSecs : timeStamp (initHrs, initMins, initSecs) // member object specified

TimeCard Print Function

```
void TimeCard :: Print() const
  {
        cout << "ID: "<< id << " Time: " ;
        timeStamp.Write();
  }
```

- The timecard object can manipulate “id” via its member functions
- It must use the Time card member functions to access private members of the Time class
- The Print() and Punch() functions both invoke methods from the Time class
  - Otherwise the encapsulation paradigm would be violated
- See examples comp1.cpp and comp2.cpp

©Ron Leishman
Image: 1082958
illustrationsOf.com

Multiple Member Objects

- When a class has several members that are objects of other classes
  - Constructors with parameters must specify the parameters for each base class

TimeCard :: TimeCard (long idNum, int initHrs, int initMins, int initSecs)
: timeStamp (initHrs, initMins, initSecs),
anotherTimeStamp (initHrs, initMins, initSecs)

- Member objects don't have to be from the same base class

Order in Which Constructors are Executed

- Given a class A
  - If A is a derived class its base class constructor is executed first
  - Finally, the body of A's constructor is executed
- Given Class A has a class member that is an object of class B
  - Class B's constructor is executed before Class A's

NewExtTime :: NewExtTime (int initHrs, int initMins,int initSecs, ZoneType initZone) : Time (initHrs, initMins, initSecs), timeStamp (initHrs, initMins, initSecs) // Time is a base class and timeStamp is a member object

Multiple Inheritance

- C++ support multiple inheritance

- (see examples: multiple inheritance1 and 2)

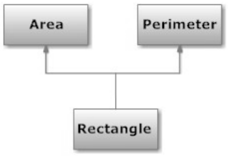


Figure: Multiple Inheritance Example