OBJECT ORIENTED PROGRAMMING-C++

SUBMITTED BY :-
RANI CHANDI
PG DEPARTMENT OF COMPUTER SCIENCE & IT
CLASS-BSc(IT)-Sem II

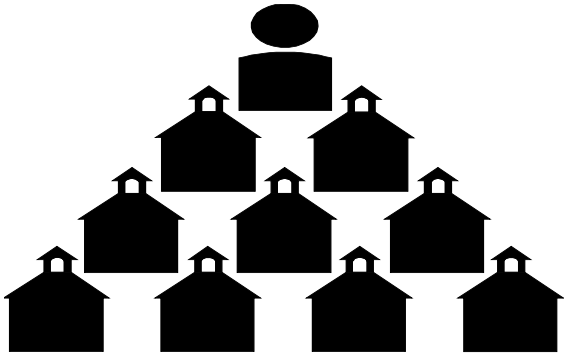
INTRODUCTION TO INHERITANCE

In C++

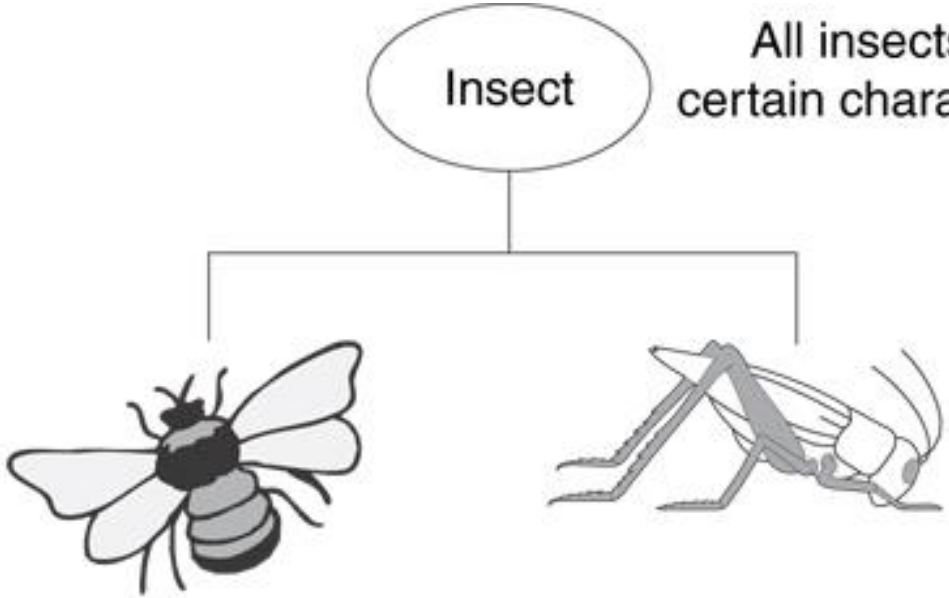
- Inheritance is a mechanism for
  - building class types from other class types
  - defining new class types to be a
    - specialization
    - augmentation
  - of existing types

Inheritance

- Single Inheritance
  - Class inherits from one base class
- Multiple Inheritance
  - Class inherits from multiple base classes
- Three types of inheritance:
  - `public`: Derived objects are accessible by the base class objects (focus of this chapter)
  - `private`: Derived objects are inaccessible by the base class
  - `protected`: Derived classes and `friend`s can access protected members of the base class



Example: Insect Taxonomy



In addition to the common insect characteristics, the bumble bee has its own unique characteristics such as the ability to sting.

In addition to the common insect characteristics, the grasshopper has its own unique characteristics such as the ability to jump.

BASED AND DERIVED CLASSES

- Often an object from a derived class (subclass) “is an” object of a base class (superclass)

Base class	Derived classes
Student	GraduateStudent UndergraduateStudent
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeImprovementLoan MortgageLoan
Employee	FacultyMember StaffMember
Account	CheckingAccount SavingsAccount

Fig. 19.1 Some simple inheritance examples.

Inheritance – Terminology and Notation in C++

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

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

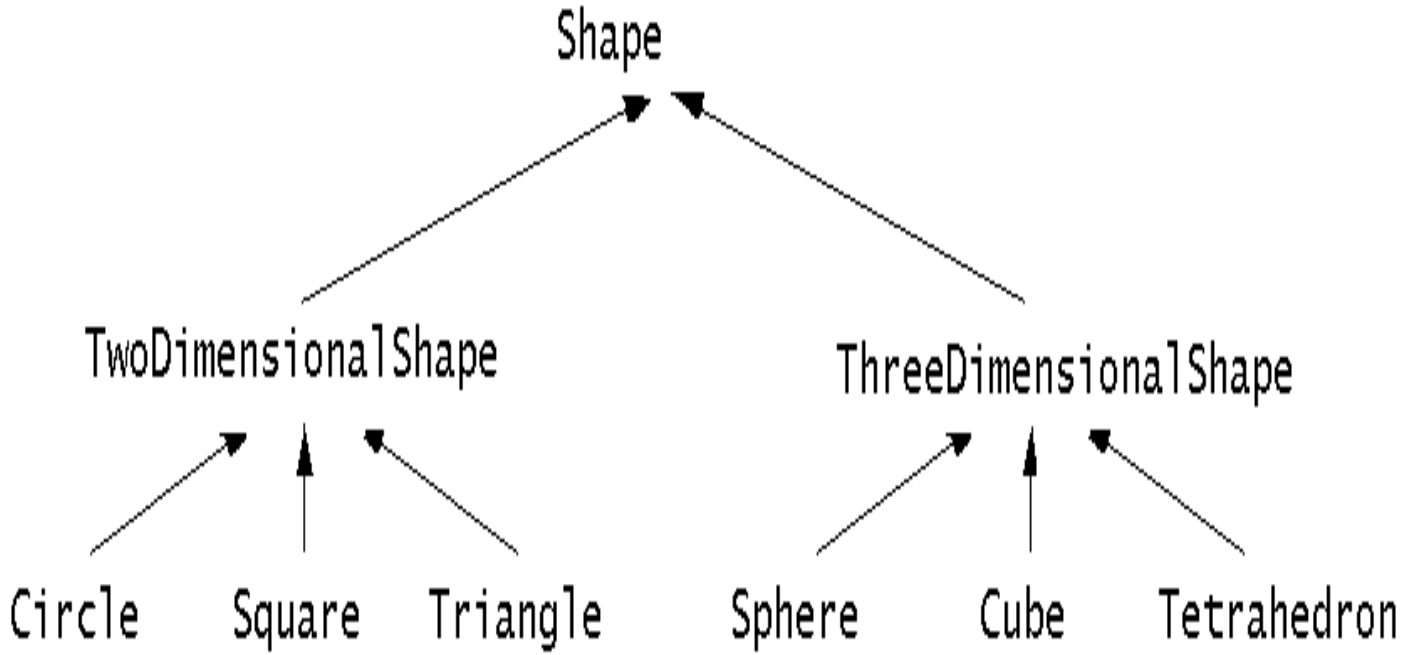
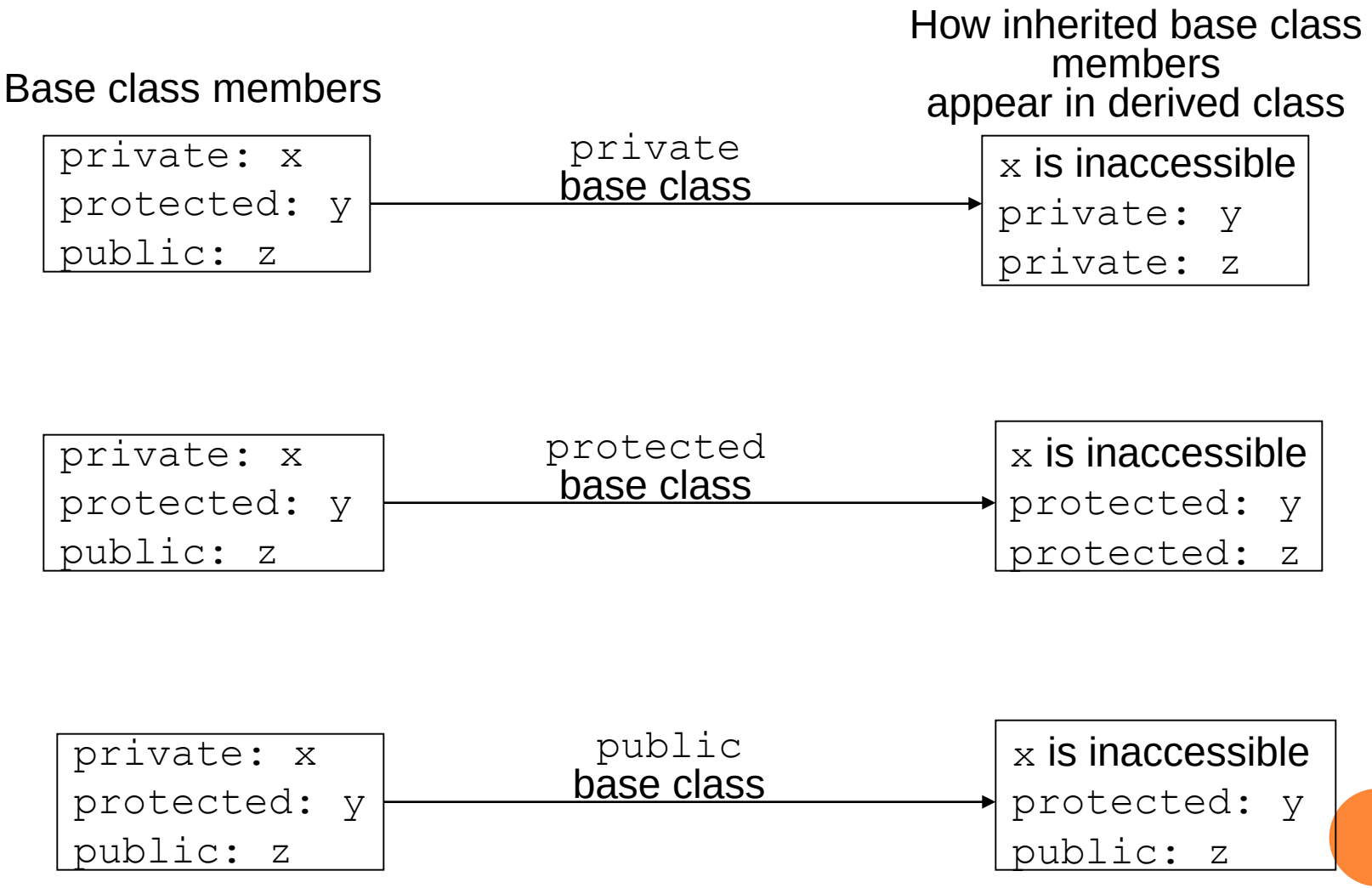


Fig. 19.3 A portion of a Shape class hierarchy.

Class Access Specifiers

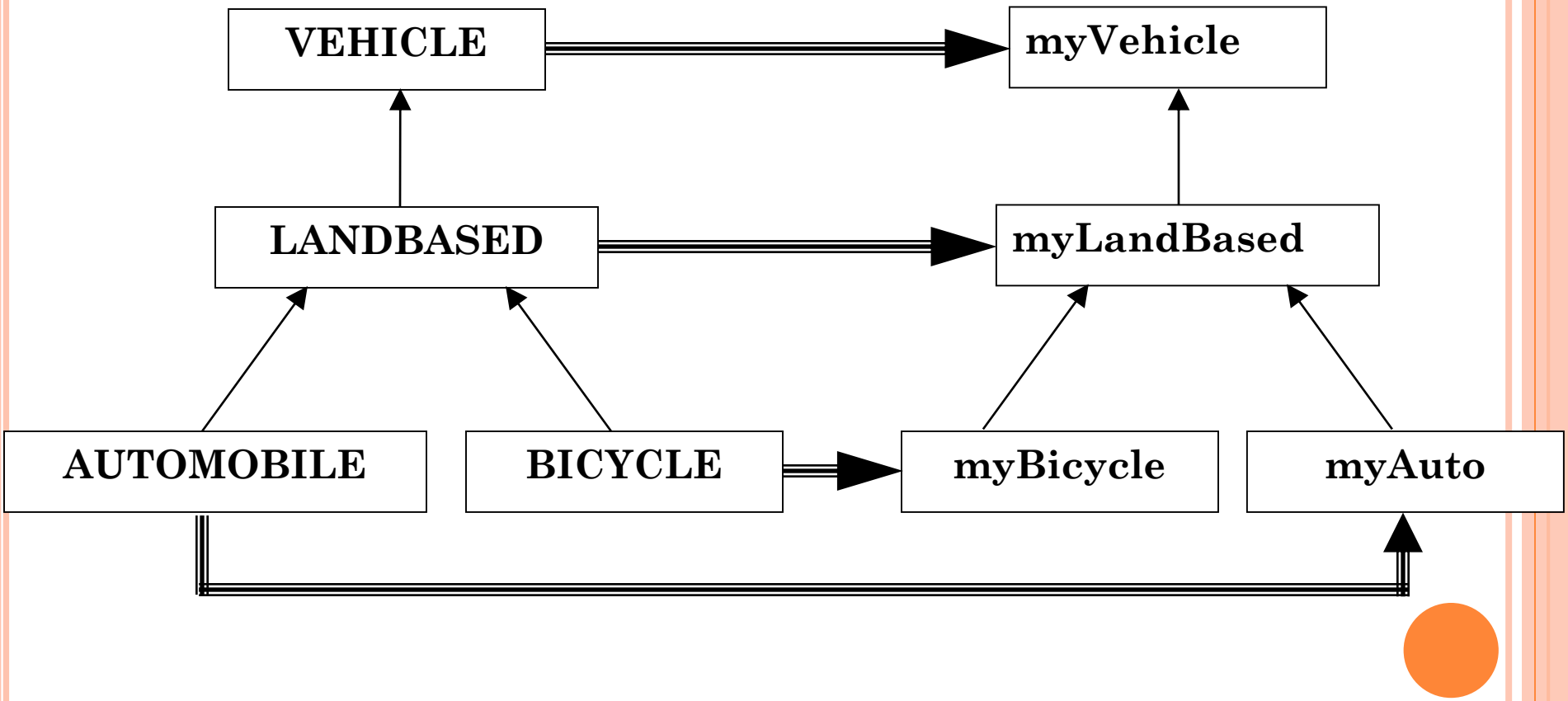
1) `public` – object of derived class can be treated as object of base class (not vice-versa)
2) `protected` – more restrictive than `public`, but allows derived classes to know details of parents
3) `private` – prevents objects of derived class from being treated as objects of base class.

Inheritance vs. Access



Class and Instance Hierarchy

We actually have two hierarchies



Single/Multiple Inheritance

Single Inheritance

Each class or instance object has a single parent

Multiple Inheritance

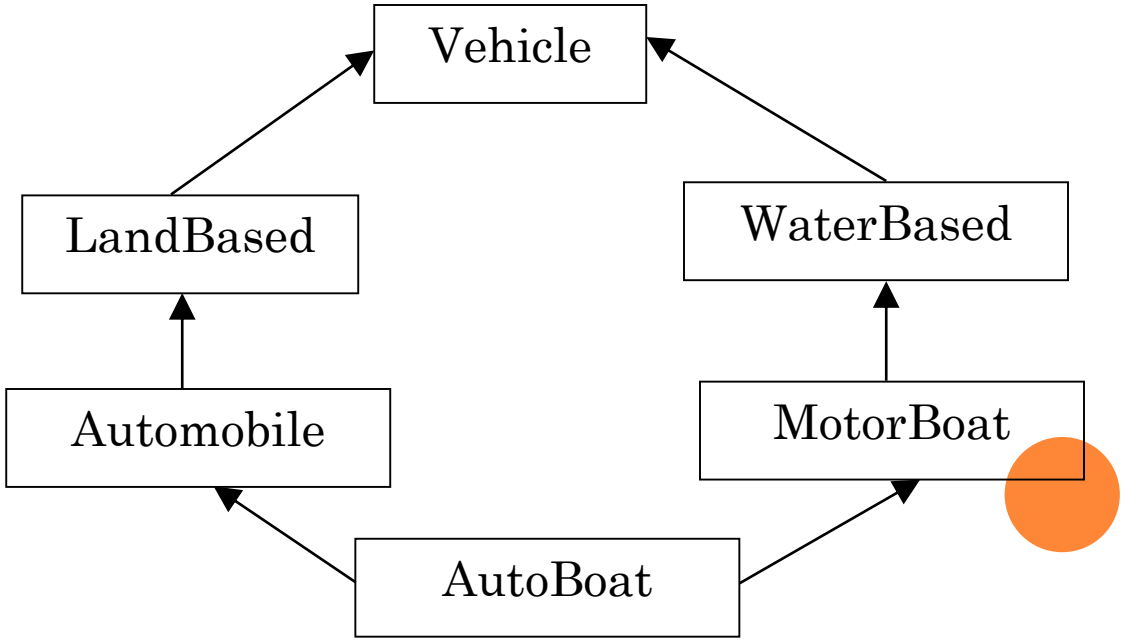
Classes inherit from multiple base classes (might not have same ancestors as shown in the example below)

Defines a relationship

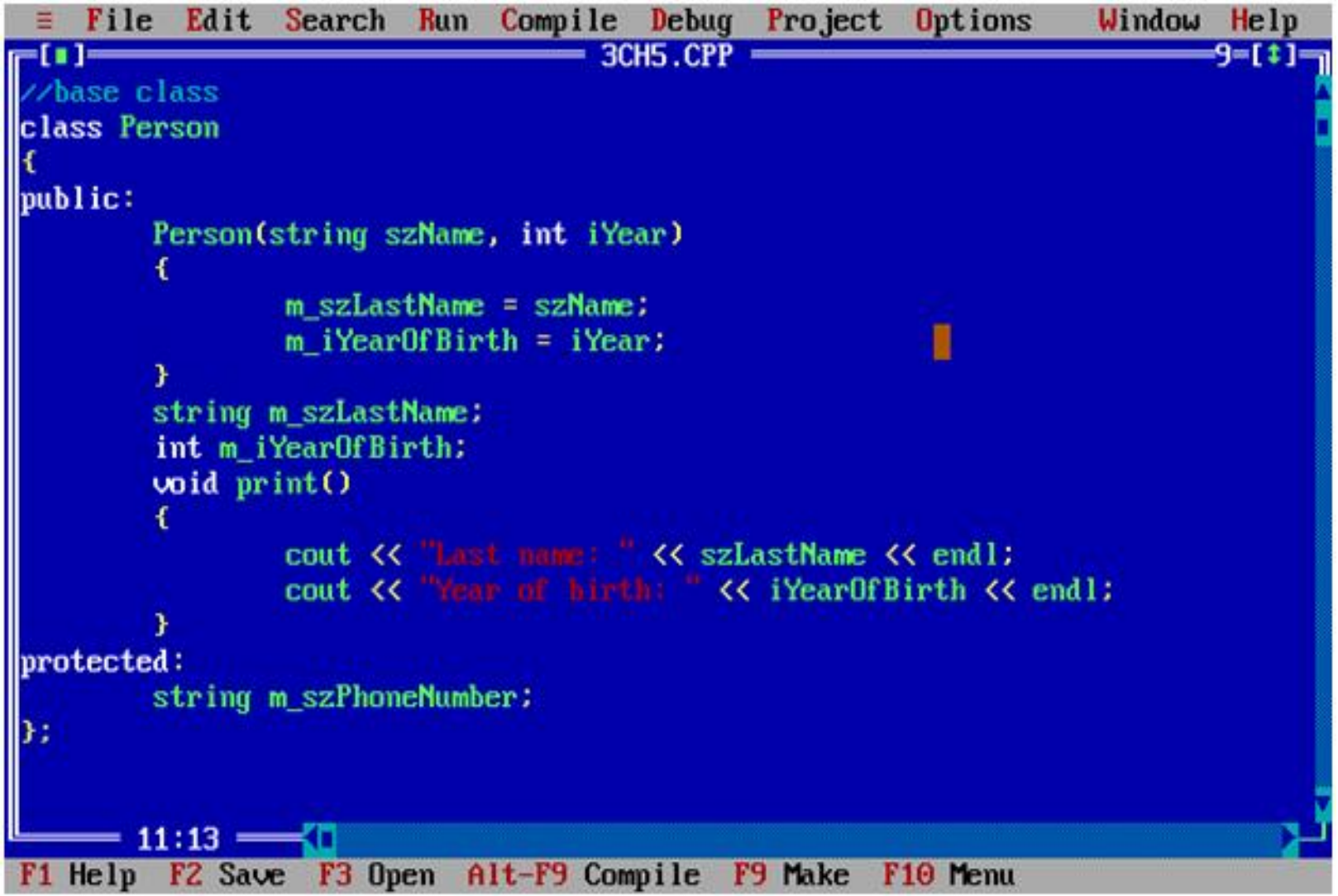
Between several (independent) class types

Example:

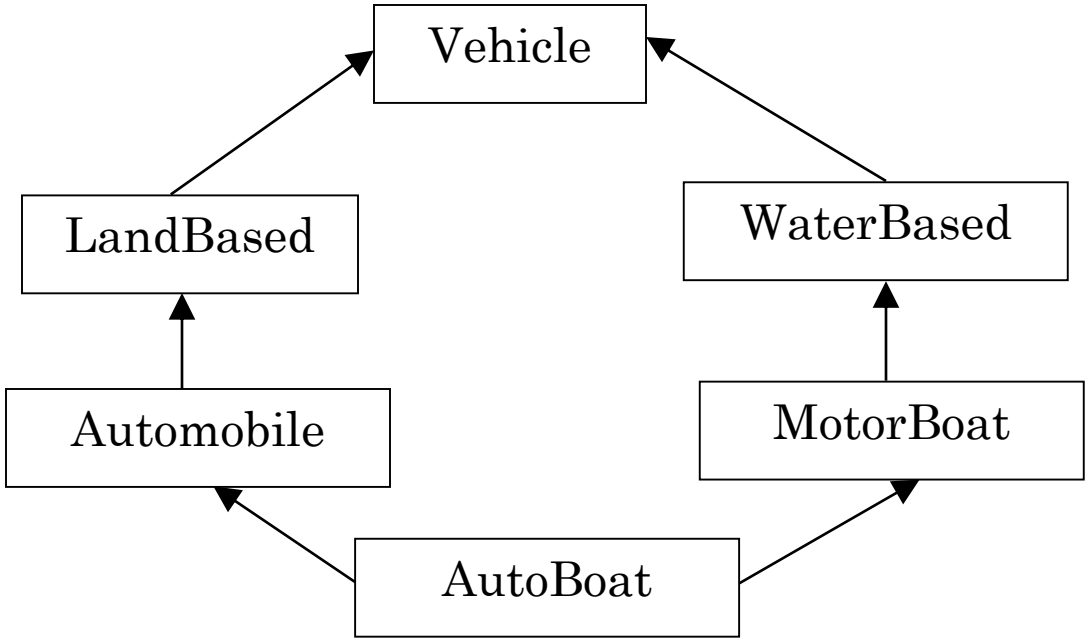
Multiple Parents Common Ancestor



EXAMPLE OF SINGLE INHERITANCE



Virtual Inheritance



The derived class AutoBoat…

Inherits Attributes and Properties

From

Automobile

MotorBoat

Vehicle

An example of multiple base classes

```
#include <iostream>
using namespace std;
class base1 {
    protected:
        int x;
    public:
         void showx() { cout << x
<< "\n";}
};
class base2 {
      protected:
          int y;
      public:
  void showy() { cout<< y << "\n";
};
```

```
class derived: public base1,
public base2 {
     public:
void set(int  i, int  j) { x= i;
y =j; }};

int main(){
derived ob;
ob.set(10, 20);//from derived
ob.showx();//from basel
ob.showy();//from base2
return 0;
}
```

EXAMPLE OF MULTILEVEL INHERITANCE

EXAMPLE

```
#include<iostream>
using namespace std;
class grandparent
{
public:
   void gshow()
   {
  cout<<"intelligent";
   }
};
class parent:public grandparent
{
public:
            void pshow()
            {
                        cout<<"\nhandsom";
            }
};
class child:public parent
{
public:
   void cshow()
   {
      cout<<"\nobedience";
   }
};
main()
{

   grandparent obj1;
    parent obj2;
    child obj3;
    cout<<"QUALITY OF GRANDPA\n";
   obj1.gshow();
 cout<<"\nQUALITIES OF FATHER\n";
 obj2.gshow();
  obj2.pshow();
  cout<<"\nQUALITIES OF CHILD\n";
  obj3.gshow();
  obj3.pshow();
  obj3.cshow();
}
```

Class Derivation

Class C be derived from base class A - PUBLIC

class C : public A

In class C

The inherited *public* members of A

Appear as *public* members of C

If myValue is a public data member of A

myValue can be accessed publicly through instances of C

```
C myC;
myC.myValue; // ok
```

Class Derivation

Class C be derived from base class B - PRIVATE

class C : private B

In class C

The inherited *public* members of B

Appear as *private* members of C

if myValue is a public data member of B

myValue cannot be accessed publicly and directly through instances of C

```
C myC;
myC.myValue; // compile error
```

Function members of C can still access public members of B as public

Simple Inheritance - Example

```
#include <iostream>

class BaseClass {
public:
        BaseClass( ) : baseValue (20) { };
        BaseClass(int aValue) : baseValue (aValue) { };
        int baseValue;
};

class DerivedClass : public BaseClass {
public:
        DerivedClass() : derivedValue (10){ };
        DerivedClass(int aDerivedValue) : BaseClass(15), derivedValue(aDerivedValue){ };
        int getDerivedValue() { return derivedValue; }
private:
        int derivedValue;
};
int main( )
{
        BaseClass base;
        DerivedClass child(5);
        cout << base.baseValue << endl;                 //  will print 20
        cout << child.baseValue << endl;                //  will print 15
        cout << child.getDerivedValue() << endl;        //  will print  5
        return 0;
}
```

Derived Class – Public Inheritance

- Generally all data is private, so we just add additional data members in the derived class by specifying them in the private section
- Any base class member functions that are not specified in the derived class are inherited unchanged, with the following exceptions: constructor, destructor, copy constructor, and operator=.
- Any base class member function that is declared in the derived class' private section is disabled in the derived class
- Any base class member function that is declared in the derived class' public section requires an overriding definition that will be applied to objects of the derived class
- Additional member functions can be added in the derived class

Derived Class-Public Inheritance

```
class Derived : public Base {
  // Any members that are not listed are inherited unchanged except
  // for constructor, destructor, copy constructor, and operator=
 public:
  // Constructors, and destructors if defaults are not good
  // Base members whose definitions are to change in Derived
  // Additional public member functions
 private:
 // Additional data members (generally private)
 // Additional private member functions
 // Base members that should be disabled in Derived
};
```

VISIBILITY RULES

- Any private members in the base class are not accessible to the derived class (because any member that is declared with private visibility is accessible only to methods of the class)
- What if we want the derived class to have access to the base class members?

1) use public access =>public access allows access to other classes in addition to derived classes
2) use a friend declaration=>this is also poor design and would require friend declaration for each derived class
3) make members protected =>allows access only to derived classes

   A *protected class member* is private to every class except a derived class, but declaring data members as protected or public violates the spirit of encapsulation

1) write accessor and modifier methods =>the best alternative

   Note: However, if a protected declaration allows you to avoid convoluted code, then it is not unreasonable to use it.

Public, Private, and Protected Inheritance

Base class member access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by any non-`static` member functions, `friend` functions and non-member functions.	`protected` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.	`private` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.	`private` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by non-`static` member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by non-`static` member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by non-`static` member functions and `friend` functions through `public` or `protected` member functions of the base class.

Fig. 19.6 Summary of base-class member accessibility in a derived class.

THANK YOU