INHERITANCE

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

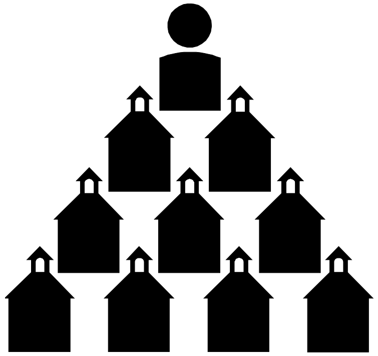
Inheritance

Subgroupings with respect to a parent are called

- Subclass
- Derived Class
- Children

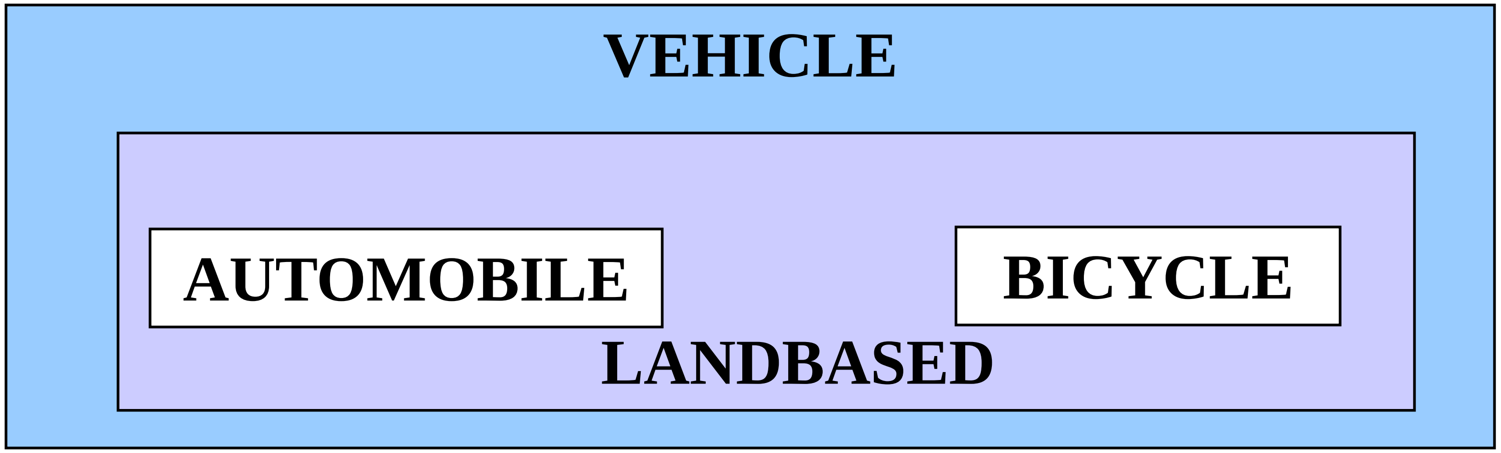
The derived class

- inherits from the parent all
  - characteristics
  - properties
  - capabilities
- can modify or extend inherited abilities



Inheritance

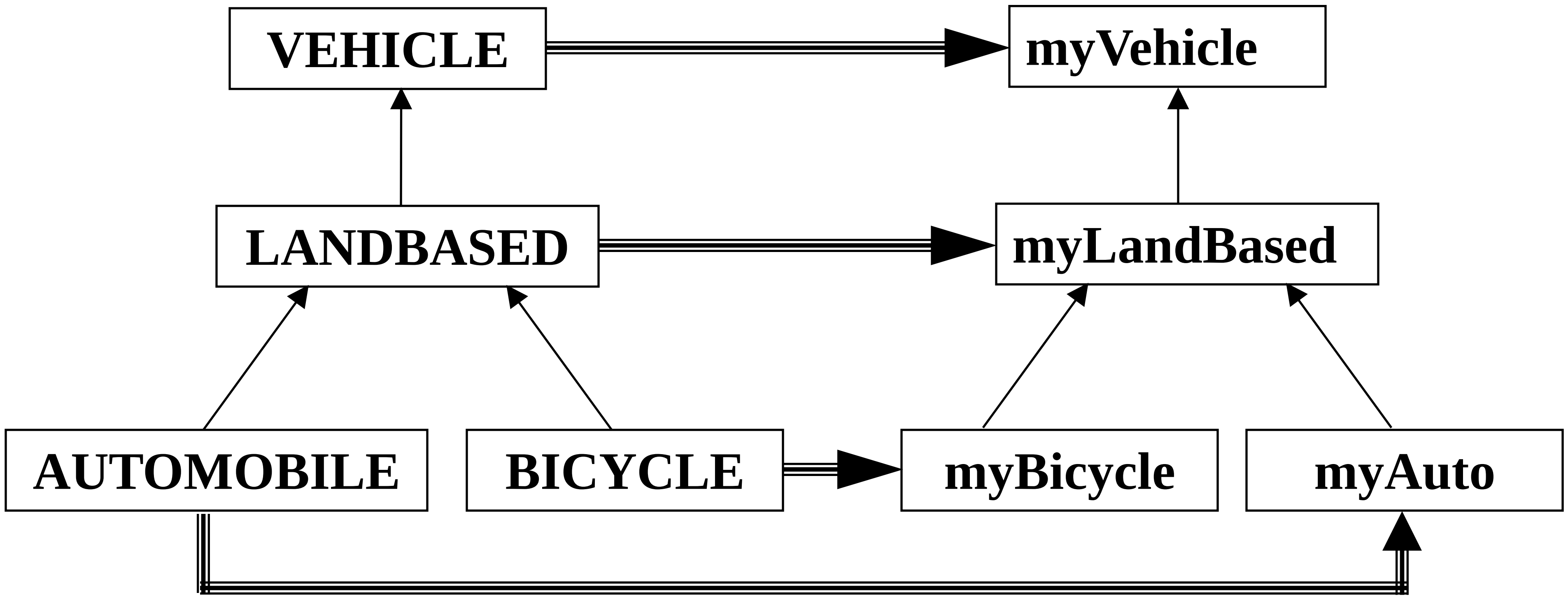
- Consider the vehicle hierarchy:



- Each entity is also a class
- Classes are used in an object centered paradigm as a means of encapsulating common data and functions shared by members of the class

Class and Instance Hierarchy

There are 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)

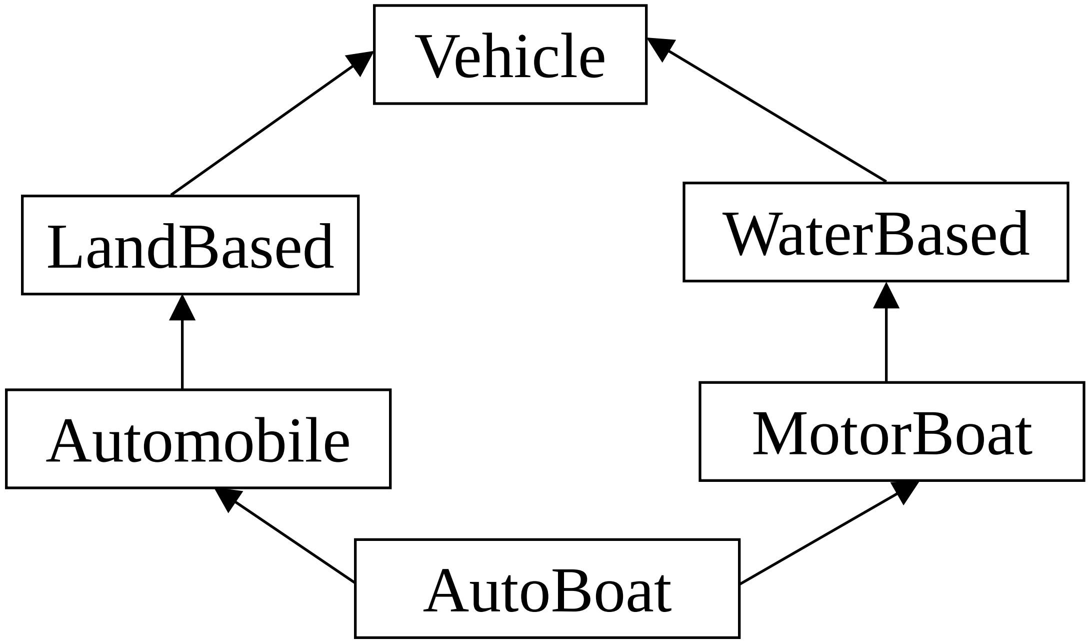
Defines a relationship

Between several (independent) class types

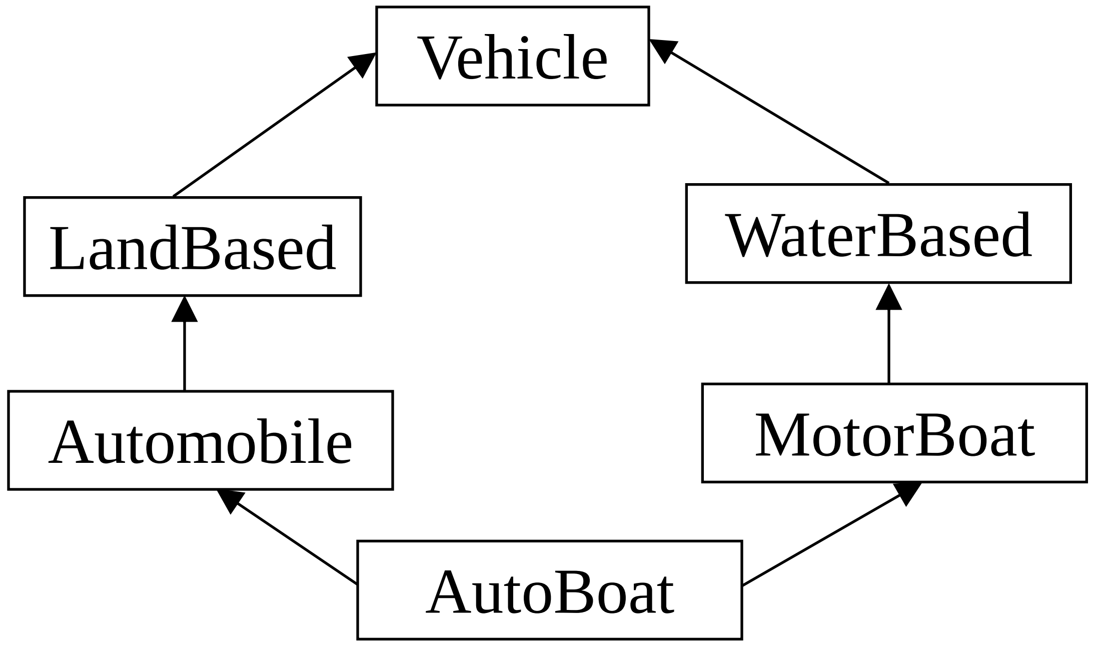
Example:

Multiple Parents

Common Ancestor



Virtual Inheritance



The derived class AutoBoat…

Inherits Attributes and Properties

From

Automobile

MotorBoat

Vehicle

Derivation Hierarchy

- The class Vehicle is an abstraction…
  - It represents an encapsulation of common
    - Properties
    - Attributes
- Its sole purpose
  - Define a class from which to derive other classes….
    - It encapsulates common
      - Data members
      - Function members
- Called
  - Abstract Base Class
  - Abstract Super Class

Class Derivation

Any class can serve as a base class…

- Thus a derived class can also be a base class
- Worth spending time at the outset of a design to develop sound definitions

syntax

class DerivedClassName:specification BaseClassName

DerivedClassName - the class being derived

specification - specifies access to the base class members

- public
- protected
- private

- private by default

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;
```

Class Derivation

<u>Class C be derived from base class B - PRIVATE</u>

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 =><u>the best alternative</u>

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

Class Derivation
Constructors and Destructors

- Constructors and destructors are not inherited
  - Each derived class should define its constructors/destructor
  - If no constructor is written=> hidden constructor is generated and will call the base default constructor for the inherited portion and then apply the default initialization for any additional data members
- When a derived object is instantiated, memory is allocated for
  - Base object
  - Added parts
- Initialization occurs in two stages:
  - the base class constructors are invoked to initialize the base objects
  - the derived class constructor is used to complete the task
- The derived class constructor specifies appropriate base class constructor in the initialization list
  - If there is no constructor in base class, the compiler created default constructor used
- If the base class is derived, the procedure is applied recursively

Inherited Member Initialization

- Initialized in two ways:
  - If the base class has only a default constructor=> initialize the member values in the body of the derived class constructor
  - If the base class has a constructor with arguments=> the initialization list is used to pass arguments to the base class constructors

<u>syntax (for Single Base Class</u>)

```
DerivedClass ( derivedClass args ) : BaseClass ( baseClass args )
        {
        DerivedClass constructor body
        }
```

- The set of derived class constructor arguments may contain initialization values for the base class arguments

Derived class DateTime

```
class DateTime : public Date // derived from base class Date
{
private:
// additional data members
int hour, minutes;
public:
// additional member functions
void DateTime(); //default constructor
void DateTime(const char* mm_dd_yy, int h, int mi); // constructor
void setTime(int h, int m);
void addMinutes(int m);
void addHours(int h);
void display(void) const;                    // overrides Date::display()
};
```

Derived Class DateTime

```
DateTime::DateTime(const char * mm_dd_yy, int h, int mi)
   : Date(mm_dd_yy) , hours(h) , minutes(mi)
      // invokes the constructor Date(const char* mmddyy) {}

void DateTime::display() const {
//add the code for displaying the date and time .
//Hint : cout<<(Date&)(*this)<<endl;  will display the date
}

void DateTime::AddHours(int h) {
//make sure that if the hours goes more than 24 you have to add 1 day
// and you can do that as follows (*this) += 1;
}
```

Derived Class DateTime

```
int main(){
   DateTime  dt(”11/06/01”,20,15);  //constructor
   dt+=5;   // operator+=() of class Date
   cout << ”Date is: ” << justdt<<endl;
   dt.AddHours(2);   // AddHours() of class DateTime
   dt.display();   // display() of class DateTime
   return 0;
}
```