ICOM 4015
Advanced Programming

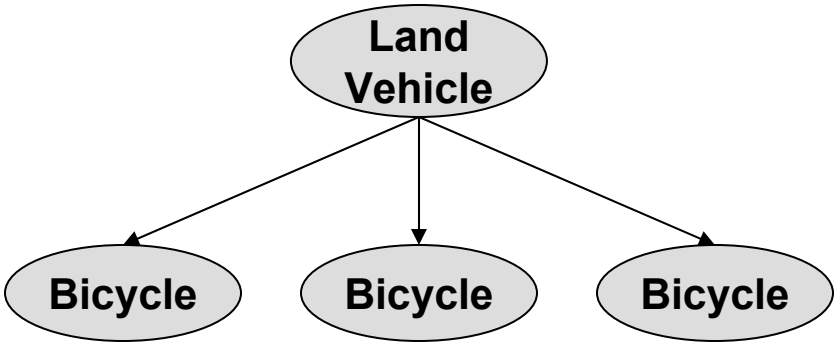
Lecture 14
Data Abstraction II
Class Specialization
Subtype Polymorphism

Prof. Bienvenido Vélez

Inheritance
Subtype Polymorphism
Outline

- Defining derived classes
  - Member inheritance
  - Method overriding
- Virtual methods
- Abstract classes

Goal Class Hierarchy



The LandVehicle Class Declaration

```
// LandVehicle.h

#ifndef LAND_VEHICLE_H
#define LAND_VEHICLE_H

class LandVehicle {
 private:
  int numWheels;
  int numAxels;
  float originalPrice;

 public:
  // Constructor without parameters
  LandVehicle(int w = 4, int a = 2, float p = 1000.0);
  ~LandVehicle();
  // Access data members
  int getAxels();
  int getWheels();
  float getOriginalPrice();   // price purchased

  // Calculate other LandVehicle properties
  int   getLife();    // return the # of years
  float getTax(int yr);
  float getValue(int yr);
};

#endif
```

The LandVehicle Class Definition

```
// LandVehicle.cc
#include "LandVehicle.h"

// Construction and destruction
LandVehicle::LandVehicle(int w, int a, float p) {
  numWheels = w;
  numAxels = a;
  originalPrice = p;
}
LandVehicle::~LandVehicle(){} // Empty destructor

// Access data members
int   LandVehicle::getAxels()          { return numAxels;      }
int   LandVehicle::getWheels()         { return numWheels;     }
float LandVehicle::getOriginalPrice() { return originalPrice; }

// Calculate other LandVehicle properties
int   LandVehicle::getLife()           { return 10;            }

// Tax calculated based on year, and price
float LandVehicle::getTax(int yr)
{
    return (getValue(yr)/100.0) * 0.25;
}

// calculated based on weight + axels + wheels
float LandVehicle::getValue(int yr) {
  if (yr > getLife())
    return 500.00;  // $500.00 after fully depreciated
  else {
    float value = getOriginalPrice();
    for (int y = 0; y < yr; y++) {
      value = value - (getOriginalPrice() / getLife());
    }
    return value;
  }
}
```

Specialization : Inheritance

Imagine that *Tax* and *Value* were calculated differently for different types of land vehicles.

```
// Car.h
#include "LandVehicle.h"

class Car : public LandVehicle {
 public:
   Car(float price) : LandVehicle(4, 2, price) {}
};
```

```
// Bicycle.h
#include "LandVehicle.h"

class Bicycle : public LandVehicle {
 public:
   Bicycle() : LandVehicle(2, 2, 700.0) {}
   float getTax() { return 0.0; }
   float getValue(int yr) {
     if (yr == 1) return 500.0;
     else if (yr == 2) return 400.0;
     else return 300.0;
   }
};
```

```
// Motorcycle.h
#include "LandVehicle.h"

class Motorcycle : public LandVehicle {
 public:
   Motorcycle(float price) : LandVehicle(2, 2, price){}
};
```

Using the Specialized Classes

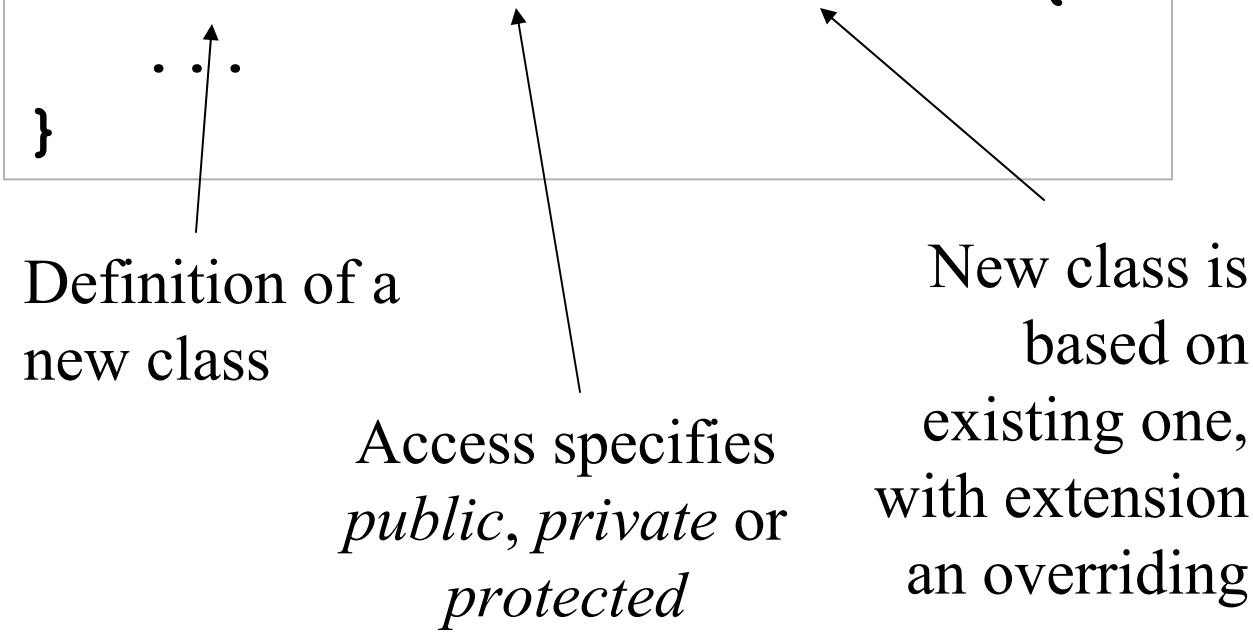
```
#include <iostream>
#include <iomanip>
#include "LandVehicle.h"
#include "Car.h"
#include "Bicycle.h"
#include "Motorcycle.h"

void print(LandVehicle& v);

void main() {
  // Define some vehicles
  LandVehicle v;
  Motorcycle m(30000.0);
  Car c(18000.0);
  Bicycle b;

  print(v);
  print(m);
  print(c);
  print(b);
}

void print(LandVehicle& v) {
 cout << setw(15) << "Axels = "
      << v.getAxels() << endl;
 cout << setw(15) << "Wheels = "
      << v.getWheels() << endl;
 cout << setw(15) << "Price = "
      << v.getOriginalPrice() << endl;
 cout << setw(15) << "Value = "
      << v.getValue(3) << endl;
}
```

Using the Specialized Classes

```
[bvelez@amadeus] ~/spring2001/icom4015/lec14 >>./main
Name: LandVehicle
        Axels = 2
       Wheels = 4
        Price = 1000
        Value = 700
Name: Motorcycle
        Axels = 2
       Wheels = 2
        Price = 30000
        Value = 21000
Name: Car
        Axels = 2
       Wheels = 4
        Price = 18000
        Value = 12600
Name: Bicycle
        Axels = 2
       Wheels = 2
        Price = 700
        Value = 490
[bvelez@amadeus] ~/spring2001/icom4015/lec14 >>
```

Anatomy of a Class Definition with Inheritance

```
newclass : access baseclass {
    . . .
}
```



- Methods in new class with same signature as base class "override" the base's method
- Methods in new class with same name only (different arguments) simply overload the name - extend functionality

Intro to Abstract Classes

```
// Vehicle.h

class Vehicle {
public:
  // No constructors
  // No data member access methods

  // Calculate other Vehicle properties
  virtual int   getLife()         = 0;
  virtual float getTax(int yr)    = 0;
  virtual float getValue(int yr) = 0;
};
```

```
#include ”Vehicle.h"

class LandVehicle : public Vehicle {
    ...
}

class airVehicle : public Vehicle {
    ...
}

class waterVehicle : public Vehicle {
    ...
}
```

The Class Hierarchy