Classes

Dr. Raman Adaikkalavan and Hakimzadeh
CS & Informatics, IUSB
Learning Objectives

• Classes
  – Understanding the basic concepts of object orientation.
    • Encapsulation, inheritance, polymorphism
  – Defining, member functions
  – Public and private members
  – Accessor and mutator functions
  – Structures vs. classes
What is Object-Oriented Paradigm?

MODELING THE REAL WORLD

- Humans' natural ability to classify, generalize and create abstractions in order to model the world.

- View the world as a series of entities and the interaction among them.

- Problems are solved through the interaction among a number of autonomous and cooperative objects.
What is Object-Oriented Paradigm?

- View the **Code and Data** as an *atomic and non-decomposable unit*.

- Traditional programming languages deliver modularization by the use of procedures and functions.
What is Object-Oriented Paradigm?

- An object is defined as a set of **built-in and/or user-defined data types along with a set of operations** which manipulate them.

- Object = Data + Code

- The functions which surround the data structure provide a **robust interface to the clients** of that data.

- Simplifies **debugging, maintenance and testing**.

- The applications or clients which use object will not be effected by the possible changes in the implementation of the object, as long as the interface or behavior of the object remains constant.
The 3 Pillars of Object-Oriented Paradigm

1. Encapsulation
2. Inheritance
3. Polymorphism
Encapsulation

- Encapsulation, makes large projects intellectually manageable.

- Encapsulation is achieved by tightly coupling the data structure and its related methods (functions) hence viewing them as an atomic unit.

- Access to the data is only possible through the public interface provided. Encapsulation hides the details of the object from the client.
Inheritance

- One of the most important aspect of object oriented systems.

- A new class can inherit reuse the behavior and structure of previously defined classes.

- The Sub-Classes (Derived Classes) can extend or add more functionality (i.e. methods and instance variables) to the base class.

- Inheritance is a natural tool to express relationships such as classification, specialization, generalization, evolution and approximation.

(c) Copyright 2007 - 2014, H. Hakimzadeh
Polymorphism comes from the Greek: "having many shapes".
Polymorphism

Two methods for achieving Polymorphism:

- Overloading
- Overriding

More later.....
Class Definitions

- Defined similar to structures
- Example:

```cpp
class <class name>  //name of new class type
{
    // Class members:
    // Data
    // Functions / methods
};
```

**Encapsulation**

Bring together data and operations, but keep "details" hidden from the user!
Example of Encapsulation:

class Car
{
private:  // Private Data
    string color;
    int year;
    string make;
    string model;

public:   // Public Methods
    void PrintColor(void);
    void SetColor(string clr);
};
Public and Private Members

• Data in class almost always designated private in definition!
  – Upholds principles of OOP
  – Hide data from user
  – Allow manipulation only via operations
    • Which are member functions

• Public items (usually member functions) are "user-accessible"
Example of Encapsulation:

class Car
{
    private:// Private Data
    string color;
    int year;
    string make;
    string model;

    public: // Public Methods
    
    void PrintColor(void);
    void SetColor(string clr);
};

(c) Copyright 2007 - 2014, H. Hakimzadeh
Functions of a class

```cpp
void Car::PrintColor(void)
{
    cout << "The Color = " << color;
}

//----------------------------

void Car::SetColor(string clr)
{
    color = clr;
}
```

Scope resolution operator:
Specifies what class the function definition comes from.
Example of Encapsulation:

```cpp
void main()
{
    Car MyCar ; // MyCar is an object of type Car
    MyCar.SetColor("GREEN");
    MyCar.PrintColor();
    cout << MyCar.color <<endl; // Note that this is an error, since we don't have
    // direct access to the Color attribute.
}
```

(c) Copyright 2007 - 2014, H. Hakimzadeh
Accessor and Mutator Functions

- Object needs to "do something" with its data
  - **Accessor** member functions
    - Allow us to access the object’s data
    - Also called "get member functions"
  - **Mutator** member functions
    - Allow us to change the object’s data
Separate Interface and Implementation

• User of class need not know the details of how the class is implemented
  – Principle of OOP → encapsulation

• User only sees the class specification:
  – Also known as "interface" for the class
    • The public member functions

• Implementation of the class hidden from the user
  – Member function definitions is kept elsewhere
  – User need not see them
Structures versus Classes

• Structures
  – Typically all members public
  – No member functions (perception)

• Classes
  – Typically all data members private
  – Interface member functions public

• Technically, same idea
  – Perceptionally, very different mechanisms
Inheritance

- Important aspect of object oriented systems.
- A new class can inherit **reuse the behavior and structure of previously defined classes.**
- The Sub-Classes (Derived Classes) can **extend or add more functionality** to the base class.
- Promotes code reuse

(c) Copyright 2007 - 2014, H. Hakimzadeh
Example of Inheritance:

// The SUPER_CAR class Inherits from the CAR class
// and add extends the class by adding a new attribute
// and two new methods.

class Super_Car : public Car // Inherits from the CAR class
{
private:
    string SuperAttribute;
public:
    string GetSuperAttribute() {
        return SuperAttribute;
    }

    void SetSuperAttribute(string SuperAttr) {
        SuperAttribute = SuperAttr;
    }
};
Example of Inheritance:

```cpp
#include <iostream>
#include <string>
#include "Car.h"
#include "Super_Car.h"
using namespace std;

void main()
{
    Car MyCar;    // MyCar is an object of type Car
    MyCar.SetColor("GREEN");
    MyCar.PrintColor();

    Super_Car BatMobil; // BatMobile is an object of type SuperCar
    BatMobil.SetColor("BLACK");
    BatMobil.SetSuperAttribute("Goes Real Fast!!");
    BatMobil.PrintColor();
}
```
Polymorphism:

- Polymorphism is the ability to associate many different meanings to a function name.

- Early vs. Late binding
Polymorphism:

- Polymorphism via **early binding**

- This is known as **overloading**
Example of Polymorphism:

Imagine in our Car class, we have two methods for setting the color:

```cpp
class Car {
private:// Private Data
    string color;
    int year;
    string make;
    string model;

public:// Public Methods
    void PrintColor(void);
    void SetColor(string clr);
    void SetColor(void);
};
```

The first `SetColor()` method takes an argument and sets the color or the car to that argument.
Example:
`SetColor(“GREEN”)`

The second `SetColor()` method takes NO arguments and sets the color or the car to a default value “YELLOW”.
Example:
`SetColor()`
Polymorphism - Overloading

```cpp
void Car::PrintColor(void)
{
    cout << "The Color = " << color << endl;
}

//------------------------------
void Car::SetColor(string clr) // Overloaded method
{
    color = clr;
}

//------------------------------
void Car::SetColor() // Overloaded method
{
    color = "YELLOW";
}
```
Thinking Objects

• Focus for programming changes
  – Before → algorithms center stage
  – OOP → data is focus

• Algorithms still exist
  – They simply focus on their data
  – Are "made" to "fit" the data

• Designing software solution
  – Define variety of objects and how they interact
Summary 1

• Structure is collection of different types

• Class is used to combine data and functions into single unit -> object

• Member variables and member functions
  – Can be public → accessed outside class
  – Can be private → accessed only in a member function’s definition

• Class and structure types can be formal parameters to functions
Summary 2

• C++ class definition
  – Should separate two key parts
    • Interface: what user needs
    • Implementation: details of how class works