Encapsulation

Dr. Hakimzadeh
CS & Informatics, IUSB
Learning Objectives

• Understanding and Expanding Encapsulation
  – Constructors
  – Destructors
  – And a few more things you can do with classes..
    • Inline functions
    • Static variables and functions
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.
Class Definitions

- Defined similar to structures
- Example:

```java
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:

```cpp
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
Adding more capability to a class

- Constructors
- Function overloading
- Destructors
- Inline functions
- Static members
- Friend functions
- Operator overloading

Later..
Constructor

- A class function which has the responsibility to initialize the objects of that class

- When an object of the class is instantiated (declared), the constructor automatically gets called.
Constructor Functions...

- Constructor should be public!

- A constructor function has the same name as the CLASS itself.

- A constructor can not return a value (no data type is associated with a constructor function).

- Constructor functions can be overloaded!
Example Constructor:

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

    Car();//Constructors for the class
    Car(string the_color, int the_year,
         string the_make, string the_model);

    void PrintColor(void);
    void SetColor(string clr);
};

(c) Copyright 2007 - 2014, H. Hakimzadeh
// Constructor
// Initialize the class variables
Car::Car()
{
    color = "No Color";
    year = 0;
    make = "No_make";
    model = "No_Model";
}

Car::Car(string the_color, int the_year, string the_make, string the_model)
{
    color = the_color;
    year = the_year;
    make = the_make;
    model = the_model;
}
Another method for creating a constructor

- The use of initialization section:

```cpp
Car::Car(string the_make, string the_model)
    : color("no_color"), year(1900), make(the_make), model(the_model)
{
    // body intentionally left empty!
}
```
void main()
{

    Car MyNewCar1;                   // No parameters
    MyNewCar1.PrintColor();

    MyNewCar2.PrintColor();

    Car MyNewCar3 ("Toyota", "Camry");
    MyNewCar3.PrintColor();

}
Watch out for...

☐ Be careful not to use parentheses with constructors with no parameters.

```c
Car MyNewCar1(); // Compiler will complain!
MyNewCar1.PrintColor();

Car MyNewCar1; // No parameters
MyNewCar1.PrintColor();
```
Destructors

- Function that is called when the object is deleted or goes out of scope.

- `Car::~Car();` // Destructor
Inline Functions

- Complete definition of the member function is provided within the class definition.

- Compiler treats inline functions differently. The code of the inline function is replicated everywhere it is called in the program. (this saves the overhead of function call)

- More efficient.
Static Members

- Typically each object has its own set of members (variables or functions)

- Statics members all the objects to share one variable/function among all the objects of the same class.

- Only static member functions can access static variables of a class.
class Box {
public:
    static int objectCount;
    static bool debug;

    // Constructor definition
    Box(double l = 2.0, double w = 2.0, double h = 2.0) {
        cout << "Constructor called." << endl;
        length = l;
        breadth = b;
        height = h;

        // Increase every time object is created
        objectCount++;
        if (debug) cout << "Debug is on" << endl;
    }

double Volume() {
    return length * width * height;
}

    static void debug_on() { debug = true; }

private:
    double length;       // Length of a box
    double width;        // width of a box
    double height;       // Height of a box
};

// Initialize static member of class Box
int Box::objectCount = 0;
bool Box::debug = true;

int main(void) {
    Box::debug_on();
    Box Box1(3.3, 1.2, 1.5);    // Declare box1
    Box Box2(8.5, 6.0, 2.0);    // Declare box2

    // Print total number of objects.
    cout << "Total objects: " << Box::objectCount << endl;

    getchar();
    return 0;
}
Adding more capability to a class

- Constructors
- Function overloading
- Destructors
- Inline functions
- Static members
- Friend functions
- Operator overloading