

Object Oriented Design

- Object oriented design (OOD) breaks problem into objects in a top-down process
- Objects are self contained entities containing both member variables and member functions
 - Member Variables: Private member data variables
Also called: Properties, Attributes, Internal states
 - Member Functions: Public member functions
Also called: Methods, Operations, Instructions
- An Abstract Data Type (ADT) defines a class of objects from which Instances (objects) may be created
- C++ Classes are used to define an Abstract Data Types (ADT) from which Objects of that class may be created (declared)

Abstract Data Type

- Abstraction
 - Separating the essential specifications from implementation details
 - Specification of *What To Do* (Not how to do it)
- An ADT (C++ class) combines both data and operations in a single conceptual unit
 - Data Abstraction: Separation of data type properties from implementation details
 - Control Abstraction: Separation of logical properties of an action from implementation

ADT - Design Example

TYPE

ChipType

DOMAIN

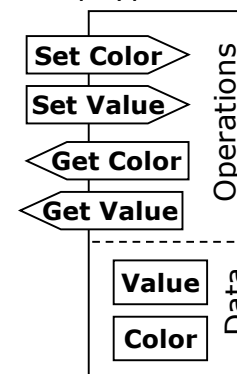
Each ChipType is a colored chip describing it's color and chip value

OPERATIONS

Set chip color
Set chip value
Get chip color
Get chip value

ADT - Implementation Example

ChipType Class



C++ Class

- The **class** keyword makes C++ an **object oriented programming language**
- Similar to struct in that user defined data types can be created, but additionally allows you to **encapsulate** functions.
- The C++ **class** is the structured type used for implementing Abstract Data Types
- Having “class” makes you an advanced and skilled programmer

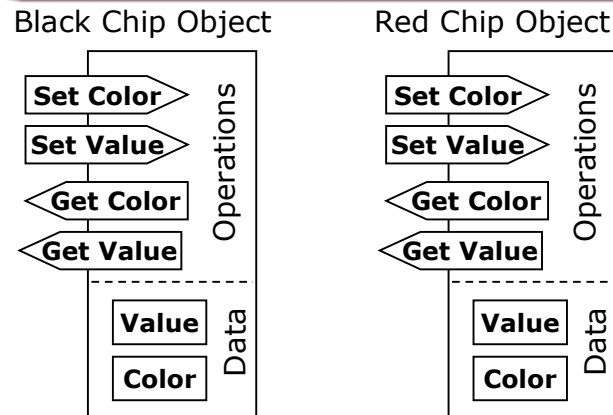
Copyright © 2003 R.M. Laurie 5

Class Definitions

- **Class:** Structured type for creating ADT
- **Object:** (class Instance)
 - A declared identifier of type class
- **Class Members:** Component of a class
 - **Member Variables:** Data Components
 - **Member Functions:** Functions that manipulate member variables.
- **Client:** Software that declares and uses objects of a particular class.

Copyright © 2003 R.M. Laurie 6

ADT - Implementation Example



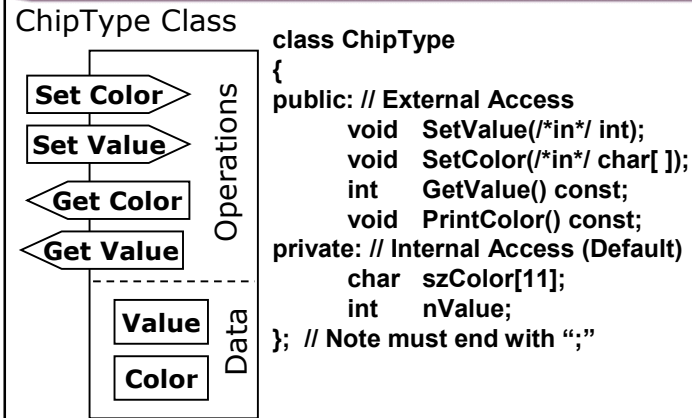
Copyright © 2003 R.M. Laurie 7

Class Members

- **Private Class Members:**
 - Data or member function is only accessible within the class not client
 - Hidden members = Black Box Concept
 - The default
 - Best to place after public functions for readability
- **Public Class Members:**
 - Member is accessible by client
 - Best to restrict to member functions not data

Copyright © 2003 R.M. Laurie 8

Class Type Declaration



Copyright © 2003 R.M. Laurie 9

Class Encapsulation

- **Information Hiding**
 - The “Black Box Concept
 - Hides Implementation Details
- **Abstraction Barrier:**
 - private: protects data and functions from being accessed by client
 - Information Hiding
- The name of a class member has **class scope** and is local to the class
- Dot “.” operator used to select member just like as used with record (data structure)

Copyright © 2003 R.M. Laurie 10

Class Object Declaration

```

int main(void)
{
    ChipType RedChip;
    int nChip;
    char szCol[10] = "Red";
    cin >> nChip;
    RedChip.SetValue(nChip);
    cout << RedChip.GetValue() << endl;
    RedChip.SetColor(szCol);
    RedChip.PrintColor();
    return 0;
}
  
```

Copyright © 2003 R.M. Laurie 11

Class Operators

- Dot “.” operator used to select member just like as used with records (data structure)


```
BlueChip.GetValue( )
```
- **Scope resolution operator “::”** is used to associate member function with class


```
long ChipCountType::GetCount() const
{
    return(lnCount);
}
```
- **Assignment “=”** used to copies data members from one object to another of the same class type.


```
RedChip = BlueChip;
```

Copyright © 2003 R.M. Laurie 12

What is Missing?

- The Member Functions
- Defined similar to any function however uses the scope resolution operator “ :: ”

```
void ChipType::SetValue(/*in*/ int nChipValue)
{
    nValue = nChipValue;
}
```

Copyright © 2003 R.M. Laurie 13

Member Functions Definitions

```
void ChipType::SetValue(/*in*/ int nChipValue)
{
    nValue = nChipValue;
}
int ChipType::GetValue() const
{
    return(nValue);
}
void ChipType::SetColor(/*in*/ char szChipColor[])
{
    strcpy(szColor, szChipColor);
}
void ChipType::PrintColor() const
{
    cout << szColor;
}
```

Copyright © 2003 R.M. Laurie 14

Can Place All In One File - 1

```
#include <iostream>
#include <cstring>
using namespace std;
class ChipType
{
public: // External Access
    void SetValue(/*in*/ int);
    void SetColor(/*in*/ char[ ]);
    int GetValue() const;
    void PrintColor() const;
private: // Internal Access (Default)
    char szColor[11];
    int nValue;
};
```

Copyright © 2003 R.M. Laurie 15

Can Place All In One File - 2

```
int main()
{
    ChipType RedChip;
    int nChip;
    char szCol[10] = "Red";
    cin >> nChip;
    RedChip.SetValue(nChip);
    cout << RedChip.GetValue() << endl;
    RedChip.SetColor(szCol);
    RedChip.PrintColor();
    cout << endl;
    return 0;
}
```

Copyright © 2003 R.M. Laurie 16

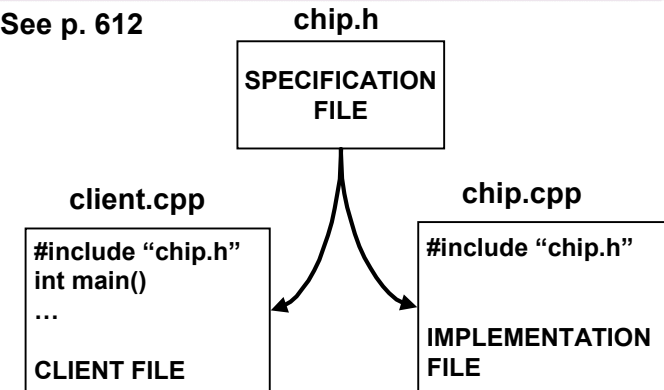
Can Place All In One File - 3

```
void ChipType::SetValue(*in* int nChipValue)
{
    nValue = nChipValue;
}
int ChipType::GetValue() const
{
    return(nValue);
}
void ChipType::SetColor(*in* char szChipColor[])
{
    strcpy(szColor, szChipColor);
}
void ChipType::PrintColor() const
{
    cout << szColor;
}
}
```

Copyright © 2003 R.M. Laurie 17

Or Three Separate Files

See p. 612



Note: Requires creation of project in Visual C++

Copyright © 2003 R.M. Laurie 18

Specification File chip.h

```
class ChipType
{
public: // External Access
    void SetValue(*in* int);
    void SetColor(*in* char[ ]);
    int  GetValue() const;
    void PrintColor() const;
private: // Internal Access (Default)
    char szColor[11];
    int  nValue;
};
```

Copyright © 2003 R.M. Laurie 19

Implementation File chip.cpp

```
#include <iostream>
#include <cstring>
#include "chip.h"
using namespace std;
void ChipType::SetValue(*in* int nChipValue)
{
    nValue = nChipValue;
}
int ChipType::GetValue() const
{
    return(nValue);
}
void ChipType::SetColor(*in* char szChipColor[])
{
    strcpy(szColor, szChipColor);
}
void ChipType::PrintColor() const
{
    cout << szColor;
}
}
```

Copyright © 2003 R.M. Laurie 20

Client File client.cpp

```
#include <iostream>
#include "chip.h"
using namespace std;
int main()
{
    ChipType RedChip;
    int nChip;
    char szCol[10] = "Red";
    cin >> nChip;
    RedChip.SetValue(nChip);
    cout << RedChip.GetValue() << endl;
    RedChip.SetColor(szCol);
    RedChip.PrintColor();
    cout << endl;
    return 0;
}
```

50
Red

Copyright © 2003 R.M. Laurie | 21

Class Constructors

- Class constructors are usually used
- Constructors guarantee the class object will be initialized
- **Syntax:** The class constructor simply uses the same name as the class type
- Multiple constructors can be used such that different parameter types can be utilized
- **Overloading** is implementing by creating different functions with same identifier that have different parameter lists
- The default constructor has no parameters

Copyright © 2003 R.M. Laurie | 22

Specification File chip.h

```
class ChipType
{
// External Access
public:
    int    GetValue() const;
    void  PrintColor() const;
    ChipType(*in* int, /*in*/ char[]);
    ChipType();
// Internal Access (Default)
private:
    char  szColor[11];
    int   nValue;
};
```

Copyright © 2003 R.M. Laurie | 23

Implementation File chip.cpp

```
#include <iostream>
#include <cstring>
using namespace std;
#include "chip.h"
int ChipType::GetValue() const { return(nValue); }
void ChipType::PrintColor() const { cout << szColor; }
ChipType::ChipType(int nChipValue, char szChipColor[])
{
    nValue = nChipValue;
    strcpy(szColor, szChipColor);
}
ChipType::ChipType() {
    nValue = 0;
    strcpy(szColor, "Dummy");
}
```

Copyright © 2003 R.M. Laurie | 24

Client File client.cpp

```
#include <iostream>
#include <cstring>
using namespace std;
int main(void)
{
    ChipType RedChip(50, "Red");
    ChipType White;
    cout << endl << RedChip.GetValue() << endl;
    RedChip.PrintColor();
    cout << endl << White.GetValue() << endl;
    White.PrintColor();

    return 0;
}
```

```
50
Red
0
Dummy
```

Copyright © 2003 R.M. Laurie | 25

Yes you can, Arrays of Class Objects

```
#include <iostream>
#include "chip.h"
using namespace std;
int main()
{
    char szChipColors[6][11] =
        {"Black", "Blue", "Red"};
    int nChipValue[3] = {5, 20, 50}, nl;
    ChipType Chips[3];
    for(nl = 0; nl < 3; nl++)
    {
        Chips[nl].SetCount(nChipValue[nl]*10);
        cout << Chips[nl].GetCount() << " ";
        Chips[nl].SetColor(szChipColors[nl]);
        cout << " " << Chips[nl].PrintColor() << endl;
    }
    return 0;
}
```

```
50 Black
200 Blue
500 Red
```

Copyright © 2003 R.M. Laurie | 26

Guidelines for using Classes

- Compiling and Linking Multiple Files
See p. 613
- Guidelines for using Class Constructors
See p 620
- Testing and Debugging Classes
See p 644

Copyright © 2003 R.M. Laurie | 27