

## Object Oriented Design

- Different then structured (procedural) languages in that program code does not have to be executed sequentially but can be event driven.
- Used to write Windows type programs using the mouse buttons and keyboard click events.
- Object Oriented Programming Languages: C++, Java, Delphi, and Visual Basic (sort of)

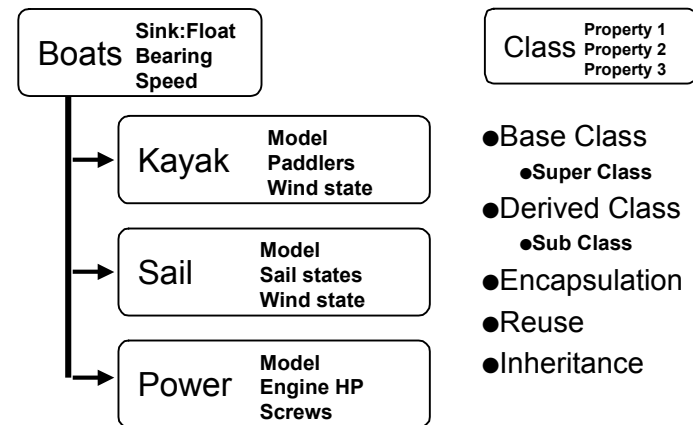
## Object Oriented Programming

- Object contains:
  - Properties: Variables, Member Data, Attributes
  - Methods: Member Functions, Instructions
- Encapsulation: Describes Object contains both properties and methods. Everything it needs to exist as an object.
- Instance: Occurrence of a class (Object)
- Message: Command for an object passed as argument to member function

## Object Oriented Programming

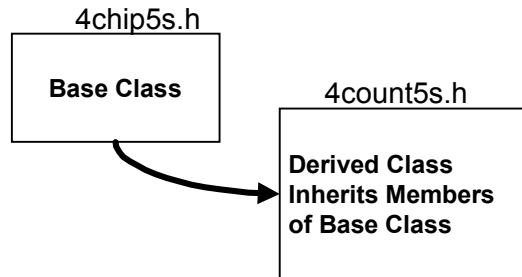
- **Class**: Objects with common properties
- **Derived Class**: Class with more specific properties. Also includes **Base Class** properties
- **Inheritance**: Derived class possesses (inherits) all properties of base class.
- **Reuse**: Classes can be reused because self contained.

## ADT (C++ Class) Hierarchy



## Inheritance and Derived Classes

- Each Derived Class (Sub Class) Inherits the properties of the Base Class (Super Class)



Copyright © 2003 R.M. Laurie 5

## Compiling and Linking Multiple Files

4client5.cpp

```
Client File
#include "4count5s.h"
int main()
...
```

4count5s.h

```
Specification File
#include "4chip5s.h"
```

4count5i.cpp

```
Implementation File
#include "4count5s.h"
```

4chip5s.h

```
Specification File
```

4chip5i.cpp

```
Implementation File
#include "4chip5s.h"
```

Copyright © 2003 R.M. Laurie 6

## 4count5s.h Specification File

```
#include "4chip5s.h"
class ChipCountType : public ChipType
// Declaring ChipType Public allows you to
// access member functions
{
public: // External Access
    void SetCount(int);
    int GetCount() const;
    ChipCountType(*in*/ int, /*in*/ char[]);
    ChipCountType();
    void PrintColor() const; // Overriding
private: // Internal Access (Default)
    int nCount;
};
```

Copyright © 2003 R.M. Laurie 7

## 4chip5s.h Specification File

```
class ChipType
{
public: // External Access
    void SetValue(*in*/ int);
    void SetColor(*in*/ char[]);
    int GetValue() const;
    void PrintColor() const;
    ChipType(*in*/ int, /*in*/ char[]);
    ChipType();

private: // Internal Access (Default)
    int nValue;
    char szColor[11];
};
```

Copyright © 2003 R.M. Laurie 8

## 4count5i.cpp Implementation - 1

```

#include <iostream>
#include <cstring>
#include "4count5s.h"
using namespace std;
void ChipCountType::SetCount(int nI)
{
    nCount = nI;
}
int ChipCountType::GetCount() const
{
    return(nCount);
}

```

Copyright © 2003 R.M. Laurie 9

## 4count5i.cpp Implementation - 2

```

ChipCountType::ChipCountType(int nChipValue,
    char szChipColor[]) : ChipType(nChipValue, szChipColor)
{
    nCount = 0;
}
ChipCountType::ChipCountType() : ChipType()
{
    nCount = 0;
}
void ChipCountType::PrintColor() const
{
    // Overriding Parent Member Function
    ChipType::PrintColor();
    cout << endl;
}

```

Copyright © 2003 R.M. Laurie 10

## 4chip5i.cpp Implementation File - 1

```

#include <iostream>
#include <cstring>
#include "4chip5s.h"
using namespace std;
void ChipType::SetValue(/*in*/ int nChipValue)
{
    nValue = nChipValue;
}
void ChipType::SetColor(/*in*/ char szChipColor[])
{
    strcpy(szColor, szChipColor);
}
int ChipType::GetValue() const
{
    return(nValue);
}

```

Copyright © 2003 R.M. Laurie 11

## 4chip5i.cpp Implementation File - 2

```

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 12

## 4client5.cpp Client File - 1

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "4count5s.h"
int main()
{
    char szChipColor[6][11] = {"Black", "Blue",
        "Red", "Green", "Silver", "Gold"};
    int nChipValue[6] =
        {5, 20, 50, 100, 500, 1000};
    int nl;
    ChipCountType Chips[6];
    for(nl = 0; nl < 6; nl++)
    {
        cout << Chips[nl].GetCount() << ' '
            << ' ' << Chips[nl].GetValue() << ' ';
        Chips[nl].PrintColor(); }
}
```

Copyright © 2003 R.M. Laurie 13

## 4client5.cpp Client File - 2

```
for(nl = 0; nl < 6; nl++)
{
    Chips[nl].SetValue(nChipValue[nl]);
    Chips[nl].SetColor(szChipColor[nl]);
    Chips[nl].SetCount(1200-nl*100);
    cout << setw(5) << Chips[nl].GetCount()
        << " X $" << setw(5)
        << Chips[nl].GetValue() << " = $"
        << setw(8) << ( Chips[nl].GetValue() *
            Chips[nl].GetCount() ) << " : ";
    Chips[nl].PrintColor();
}
return 0;
}
```

Copyright © 2003 R.M. Laurie 14

## Output

```
0 0 Dummy
0 0 Dummy
0 0 Dummy
0 0 Dummy
0 0 Dummy
0 0 Dummy
1200 X $ 5 = $ 6000 : Black
1100 X $ 20 = $ 22000 : Blue
1000 X $ 50 = $ 50000 : Red
900 X $ 100 = $ 90000 : Green
800 X $ 500 = $ 400000 : Silver
700 X $ 1000 = $ 700000 : Gold
```

Copyright © 2003 R.M. Laurie 15

## Member Function Overriding

- Overriding is reimplementing a member function inherited from the base class (super class, parent class)
- Example was the member function `void ChipCountType::PrintColor() const`

```
{
    // Overriding Parent Member Function
    ChipType::PrintColor();
    cout << endl;
}
```
- Note: Specification file is no different

## Inheritance Constructors

Inheritance: Specify base class prior to actual parameter list

```
ChipCountType::ChipCountType(int
nChipValue, char szChipColor[])
: ChipType(nChipValue, szChipColor)
{
nCount = 0;
}
```

Constructor Initializer

Inherited Class Type

## Object Oriented Design

- Identify the objects
  - Look at problem domain (nouns & verbs)
  - Nouns suggest Objects
  - Verbs suggest Operations
- Determine the relationships
  - Identify inheritance for objects
  - Identify composition for objects
- Design the Drivers
  - Top level algorithm that links objects
  - GUI and command processor

## Design Implementation

- Choose Suitable Data Representation
  - Use a built-in (simple) data type
  - Use an existing ADT (Class)
  - Create a new ADT(Class)
- Create algorithms to implement abstract operations
- Test and Debug ADT from derived to base classes.