Object-oriented Programming

⇒ Support Characteristics of Language

⇒ Exam from a syntax & semantic aspect

⇒ Added features

⇒ Using the object concept

⇒ Analysis - Chpt 21

⇒ Design - Chpt 22
Language Features

Deal with C++ as example

- **class** - starts definition of object
  - like C *structure* or Pascal *record*
  - Unlike those it can contain both data and functions

- Data and functions can be available or hidden
  - syntax - *private*, *protected* or *public*
  - allows for a controlled information interface

- Data - provides both simple and complex structures

- Functions
  - function name overloading - more effective function calls
  - operator functions - enables expressions and assignments
  - pointers to functions
C++ Language Features (cont’d)

Inheritance - base and derived classes

⇒ Complete flexibility to share information - *private*, *protected*, *public* within classes and at the interior level.
⇒ Enables messages between object
⇒ Enables utility operations without exterior modification - black box processing

⇒ Implements decomposition
  ⇒ Class hierarchy
  ⇒ Sharing so that base has general information
  ⇒ Derived class can have specific information

⇒ Enables significant reuse and upgrade
C++ Language Features (cont’d)

Function calling flexibility & name overloading

⇒ Name overloading between class levels
  ⇒ keeps process modularity
  ⇒ used for adding or overriding activity

⇒ Polymorphism - dynamic (run-time) binding
  ⇒ can easily add sub class with minimum code changes
  ⇒ duplicate features of other classes

⇒ Uniform exception handling
  ⇒ cause at any level & handle at another level
  ⇒ multiple handling with flexible information transfer and actions

⇒ Properties - adds effective design & runtime data control
Polymorphism example

⇒ Draw for various objects

// object definitions

class Shapes { public:
    virtual void Draw(void);
    // other stuff in class
};
class Rectangle public: Shapes { public:
    void Draw(void);  // draw rectangle
    // other stuff
    private:
        Point upper_lf, lower_rt;
};
class Circle public: Shapes { public:
    void Draw(void);  // draw circle
    // other stuff
    private:
        Point center;
        real radius;
};

// code for drawing shapes

void DrawShape(Shape& aShape)
{
    // other code
    Draw(aShape);
}

When different derived class parameters are passed dynamic binding make the correct call
C++ Features (cont’d)

⇒ Templates
  ⇒ Create easily reusable code
  ⇒ Provide standardization
  ⇒ Minimize errors and debugging

⇒ Standard Template Library
  ⇒ Range of containers - vector, list, deque, set, string, stack, queue, etc.
  ⇒ Range of algorithms - sort, accumulate, swap, find, etc.
  ⇒ Useable with standard and structure defined variable

⇒ Complex structures need to define component operators functions - >, =>, <, ==, etc.

If your not using STL in your coding your working way to hard.
Object use in Software Engineering

⇒ Template Libraries

⇒ Component Object Modules (COM)
  ⇒ Create COM-based classes and components
    ⇒ Based on the concept of Client & Server
    ⇒ Automation Servers, OLE, ActiveX controls, etc.
    ⇒ Extended to DCOM - Distributed (network) COMs

⇒ COM is both a spec and an implementation
  ⇒ Spec - define how objects are created & their communications
  ⇒ COM implementation is in libraries which support core services.
COM Interface

Interface:
1. Global Unique Identifier GUID
2. Implement IUnknown
3. Once published its immutable.
   Can be added to but not changed
4. Language Independent
5. Interfaces are not object; they provide access to objects
6. Interfaces can be redirected without client or server knowledge

Interface - IUnknown contains the following routines
- QueryInterface - provides pointers to other interfaces the object support
- AddRef and Release - Simple counting methods which keep track of lifetime information.
COM Interface (cont’d)

- Interface
- COM Object
- Interface implementation
- vtable
- Pointer to function 1
- Pointer to function 2
- Pointer to function 3
- Interface functions
COM Interface (cont’d)

Structure for In-process Server

- Client
  - COM Object
  - Inprocess Proxy

Structure for Out-of-process server

- Network server
  - DCom
  - Stub
  - Inprocess Object

- RPC - remote procedure call
COM Characteristics

⇒ Marshalling
  ⇒ In-proxy server - takes interface pointer and makes a proxy pointer for client
  ⇒ Transfers arguments by pushing on a stack and transferring stack.

⇒ Aggregation - a COM object makes use of another COM object.
  ⇒ done without knowledge of client.
  ⇒ COM object must enable himself to look like client.
  i.e., use CoCreateInstance or CoCreateInstanceEx
COM Features & Req.

⇒ Query features
  ⇒ Client can always query interface to determine services
  ⇒ Objects allow clients to request known interfaces
  ⇒ With *IDispatch*, client can query about methods supported
  ⇒ Server has no expectations about client using the object

⇒ Two general characteristics
  ⇒ Controllers - request services and interact
    ⇒ Example - sending information to another client
  ⇒ Container - enters into client process to provide control, display, etc.
    ⇒ Example - button in your window
  ⇒ Provide general functionality
COM Implementation - Type libraries

Provide design and user interface info

⇒ Contain type information
  ⇒ properties, methods, arguments, structures (unions, records, alias, module data types, etc.)

⇒ includes the identifiers to CoClasses (CLSID - GUID), interfaces (IIDs), dispatch identifiers (dispID)

⇒ References to other type libraries

⇒ Registry information access

⇒ Information necessary to compile for use
Library tools interface - typical

- **TLIBIMP** - takes existing library and creates library interface file
  - TLB.cpp and TLB.h files for compilation
- **TRegSvr** - tool for registering services in Windows registry
- **Microsoft IDL compiler (MIDL)** compiles IDG scripts for header files
  - for use with MS Win32 SDK (Software Development Kit)
- **OLEView** - library browser tool - can be found on web
- **MKTYPLIB** - Compiler for ODL Script to create type libraries
  - for use with MS Win32 SDK

**Tools to create necessary libraries for your COM**

- Libraries created depend upon how your COM interface functions
- Simplify tasks of GUID, CoClass, define & update interface
Objects - Summary

Extremely powerful in code development

⇒ C++ with extensions - properties, function utilization, basic interface

⇒ Template - adds easy development for familiar functions

⇒ COM and related object development interfaces
  ⇒ extremely useful for developing complex interfaces
  ⇒ augments object (re)-use broadly
  ⇒ difficulty to use - improvements will come in time
  ⇒ security can be easily compromised - know your provider.
Object in Design

- attributes, operations, messages, events
  - object data & properties
  - methods

- Identifying the element in an object model
  - grammatical “parse” - noun & noun clauses
  - problem statement - specification
  - at least partly available
  - entities, things, occurrences, events, roles, organization units, places structures

SafeHome example:
- homeowner
- sensor system
- control panel
- installation
- system
  - role or external entity
  - entity
  - occurrence
  - thing
Identifying elements (cont’d) - attributes & operations

Attributes:

Sensor information = sensor type + sensor number + location + alarm characteristics

Alarm characteristics = alarm type + delay time

Activation/deactivation information = master password + number of tries + user password(s)

Identification information = system ID + system status + phone number

Display panel attributes = window + menu + buttons + icons + controls

Operations:

Display, query, call, activate/deactivate, event handlers, etc.
Management of object-oriented projects

Similar to other design activities in projects

⇒ Establish process “framework” for project
⇒ Examine for reusability features
⇒ Develop resource effort and time estimates
⇒ Establish deliverables and milestones
⇒ Do any risk management needed
⇒ Setup tracking, monitoring and control project structure
Typical process sequence for an OO project

See Fig. 20.11
Typical of the sequence and iteration in a project

⇒ Better than the figures in chapters 1 & 2.
⇒ At an increment - baseline and release.
⇒ Tracking consists of dealing with the milestones in the stages
Object oriented analysis

Begin with problem definition

⇒ Define functional and operational requirements

⇒ Describe end user

⇒ Provide a basis for validation testing

⇒ In the process analyze using scenarios
  ⇒ Responsibility - collaborator modeling
    ⇒ Similar to service - viewpoint
Responsibility - collaborator modeling

- **Responsibilities**
  - attributes & operation relevant to the class
  - “anything the class does or knows”
  - break up - should be shared
  - “has a” and “is a” relate to sub or inherited class

- **Collaborators**
  - classes which provide the class with information needed
  - “request for information or action”

- Sec. 21.5 illustrates for SafeHome project.

<table>
<thead>
<tr>
<th>Class name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Class characteristics:</td>
<td></td>
</tr>
<tr>
<td>Responsibilities:</td>
<td>Collaborators:</td>
</tr>
</tbody>
</table>
Using the object model of chapter 21

- CRC & attribute, etc.
- Message Design
- Class & Object Design
- Subclass Design

User orientation

attributes, operation, collaborators

CRC Cards

Object Relationship Model

Object Behavior Model
Unified Modeling “Language”

⇒ Systems design
  ⇒ focuses on the software architecture
  ⇒ data entities, data flow and control flow relationships

⇒ Object design
  ⇒ focuses on the objects and their interactions - collaborations
  ⇒ detailed specification of attribute, messages and procedural structure

⇒ Concurrency
  ⇒ Which classes can run “simultaneously” with minimal or controlled interaction
Design Process

Illustrated in section 22.4 for SafeHome Project