Object-oriented programming:  
Role-based design

Topics:
- Definitions/examples of roles and collaborations
- How to assemble an application by composing collaborations

Recall from Button and ButtonListener
Reusable classes with dynamic behavior
- Issue: How to design Button w/o knowing class of objects that are to receive “button pressed” messages
- Solution:
  - Invented an interface class that declares nothing but a buttonPressed operation
  - Designed class Button with respect to this interface class
  - Requires:
    - Receiver classes to implement the interface
    - Receiver objects register with the Button object
Qualities of our design

Class **Button** very reusable

We can understand the **Button-ButtonListener** collaboration with little knowledge of **Button** and no knowledge of **DocManager**

– Example of **separation of concerns**

Clear mechanism for adapting arbitrary class to implement the **ButtonListener** interface

Program structured differently from programs that “compute a function”

---

Separation of Concerns

• De-couple interdependent parts of a problem
  – only way to master complexity
• Pull different parts of the problem apart
  – Appears simple, but deceptively hard
  – *Divide and Conquer* is a SoC strategy
• Partitioning strategies can make or break
  – time => schedules
  – concepts => user interface vs. algorithmic code
  – process => generating tasks
**Abstraction**

**Defn:** Process by which we identify the important aspects of a phenomenon and ignore its details

Example:
- STL class vector vs *bounded sequence*
- The latter is an abstraction

Two popular abstractions over objects:
- Class: set of objects with same characteristic
- Role/collaboration: set of objects that collaborate by sending messages back and forth to achieve some goal or purpose

---

**Collaboration**

**Defn:** Cohesive pattern of interaction (i.e., message exchange) among multiple objects to accomplish some goal or purpose

Examples:
- Message exchange between `printButton` and `fileMgr`
- `printButton` sends `buttonPressed` message to `fileMgr`

Note: Refers to set of *objects*, not classes
Role

**Defn:** Abstraction of an object in a particular collaboration

- Defines only that subset of the object’s characteristics that are meaningful to the collaboration
  - Thus, each object might play many different roles, one for each collaboration in which it is involved; and
  - A given role could be played by many different kinds of objects

Example: `fileMgr` object might:
- play the `ButtonListener` role with `printButton` object
- play the `Model` role with `viewPort` object

Roles allow to easily synthesize new collaborations

---

Role declarations

In C++, we declare a role using an interface class

Examples:

```cpp
class ButtonListener {
  public:
    virtual void buttonPressed( const string& )=0;
};

class ViewportModel {
  public:
    virtual bool retrieve( unsigned lineNumber,
                           string& line ) const =0;
};
```
Roles are Very Familiar

Roles Across Objects
Collaboration Synthesis

Role-based designs simplify the construction of new collaborations, a process known as synthesis.

Each object in a running program might play a role in multiple collaborations:
- If object O must play roles \{ R_1, R_2, \ldots, R_n \} then:
- The class of object O must implement the interfaces that specify these roles.

Application construction in this paradigm involves:
- Identifying the collaborations required to accomplish system goals
- Synthesizing custom classes to declare objects that play roles in these collaborations.

Example

Suppose we want to design a graphical browser that allows users to view and print documents:

```c
#include <FL/FL.H>
include "GUIManager.h"

GUIManager::GUIManager(): waitInterval (1) 
void GUIManager::selfcheckingInterval 
{ waitInterval = interval; }
void GUIManager::registerListener 
{ listeners.push_back(l); }
void GUIManager::runEventLoop() 
{ while (FL::page()) 
  FL::wait(waitInterval); }

viewPort object collaborates with fileMgr
fileMgr object that gets the data
DocMgr is involved in two collaborations playing a different role in each
printButton button object that sends messages to fileMgr to initiate printing
```
Example: Class ViewPort

class ViewPort : public Fl_Multiline_Output {
public:
    ...
    void registerModel( ViewPortModel* vpm )
    { model = vpm; }

protected:
    ViewPortModel* model;
    void resize();
};

On resize, retrieves lines from ViewPortModel.

Example: Class DocMgr

class DocMgr {
public:
    ...
    void printDocument() const;
    unsigned docSize() const;
    const string& docLine( unsigned ) const;
    void insertLine( unsigned, const string& );
    void appendLine( const string& );
    void deleteLine( unsigned );
};
Multiple inheritance

Used to “glue” two (or more) existing classes to form a new class.

Syntax:

class D : kind₁ B₁, kind₂ B₂ { ... };

Many interesting combinations:
- \( \text{kind}_1 = \text{kind}_2 = \text{public} \)
  - \( B_1 \) is abstract, \( B_2 \) is concrete
  - Both \( B_1 \) and \( B_2 \) are abstract classes
- \( \text{kind}_1 = \text{public}, \text{kind}_2 = \text{protected} \)

Class Diagram Notation

We use multiple inheritance to make an existing class implement an interface.
Example: Synthesis of Multiple Roles

class MyDocMgr : public DocMgr,
  public ButtonListener,
  public ViewportModel {

public:
  void buttonPressed( const string& s )
  { if(s == “print”) { DocMgr::printDocument(); } }

  bool retrieve( unsigned    lineNo,
                string&     line ) const
  { bool retVal = (lineNo < DocMgr::docSize());
    if (retVal) line = DocMgr::docLine(lineNo);
    return retVal;
  }
};

Motivation: Explaining a Design

Modern OO systems comprise lots of collaborations

To understand such systems, requires visualizing:
  – the inter-connection of these objects (i.e., structure)
  – the dynamic interactions among these objects (i.e., behavior)

Problem: How can we visualize these phenomena in a useful way?
Sequence Diagrams

Illustrate one instance of one collaboration among multiple objects

- Of course, each object might play a role in many collaborations

Notation:

- Use of spatial position to reflect time dimension
- Use of vertical bars to denote object “activity”
- Graphic denotation of returns from operations

---

Example: Sequence Diagram

```
GUI
    resize() message or function call

vp:ViewPort
    retrieve()

fileMgr:MyDocMgr
    docSize() retrieve()
    docLine()
    docSize()

Object instances
```

---

CSE370: Software Engineering

W. McUmber
Exercise

Draw a sequence diagram to illustrate collaboration between `printButton` and `fileMgr`