Frameworks

Libraries and Frameworks

- Old days: Subroutine libraries are collections of useful subroutines.
  - Callbacks are possible for example:

```
#include “integration.h”
// “integration.h” declares:
//    double integrate( double, double, double*(double) )

// A local function
double myFunc(double x) { sin( exp( x ) ) ; }
…
    // A call to a library routine
    double area = integrate( 0.0, b, &myFunc ) ;
    // This invocation of the library routine calls back to myFunc.
```

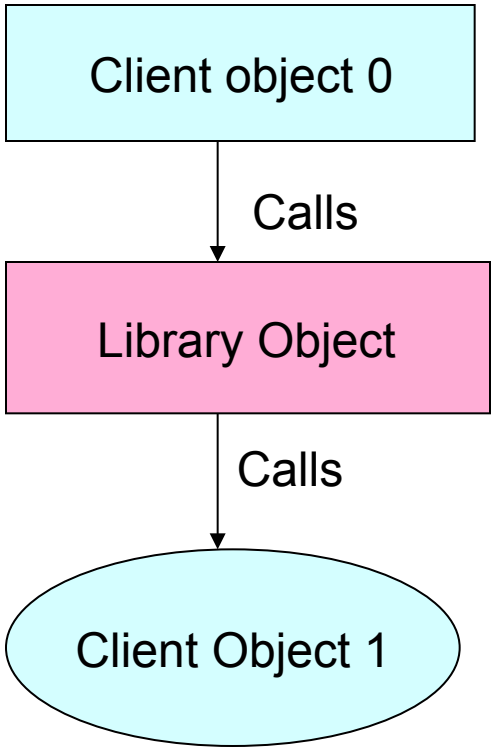
Libraries and Frameworks (cont.)

- Object oriented days:
  - Simple libraries are collections of useful concrete classes.
    - Library objects are at low-level only.
  - Frameworks are collections of concrete classes, interfaces and abstract classes such that
    - The library classes routinely call back to client level code
    - Abstract classes, concrete classes and interfaces intended to be specialized or realized by client code are called “Extension points”
    - Client *objects* may be used by library objects

Libraries and Frameworks (cont.)

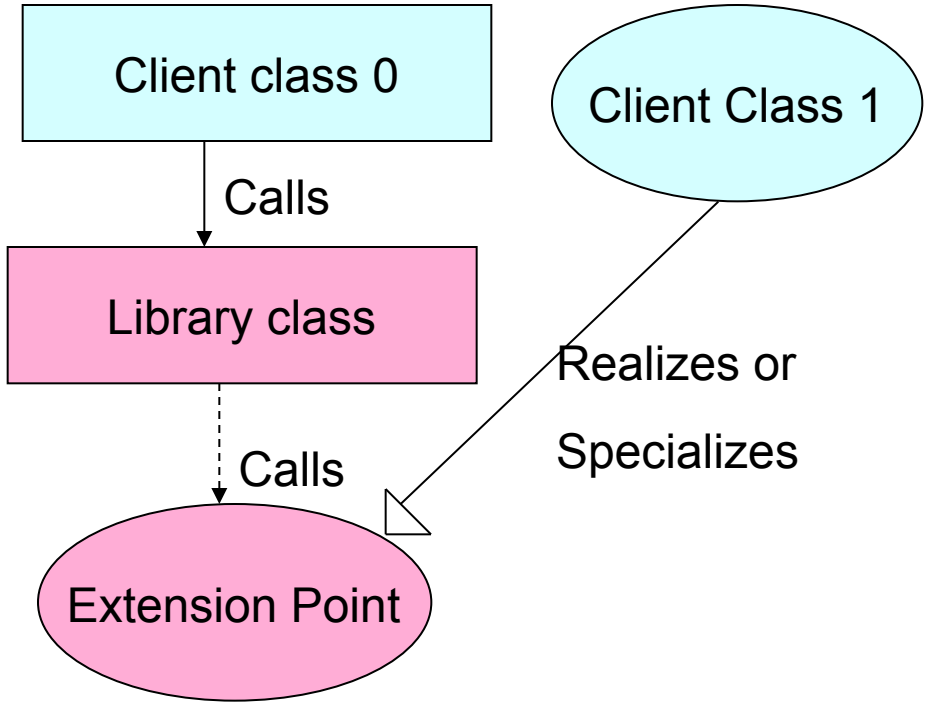
Dynamic (run-time) view

Library code calls client code



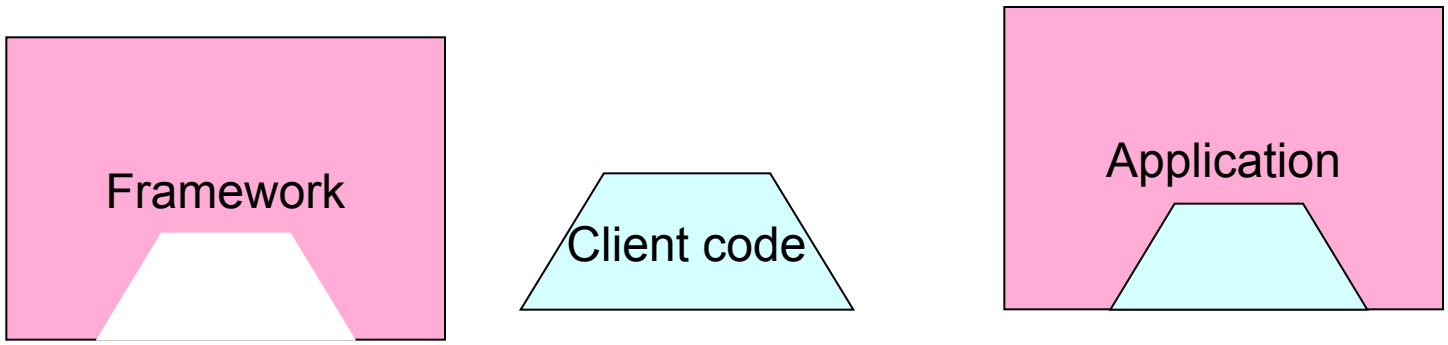
Static (compile-time) view

Library code does not depend on client code



Application Frameworks

- Application Frameworks
  - are incomplete applications
  - client code completes the application.



Example Document/Editor Framework

- Many applications are basically editors.
  - Functionality of the File Menu is essentially the same.

Extension Point: Document

```
abstract class Document {
    private boolean modified = false ;
    abstract public void writeTo( Writer w )
        throws IOException ;
    abstract public void readFrom( Reader r )
        throws IOException ;
    public void save() { …writeTo( w ); modified = false ;}
    public void open() { …readFrom( r ); modified = false ; }
    public void setModified() { modified = true ; }
    public boolean getModified() { modified = false ; }
    public void setFile( File f ) { file = f ; }
}
```

Extension Point: EditorComponent

```
abstract class EditorComponent
    extends Component
{
   abstract public void setDocument( Document e);
   abstract public Document getDocument() ;
   abstract public void AddMenus( MenuBar b ) ;
}
```

Extension point: AbstractFactory

```
interface AbstractFactory {
    Document makeDocument() ;
    EditorComponent makeEditorComponent() ;
}
```

The Document Editor Framework

- The Framework code
  - Is parameterized by an AbstractFactory
  - Handles “New”, “Open”, “Close”, “Save”, “Save As”, “Print”, and “Exit” actions.
  - Also actions on Frames such as resize, minimize, move, close, etc (mostly inherited from Frame).

Parameterizing in Java

```
// Library class
class DocEditorFrameWork {
    // Constructor
    DocEditorFrameWork( AbstractFactory f ) { … }
    … }
```

```
// Executive layer, puts the parts together.
class Main {
    static public void main( … ) {
        … new DocEditorFrameWork(
                        new Factory() ) … ) ; }
```

Parameterizing in C++

- You could use the Abstract Factory pattern
- or you could use templates.

```
int main() {
    DocEditorFrameWork
        <MyEditorClass<MyDocumentClass> >
            frameWorkObject ;
…
}
```