CSE 143

Beyond Basic C++

Templates
Modern Applications Development
Java
143 Wrapup

What's Left To Do?

- Beyond the C++ covered in this course
  - Many topics, many more details of topics we did cover
  - Main omission: templates: a C++ power feature
- Trends in programming
- Applications development frameworks
  - MFC (C++)
- Java
- A look back at the topics in 143

A Problem with Reusing Code

- Inheritance gives us a way to extend and reuse code
- Sometimes inheritance isn't the solution
- Example: Bank simulation. I have implemented a queue of customers; I also need a queue of stock transactions.
  - No "is-a" or "has-a" relationships between the items
  - Must reimplement the queue from scratch
- Would really like to have one Queue class, which could somehow be reused with different item types

Templates

- A ***template*** is a general pattern for a class or a function in C++
- Everything is filled in, except one or more types
- Examples:
  - a queue template class, with all the definitions complete, methods implemented, etc, but the type of the data item left open as a parameter
  - a sort template function: type of the item being sorted is left open
- An extremely powerful feature of C++
  - Found in only a few advanced programming languages.

A Template Class

```
template <class T> class Queue {
public:
    void insert(T item);
    T remove();
};

// in queue.cpp
template <class T>
void Queue<T>::insert(T item)
    { ... }
template <class T>
T Queue<T>::remove()
    { ... }
```


Using Templates

```
Queue<int> intQueue;
Queue<double> dblQueue;
intQueue.insert(5); intQueue.insert(7);
dblQueue.insert(3.9); dblQueue.insert(-5.3);
double dv = dblQueue.remove();
int iv = intQueue.remove();

Queue<Book *> books;
books.insert(new Book("Moby Dick"));
books.insert(new Book("Java for Losers"));
Book *eveningReading = books.remove();
```

Standard C++ Library

- The new Standard Library of C++ contains templates for many useful container types and generic algorithms
  - Originally called the Standard Template Library (STL)
- Includes
  - container class templates: list, set, map, stack, queue, vector, etc.
  - generic algorithms for searching, sorting, merging, etc.
  - iterators to link containers and algorithms
- To use these, you need to understand
  - 1. C++ templates and container classes
  - 2. The data structures and algorithms themselves (abstractly)
  - 3. Exact usage details (method names, parameters, etc.)

Trends in Programming

Old School	*New Wave*
Input/process/output	Event-driven
Reuse via libraries of functions	Reuse via libraries of classes, components, and design patterns
Programmer calls functions	Programmer inherits from classes, links components together
COBOL, C/C++, Ada, Pascal, etc.	C++, Java, Visual Basic, scripting languages, etc.
Data stored in files and databases	All that, plus data from OO databases, persistent object stores, networks, Web

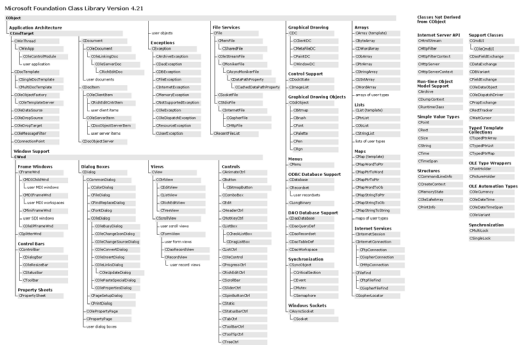
Beyond Objects: Components

- Component: a "sealed" object
  - Some methods and data are "exposed" to the outside world
- Language-neutral
  - source code not visible
  - may be used within any compliant programming language or environment, possibly even at a distance.
- Supporting and related technologies
  - Microsoft: VB, COM, OLE, Active-X, ASP, etc.
  - Sun: JavaBeans
  - CORBA
  - Scripting languages (VBScript, JavaScript, etc.)

Windows C++ Application Development

- Much C++ Windows development uses Microsoft Foundation Classes (MFC)
- MFC Key features
  - Graphical User Interface (GUI)
    `Windows, menus, buttons, drawing areas`
  - Event-driven
    `Respond to internal and external events`
    `Multi-threaded`
  - Object-oriented
    `Built-in class hierarchies for standard reusable objects`
  - Programmer's job
    `Understand the hierarchy; use and extend given classes; hook into events; add custom logic`

MFC Class Hierarchy



Key MFC Classes

- Everything descends from **CObject**
- CObject/CCmdTarget/CWinThread/**CWinApp**
  - One per application, container for the whole thing
- CObject/CCmdTarget/**CWnd**
  - A window (rectangular area); about 50 subclasses
  - FrameWnd: resizable main frame
  - CControlBar, CDialog, CButton, CEdit, etc. for user interaction
- CObject/**Exception**
- CObject/**CFile**
- CObject/**CDC**: "graphics context"

Using MFC

- Hard to learn
  - "Wizards" help somewhat
  - Nevertheless, a big improvement over previous environment:
    `Win16/Win32 API: Hundreds of individual C functions`
- Reasonably widely used
- Not perfectly integrated with Windows OS
  - mismatch with event handling
- Not part of C++
  - Mismatch or conflicts with standard libraries
  - Compiler can't check everything
  - Windows only - not available on other platforms

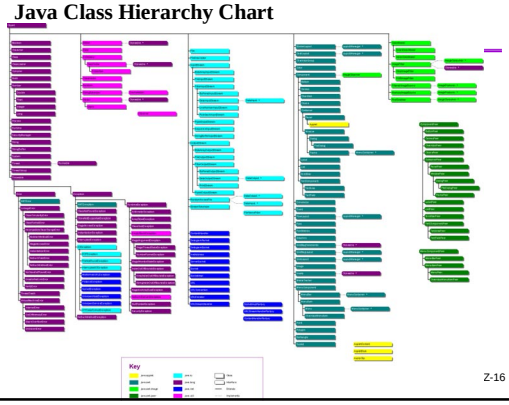
Java

- A new language created by Sun Microsystems
- Based on C++, but simpler
  - similar syntax
  - no explicit pointers
    `'new' but no 'delete': garbage collected`
  - safety checking (array bounds, etc.)
  - no preprocessor
- Designed from the ground up to be
  - object-oriented
    `no stand-alone functions`
  - GUI (AWT: Abstract Windows Toolkit, SWING)
  - platform-independent
  - Internet-friendly

Java's Object Model

- Similar to C++ in notation and overall concept
  - But some fundamental differences
- All classes descend from "Object"
- All methods automatically virtual
- Deliberately missing some C++ power features
  - No multiple inheritance
    `But provides "interfaces," which are similar to abstract base classes with no data.`
  - No operator overloading
  - No templates (but some form of templates might be added to the language in the next year or two)

Java Class Hierarchy Chart



Sample Java Code

```
import javax.swing.*;     // access libraries
import java.awt.*;        // (similar idea to include
import java.awt.event.*;  //  but not the same)

// in Java, everything goes in a class
public class AnApplication {

    // main is "static"; one instance per class
    public static void main(String[] args) {
        // create a window frame
        JFrame frame = new JFrame("MyApplication");
        // create a label to go in it
        JLabel label = new JLabel("Hello, World!");
        frame.getContentPane().add(label);
```


Sample Code (cont)

```
        // create an object to handle events
        frame.addWindowListener(new WindowAdapter() {

            // called when the window is closed
            public void windowClosing(WindowEvent e){
                System.exit(0);  // exit the program!
            }
        });

        frame.pack();            // organize the frame
        frame.setVisible(true); // show the frame
    }
}
```

Java Memory Model

- All objects and arrays allocated on the heap – even if only used locally in a function

```
Thing thing = new Thing();
```

- No "pointers" – use of references is implicit

```
thing.method(argument);
```

- Automatic garbage collection
  - Storage allocated to an object is reclaimed when the object is no longer accessible
  - No explicit "delete"
  - Rarely need destructors

Wrapping Up 143

- What did we learn?

["Professor, why is this slide blank?"]

Knowledge And Skills

- C++ Programming Specifics
  - Classes
  - Dynamic memory
  - Stream I/O, Overloading, other C++ specifics
- General programming
  - Recursion
  - Object-oriented programming style

-oriented. A clumsy, pretentious device, much in vogue. Find a better way of indicating orientation or alignment or direction.

W. Strunk & E. B. White, *The Elements of Style*

Knowledge and Skills (cont.)

- Software Engineering
  - interpreting specs
  - building sizable systems
  - documenting (charts, descriptions, comments)
  - robustness
  - testing
  - techniques for code reuse

Knowledge and Skills (cont.)

- Data structures and algorithms
  - Analysis of complexity

```
Big-O notation
```

  - Classic ADTs: List, Queue, Stack
  - Sorting and Searching, incl. Binary Search, quadratic sorts, QuickSort, MergeSort
  - Tree concepts
  - Binary Trees and traversals
  - Binary Search Trees
  - Tables and hashing

What's Beyond 143?

- CS or CE Major
  - CS: more software emphasis
  - CE: more hardware emphasis
- Other major + CSE courses
  - Long-term, a real winner
  - Combine interest/aptitude in any field with CS knowledge
- "Real-world" programming
  - Often involves maintenance of existing programs
  - Requires knowledge of customer application
  - Work as part of a team
  - May use some specialized programming tools

Courses

- After 143, it's assumed you can program!
- Non-majors courses
  - 373 (Data Structures) Most direct successor to 143
  - Then 410 (Computer Systems), 413 (Prog. Languages), 415 (Artificial Intelligence)
- Majors courses
  - Need permission if not CSE major
  - 321 (Discrete Structures)
  - 322 (Formal Models)
  - 326 (Data Structures)
  - 370 (Digital Logic)