CS2704: Object Oriented Software Design

Topic 1: Introduction

Dr. Ben Keller
Spring 2000
Outline

• Design Hierarchy
• Design philosophies
  – Procedural
  – Modular
  – Object-oriented
• Object-oriented design strategies
• Course overview
Design Hierarchy

- Describe system in terms of components, and components in terms of subcomponents
- Requires *abstraction* - hiding of details of components
- *Top-down*: decompose system into components
- *Bottom-up*: build system from small components
Procedural Programming

- Problem is divided into sequence of sub-problems to be solved
- Program is sequence of procedure calls
- Think in terms of tasks and subtasks
- Languages: C, Pascal, Fortran, COBOL, etc
Procedural Design

• Key: identify simple tasks that can be programmed easily

• Design notations:
  – structure charts - which procedures call which
  – dataflow diagrams - how data moves from one task (“process”) to another (business apps)
Problems in Procedural Programs

• Large program made up of many small procedures
• Not clear which does what to what data
• No enforced control over access to data
• Difficult to fix bugs, modify, and use procedures in other programs
Software Engineering Goals

• *Reusability* – components can be used in many applications
• *Extensibility* – ease of change
• *Flexibility* – modifications do not “break” system
Modular Programming

• Data and procedures collected (and hidden) in *module*
• Can make so that only procedures in module can modify data
• Design: think about necessary data types, and wrap modules around data
• Languages: Ada 83, Modula, SML,…
Problems with Modules

- Modules solve most problems of procedural programs
- Allows information hiding
- Only have encapsulation if data of type is stored in module
- Want module to be data
- Difficult if want more than one copy
Object-Oriented Programming

• Think of building program from parts
  Like building a machine!
• Parts are *objects* that interact to solve problem
• Define *classes* of objects that can be reused
• Think in terms of objects and interactions
• Languages: C++, Java, Eiffel, Smalltalk, etc
Why Object-Oriented?

• Procedures organized around objects
  – Data access easier to understand
  – Data access easier to control
• “Easier” to avoid design problems
(Shhh, Can you keep a secret?)

- Object interactions are defined by methods, which are just procedures
- Programs still sequences of “procedure” calls, but think of as interacting objects
Generalization in OOP

- Two approaches to defining classes in terms of others
  - *Inheritance* – “inherit” properties of other classes
  - Parameterized classes (*templates*) – class defined in terms of parameter classes
- *Design patterns* are solutions to common design problems
Object-Oriented Design

• Identify objects and classes
  *strategies*: abstraction and separation

• Identify how objects interact in system
  *strategy*: composition

• Identify hierarchies of related classes
  *strategy*: generalization
Object-Oriented Design Strategies

- Abstraction – modeling essential properties
- Separation – treat what and how independently
- Composition – building complex structures from simpler ones
- Generalization – identifying common elements
Connections

- Design Strategies:
  - abstraction
  - separation
  - composition
  - generalization

- Object-Oriented Software Structures:
  - objects
  - classes
  - inheritance
  - templates
  - design patterns

- Software Engineering Goals:
  - reusability
  - extensibility
  - flexibility
Course Overview

- C++ classes
- Design notations
- Composition
- Design
- Generalization
- Design