CS 370
REVIEW: Design Heuristics

DR. MICHAEL J. REALE
FALL 2015
Major Design Heuristics

- Find real-world objects
  - Data and functions

- Form consistent abstractions
  - Abstraction = “You can look at an object at a high level of detail.”
  - Base classes, class interfaces, function interfaces

- Encapsulate implementation details
  - Encapsulation = “FURTHERMORE, you aren’t allowed to look at an object any other level of detail”
  - Hiding lower levels
Major Design Heuristics

- **Inherit when simplifies design**
  - However, avoid:
    - Base classes with only one subclass
    - Deep inheritance trees

- **Hide secrets (information hiding)**
  - “Black box” design
  - Things to hide:
    - Inner workings (as much as possible)
    - Data formats
    - Areas likely to change
    - Area with errors that needs to be “walled off”

- **Identify areas likely to change**
  - Design interface so change will not affect interface
Major Design Heuristics

• Keep coupling loose
  ○ coupling = how tightly class/routine is related to other classes/routines
    § Size = # of connections
    § Visibility = how obvious connection is
      ○ Simple-data-parameter → normal
      ○ Simple-object (one object instantiates another) → fine
      ○ Object-parameter → tighter
      ○ Semantic → most insidious; uses knowledge of classes/routines
        inner workings
    § Flexibility = how easily you can change connections
Major Design Heuristics

- Look for common design patterns
  - **Design patterns** = time-tested patterns that can be applied to specific situations
    - **Advantages:**
      - Easier to discuss and communicate design
      - Reduce errors → standardized way to solve certain problems
      - Gives you suggestions for design alternatives
    - **Warnings:**
      - Don’t force-fit a pattern
      - Make sure pattern fits problem
Barriers of Information Hiding

• Things that break Information Hiding
  ○ Excessive distribution of information
    ▪ E.g., hardcoded 100 instead of MAX_EMPLOYEES
  ○ Circular dependencies
  ○ Global data (and to a lesser extent class data)
Popular Design Patterns

- Design patterns fall into one of three categories:
  - Creational
    - Way to create objects while hiding creation logic
  - Structural
    - Class and object composition
  - Behavior
    - Communication between objects
Popular Design Patterns

- **Creational**
  - Factory Method
    - Makes classes derived from a base class
  - Abstract Factory
    - Effectively a factory of factories
  - Singleton
    - Only one instance; accessed without explicit instantiation
Popular Design Patterns

- **Structural**
  - **Adapter**
    - Converts class interface \(\rightarrow\) different class interface
  - **Bridge**
    - Interface and implementation \(\rightarrow\) can vary each without varying the other
  - **Composite**
    - Group of objects treated like single object
    - Creates tree structure
  - **Decorator**
    - Add new functionality to an existing class without altering its structure
      - Wrapper around existing classes
      - Keeps class method signatures intact
  - **Facade**
    - Provides consistent interface to code that wouldn’t otherwise offer a consistent interface
      - Single class with simplified methods that delegates calls to existing classes
Popular Design Patterns

- **Behavior**
  - **Iterator**
    - Access elements sequentially without knowing underlying representation
  - **Observer**
    - Keeps multiple objects in sync with one another
    - Object (observer) notifies set of objects of changes
  - **Strategy**
    - Defines a set of algorithms/behaviors that are dynamically interchangeable with each other
  - **Template**
    - Defines structure of an algorithm but leaves some of the detailed implementation to subclasses
Other Useful Design Heuristics

- **Aim for strong cohesion**
  - Classes/routines should have a well-defined central purpose

- **Build hierarchies**
  - Class hierarchies, routine hierarchies, etc.

- **Formalize class contracts**
  - Pre-conditions and post-conditions
  - Asserts

- **Assign responsibilities**
  - What is each object responsible for?
  - Similar but broader question than, “What does each object hide?”
Other Useful Design Heuristics

- **Design for test**
  - In particular, design interfaces thinking about how you would test classes/subsystems/routines

- **Avoid failure**
  - Think about how things might fail (and how to prevent it)

- **Choose binding time consciously**
  - Binding time = time a specific value is bound to a variable
  - Earlier → simpler, but less flexible
Other Useful Design Heuristics

- Make central points of control
  - “The Principle of One Right Place”
  - Even a named constant (e.g., MAX_EMPLOYEES) centralizes control

- Consider using brute force
  - A brute force solution that works better than elegant solution that doesn’t

- Draw a diagram
  - Doesn’t have to formal → helps you focus your ideas and thoughts

- Keep your design modular
  - Try to design things with the “black box” idea in mind