Classes and Responsibilities

Responsibility / Providing a service

Deciding on class *Responsibility*

- Each class can be seen as providing a *service* to it clients in the system.
- This service is its “responsibility”
- Other classes (client code) access the class’s service via its public interface.

Consider some human examples

- A baker has the responsibility of baking a cake
- An appliance repairer has the responsibility to fix broken ovens

Information hiding / Keeping a secret

- By parcelling out responsibilities we reduce the information burden on people.
  - Bakers don't need to know how to fix ovens
  - Repairers don't need to know how to make cakes
- Thus each responsibility is associated with a "secret" which is a set of information that the rest of us don't need to know to get our jobs done.

Information hiding / Keeping a secret

- Behind its service, each class hides some information that its clients do not need to get their job done.
- The information hidden might reflect either
  - An internal design decision or
  - An aspect of the system's specification

Information hiding / Keeping a secret

- Specifications impose a number of *constraints*.
- Design of a system involves a number of smaller *design decisions*.
- To keep the design simple we try to limit the impact of each constraint or design decision to 1 class (or a small number of classes)
- Each class *hides* the *information* about a constraint or design decision.

Good design

Good design will

- Keep the responsibilities of each class few, simple, and simple to access
- Break complex responsibilities down hierarchically so that no one class is too complicated
- Hide the effect of each design decision and requirement in 1 class --or as few classes as possible
- With priority on design decisions and requirements that are likely to change or differ when the component is reused

Example: A Chess Playing Robot

Needs classes or packages to do the following:

- Executive: Manage the flow of the game.
- Input: Sense moves made by the human
- Output: Move the pieces on the physical board.
- Board: Internal representation of board states.
- Rules: Knows for each board state what the legal moves are.
- Strategy: Simulates and evaluates different courses of action.

Kinds of classes

- Information storage
  - Service: An abstract interface to the information
  - Secret: The data structures used.
  - Examples: Classes representing sequences, sets, finite functions, chess boards, trees, graphs, diagrams, UML models.
- Algorithmic
  - Service: Performing some manipulation of data.
  - Secret: The algorithm used.
  - Examples: Sort a sequence, find shortest path in a graph, determine best move in a chess game, plot best course through a room.

Kinds of Classes (continued)

- Device interface
  - Service: Provide access to an external device
  - Secret: The nature of the device and the protocols for using it.
  - Examples: Sensing the input from a touch sensitive chess board, moving a robotic arm to a specific location, outputting to a console (or console window)
- OS interface:
  - Service: Services provided via the OS.
  - Secret: The OS being used and the means to communicate with it.
  - Examples: Interacting with the file system or window system or process system.

Kinds of classes (cont. again)

- Formatting and Parsing Classes
  - Service: Input and output of data in specific formats
  - Secret: The format of files
  - Examples: Input and output filters in word-processors. Configuration file reading and writing.
- Adaptation Classes
  - Service: Representing other classes in a way conforming to a given interface.
  - Secret: Which classes are being represented.
  - Examples: Event listener classes (e.g. ActionListeners in Java). Iterator classes (e.g. Enumerations in Java).

Kinds of classes (cont. yet again)

- Structural classes
  - Service: Providing a single interface to a number of objects,
  - Secret: The details of those objects
  - Example: Façade classes (classes). Container classes in Java's AWT.
- Creators
  - Service: Creating objects needed by other classes.
  - Secret: The method of creation and the concrete class's of the objects.
  - Example: Factory classes (later)

Kinds of classes (one last one)

- Arbitrary facts
  - Examples: Numerical constants, rules for financial calculations, rules of games, rules for formatting dates and monetary amounts, strings corresponding to messages, names of provinces and countries, rules for checking validity (e.g. of postal codes, credit card number), etc.
  - These “facts’’ tend to change as software is used in other countries and languages (internationalization).

Evaluating a design

- We can consider the impact of each kind of likely change on the design.
  - No recoding: Change is confined to constants or configuration files.
  - Class level: One class needs to change.
  - Package (or cluster level): Changes are confined to a small number of closely related classes.
  - Global: Public interfaces of packages must change. Changes span multiple packages.

Evaluating the Chess Example

- Changes to input or output method
  - E.g. Video / mouse rather than robotic
  - Confined to Input and Output classes
- Changes to strategy
  - Confined to Strategy classes
- Changes to rules
  - Unlikely, but would affect mainly the Rules classes
- Changes to representation
  - Affect only the Board classes

Families of products

- By interchanging a set of classes with other classes that implement the same interface, we obtain a family of products.
- Examples:
  - Port to new OS: Change only OS interface classes.
    - Porting the Java AWT requires implementation of certain "peer" classes representing buttons, windows, etc.
  - Port to new device: Change only device interface class.
  - Teaching Machine, Interpret new language: Change only the implementation of the "Language" interface.

Documenting Classes

- Short description
- Service(s) provided
- Secret(s)
- Interface
  - In Java, simply document each public method separately.
- Collaborators: What classes does this class depend on and why. (In UML use a class diagram.)

Documenting Classes (JavaDoc)

```
/** Represent the state of a chess board.
<p> Stores the location of the pieces on a
   board and other information on the state of
   the game.
@author Theodore Norvell */
public class ChessBoard {
   /** Construct a board with all pieces in initial
      positions. */
   public ChessBoard() { … }
```

Keeping classes isolated

A class is isolated if it depends on few other classes.

By isolating components (packages and classes)

- Changes can be restricted to fewer classes
- Components are more reusable
- Components can be tested in isolation

More to read

- David Parnas: On The Criteria To Be Used In Decomposing Systems Into Modules, Comm. ACM, Dec., 1972.
  - https://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf