Section 2.2
Object Design Categories

1. MVC and not-MVC
2. Major object categories
3. Collection classes

2.2.1 MVC and Not-MVC

- What is MVC?
  - Model-View-Controller
  - it's a technique for organizing classes within a system
    - in software engineering, a *system* is a large program
- Controversy
  - some say MVC is a *design pattern*
  - others say it's an *architectural pattern*
  - maybe it's both

MVC and Not-MVC (cont.)

- MVC as a design pattern
  - a design pattern is a solution to a common design problem
  - MVC as a design pattern is very similar to the Observer pattern
  - you can’t have MVC without Observer

- MVC as an architectural pattern
  - an architectural pattern organizes classes at a very high-level
    - think: client-server, peer-to-peer, etc.
  - still offers a specific design for interfaces between 3 components
    - model, view, controller

MVC and Not-MVC (cont.)

- What are *object design categories* **not**?
  - they are *structured* like MVC
    - same kinds of objects
  - they do not *behave* like MVC
    - no use of the Observer design pattern
  - therefore object design categories are **not MVC**

- So what are object design categories?
  - they are an OO design technique that promotes encapsulation
  - they help us design classes that are single-purpose and reusable

2.2.2 Major Object Categories

- What are the major categories?
  - *entity* objects
  - *control* objects
  - *boundary* objects
    - can be called *view* or *UI* objects

- Entity objects
  - they represent *real-world* information tracked by the program
  - they often represent *persistent* information
    - persistent objects survive between program executions
  - examples: Book, Library, Patron classes

Major Object Categories (cont.)

- Control objects
  - they are in charge of the program control flow
  - they manage how classes interact with each other
  - in a typical OO program, **`main()`** function has two lines of code:
    - create a control object
    - call some launch function on that object
  - the control object then takes charge of program control flow

- Boundary objects
  - they are solely responsible for interactions with end users
    - ... and with other systems (but this is outside the scope of this course)
  - ideally, **no other classes** should communicate with end user

Major Object Categories (cont.)

- Why separate objects into categories?
  - it's easier to make changes
  - modifiability, extensibility are very important SE qualities

- Examples:
  - entity objects can be reused between programs
    - if we model Book correctly, that class should be reusable
  - replacing a UI just means implementing new boundary objects
    - entity and control objects can stay the same

2.2.3 Collection Classes

- What is a collection?
  - a data structure that stores multiple data of the same type
  - two options:
    - primitive collection
    - collection class

- What is a primitive collection?
  - it's a type of collection built into the programming language
    - very basic, with no special features
  - it's a fancy way of saying *array*

Collection Classes (cont.)

- What is a collection class?
  - it's a class whose purpose is to store a collection
  - it's sometimes called a *container*
  - it must use an internal collection data structure
    - called the *underlying* container or collection

- Why not use arrays everywhere?
  - think: principle of least privilege
  - a primitive array doesn't restrict operations on itself
    - any part of the program can add to, delete from, or modify an array
  - it's **bad** software engineering

Collection Classes (cont.)

- Advantages of using collection classes
  - we can hide the data inside a collection object
  - we have 100% control over how the data is accessed and modified
    - we write the member functions
    - we decide if and how data is accessed, added, deleted, modified
  - it's **good** software engineering