1 Command–query separation

Methods can be classified broadly in three categories:

- **Observer/query method**: report on state of receiver object but have no (visible) side effects.
- **Creator/factor method**: creates a new object, does not have a side effect on other objects.
  - Creator methods appear in a creator interface.
  - Constructors are one kind of creator.
  - May be a static method, a constructor, or implemented using the abstract factory pattern.
  - Producer methods: create new object from existing objects of the same type (e.g., string concatenation). Producer method is method of the object.
- **Mutator/command method**: primary purposes is a side effect to the receiver object
  - Side effect is abstract: the receiver may be implemented in terms of other objects.
  - Does not return much (e.g., remove method returns a boolean).
- **Command–Query Separation pattern (or principle)**: methods should fall into one of these categories. Makes interface easier to understand.

2 User interface design

A good user interface:

- enables users to get their job done efficiently, easily, and enjoyably.

Not about:

- Making programmer’s job easy by exposing underlying functionality as directly as possible.
- Providing the largest feature set possible.
- Giving users what they say they want (your users are not good UI designers). (But listen carefully to your users anyway!)

**Principles for UI design:**

- Know the users and design to them.
  - frequent or occasional user?
  - training expected?
  - Infrequent/novice users:
    * App should be easy to learn (gentle learning curve)
    * UI consistent with similar apps
    * Simple, clear displays
    * Use of metaphor to communicate (e.g., icons)
    * Gradual feature exposure
    * No loaded guns
You are not like the user: user testing is required.

- **Frequent/power users**
  - App should be efficient and powerful
  - Information-rich displays
  - Rich controls with shortcuts for common actions
  - Easy to remember

- **Direct manipulation vs. I/O**
  - UI is an abstraction of the underlying application state (the model)
  - Abstraction does not need to match the implementation
  - Good implementation strategy: Model != View != Controller
  - Good abstraction: Model = View = Controller
  - Example: driving a car vs. programming/remote-controlling a car (Word vs. TeX)
  - Goal: app feels like extension of user

- **Interaction time scales**
  - < 1/60s: biologically imperceptible: faster than neurons
  - < 1/30: fast enough for continuous-feedback tasks (e.g., mouse tracking)
  - < 1/10s: imperceptible delay for discrete actions, e.g. button clicks.
  - < 1/2s: fast but noticeable (ok for command-response interaction)
  - 1/2s – 5s: increasingly annoying but user stays focused
  - 5s – 10s: User starts to lose attention.
  - 10s – 1min: User becomes distracted and productivity declines. App needs to support parallel activities.
  - > 1 min: Significant loss of productivity. User leaves for coffee.

- **UI is a dialogue**
  - GUI helps by restricting the vocabulary
    - Channels ‘utterances’ (user actions) into meaningful directions
    - Reduces memorization
  - Avoid excessive modes
    - Modes: states of application that restrict interactions.
  - App needs to be good conversation partner
    - Ratify actions quickly
    - Be responsive (e.g., highlighting)
    - Show progress on longer actions
  - Work out the conversations
    - Use a set of *use cases* to figure out what users will have to do.
    - Eliminate unnecessary user actions.
    - Aim for short interactions with clear progress: *intermediate goal satisfaction* (cf. DisneyWorld ride lines)
  - Obvious controls
    - Easy to find and identify
* Don’t set up user for a fall: disable invalid actions
  - Avoid overload
    * Human can only hold 7 things in their head at once
    * Avoid long menus, lots of buttons
    * Design visuals carefully for rapid comprehension