Clean Code: Writing Code for Humans

Classes

Cory House
BitNative.com
Twitter: @housecor

pluralsight
hardcore developer training

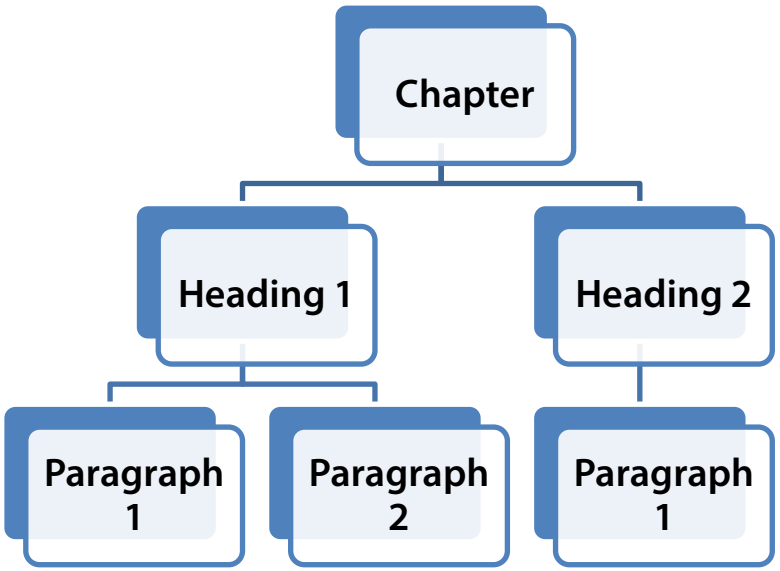
References

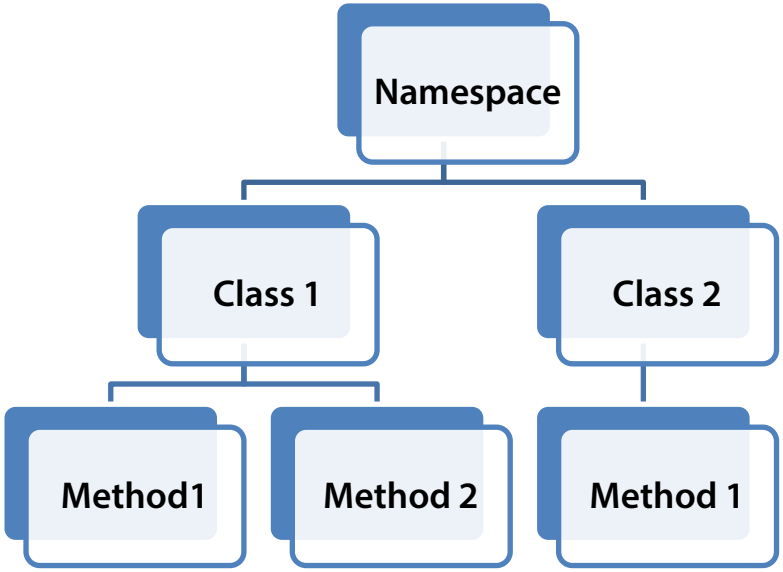
- SOLID Principles - Steve Smith
  - http://bit.ly/13sEdjV
- Design Patterns - Steve Smith, et al.
  - http://bit.ly/15qxUwK

Agenda

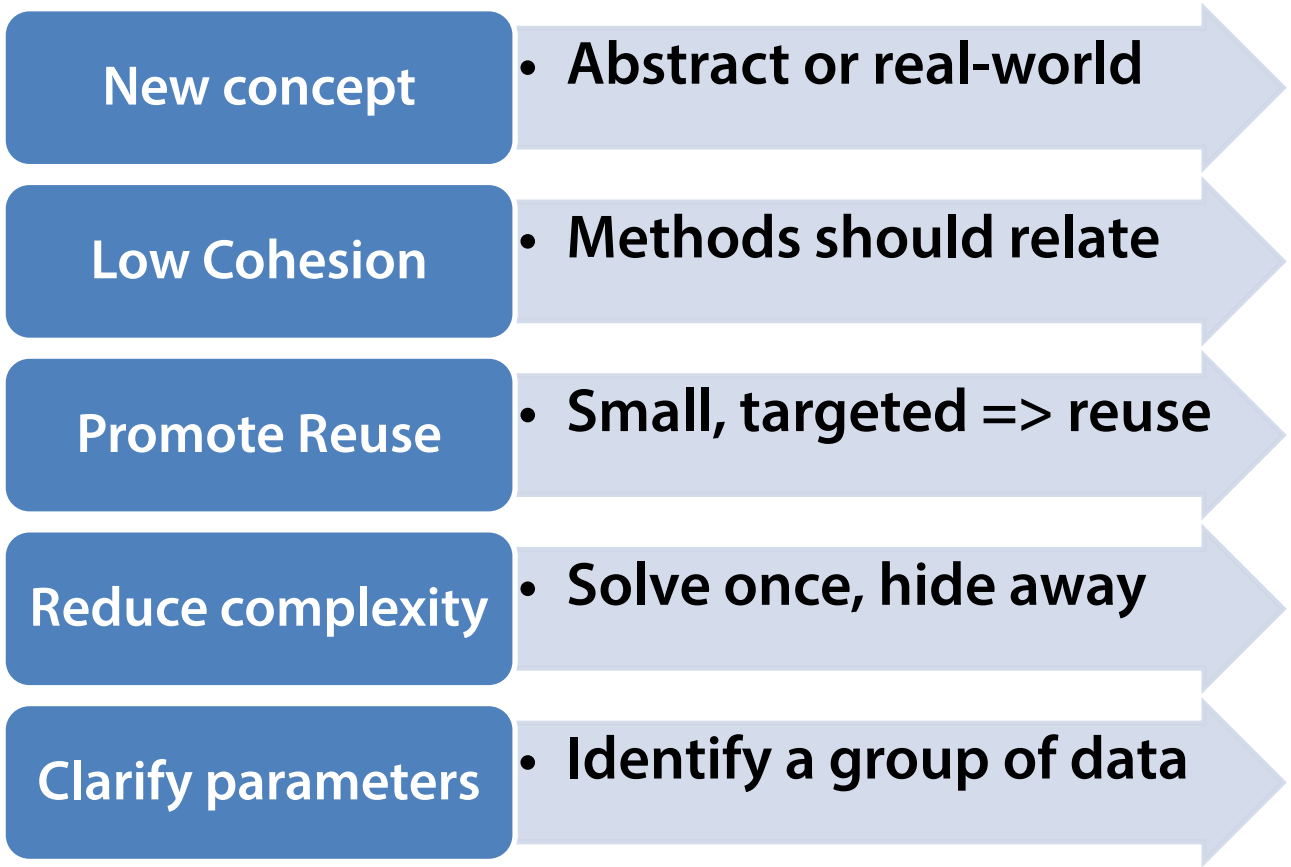
- When to create a class
- Cohesion
- Organization
- Primitive Obsession
- Outline Rule

Classes are like headings in a book





When to create a class



High Cohesion

- Class responsibilities should be strongly-related.
  - Enhances readability
  - Increases likelihood of reuse
  - Avoids attracting the lazy
- Watch out for:
  - Methods that don't interact with the rest of the class
  - Fields that are only used by one method
  - Classes that change often

High Cohesion

Low

- **Vehicle**
  - Edit vehicle options
  - Update pricing
  - Schedule maintenance
  - Send maintenance reminder
  - Select financing
  - Calculate monthly payment

High

- **Vehicle**
  - Edit vehicle options
  - Update pricing
- **VehicleMaintenance**
  - Schedule maintenance
  - Send maintenance reminder
- **VehicleFinance**
  - Select financing
  - Calculate monthly payment

Sniffing out lack of cohesion

Dirty

- WebsiteBO
- Utility
- Common
- MyFunctions
- JimmysObjects
- *Manager / *Processor/*Info

Specific names lead to smaller more cohesive classes

Deep Thoughts

Ever complain that a class is too small?

Signs it's too small:

1. Inappropriate intimacy
2. Feature envy
3. Too many pieces

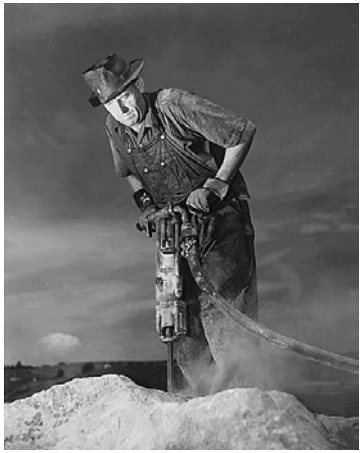
Primitive Obsession

Dirty

```
private void SaveUser(string firstName, string lastName, string state, string zip,
    string eyeColor, string phone, string fax, string maidenName)
```

Clean

```
private void SaveUser(User user)
```



1. Helps reader conceptualize
2. Implicit -> Explicit
3. Encapsulation
4. Aids maintenance
5. Easy to find references

Principle of Proximity

- Strive to make code read top to bottom when possible
- Keep related actions together

```
private void ValidateRegistration()
{
    ValidateData();

    if (!SpeakerMeetsOurRequirements())
    {
        throw new SpeakerDoesntMeetRequirementsException("This speaker doesn't meet our standards.");
    }

    ApproveSessions();
}

private void ValidateData()
{
    if (string.IsNullOrEmpty(FirstName)) throw new ArgumentNullException("First Name is required.");
    if (string.IsNullOrEmpty(LastName)) throw new ArgumentNullException("Last Name is required.");
    if (string.IsNullOrEmpty(Email)) throw new ArgumentNullException("Email is required.");
    if (Sessions.Count() == 0) throw new ArgumentException("Can't register speaker with no sessions to present.");
}

private bool SpeakerMeetsOurRequirements()
{
    return IsExceptionalOnPaper() || !ObviousRedFlags();
}
```

The Outline Rule

Collapsed code should read like an outline.
Strive for multiple layers of abstraction.

- **Chapter Title**
  - Heading 1
    - Paragraph 1
    - Paragraph 2
    - Paragraph 3
  - Heading 2
    - Paragraph 1
    - Paragraph 2
  - Heading 3
    - Paragraph 1

- **Class**
  - Method 1
    - Method 1a
      - Method 1ai
      - Method 1bii
    - Method 1b
    - Method 1c
  - Method 2
    - Method 1
    - Method 2
  - Method 3
    - Method 1

The Outline Rule

Typical Class

- **Class**
  - Method 1
    - Method 1a
      - Method 1ai
      - Method 1aii
      - Method 1aiii
    - Method 1b
    - Method 1c

Strive for this

- **Class**
  - Method 1
    - Method 1a
      - Method 1ai
      - Method 1aii
    - Method 1b
      - Method 1bi
      - Method 1bii
    - Method 1c
  - Method 2
    - Method 2a
    - Method 2b
  - Method 3
    - Method 3a
    - Method 3b

Summary

- Cohesion – Strongly related methods
- Watch for Primitive Obsession
- Organize to read top-bottom where possible
- Place related code together
- Multiple layers of abstraction: Should read like a high-level outline