Writing Clean Classes

Cory House
PRINCIPAL CONSULTANT

@housecor www.cleancsharp.net

Agenda

Classes are like book headings

When to create a class

Measuring quality

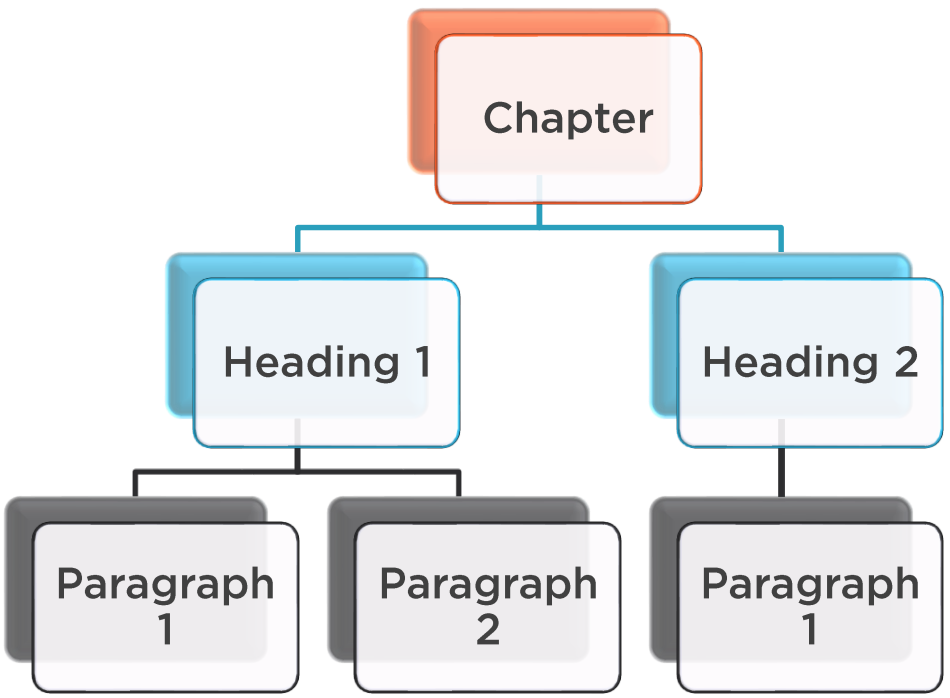
- Cohesion
- Name
- Size

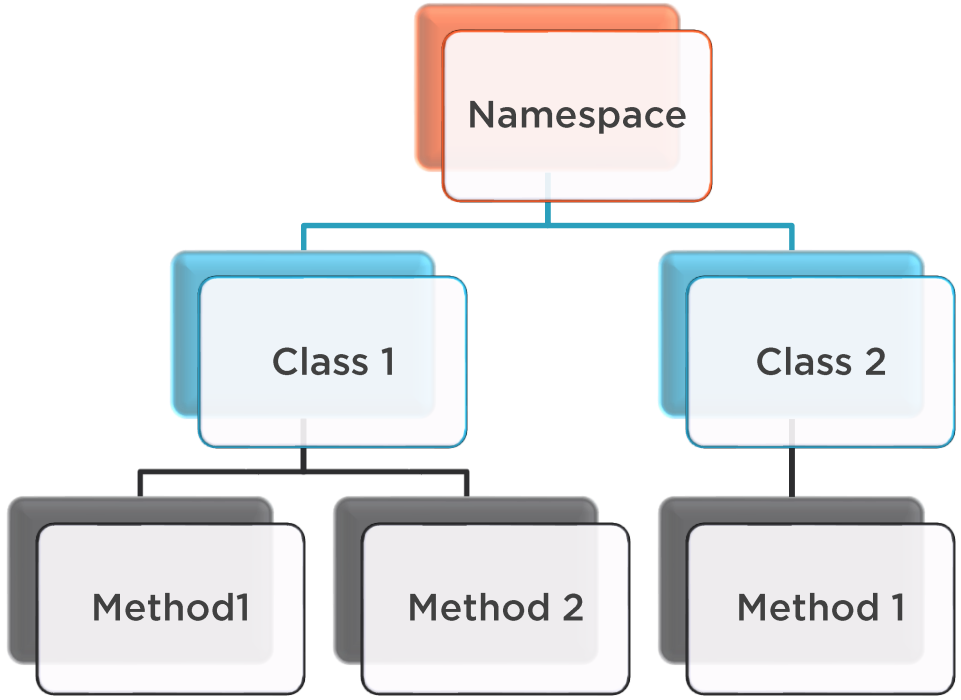
Primitive obsession

Proximity principle

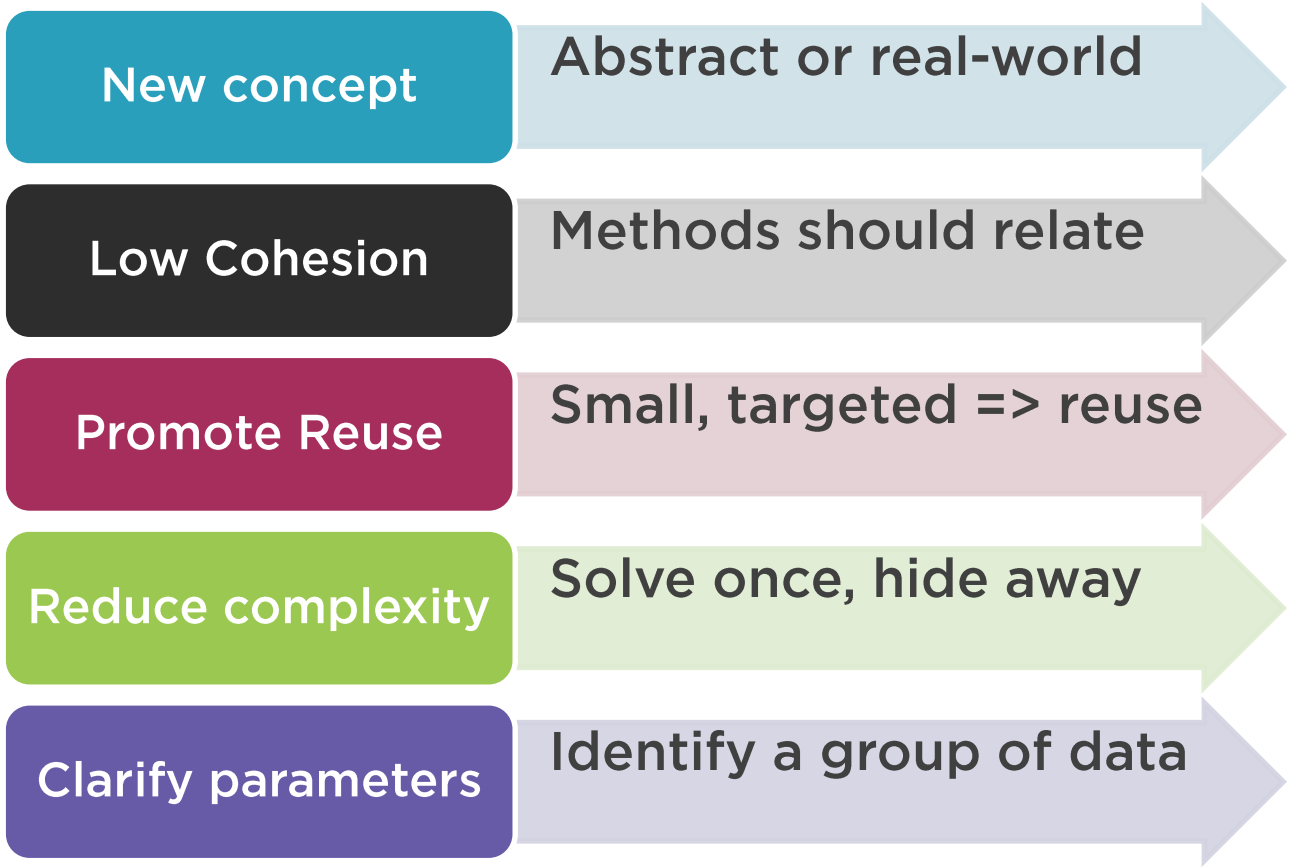
Outline Rule

Classes Are Like Book Headings

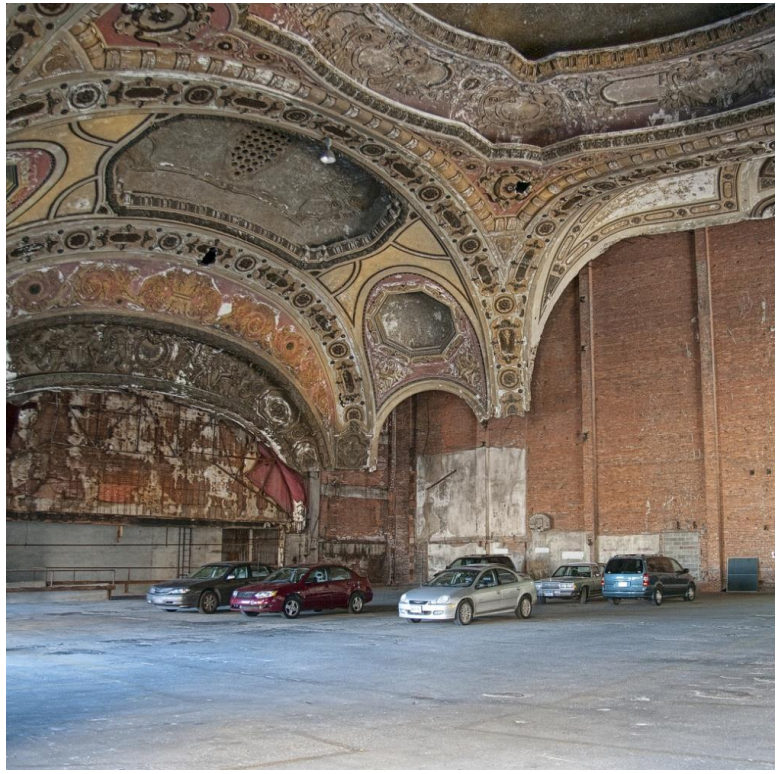




When To Create a Class



Class Responsibilities Should Be Strongly-related



Enhances readability

Increases likelihood of reuse

Avoids attracting the lazy

Watch for:

- Standalone methods
- Fields used by only one method
- Classes that change often

Low vs High Cohesion

Low Cohesion

Edit vehicle options

Update pricing

Schedule maintenance

Send maintenance reminder

Select financing

Calculate monthly payment

High Cohesion

Vehicle

- Edit vehicle options
- Update pricing

VehicleMaintenance

- Schedule maintenance
- Send maintenance reminder

VehicleFinance

- Select financing
- Calculate monthly payment

Broad Names Lead to Poor Cohesion

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? (It's rare)

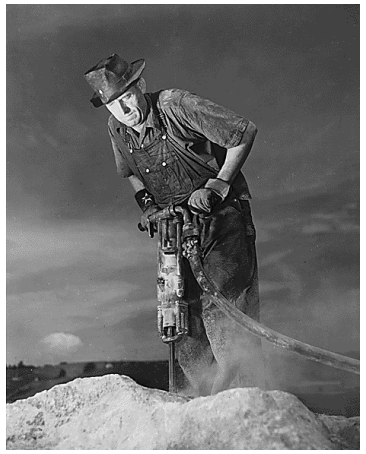
Signs it's too small

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

Primitive Obsession

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

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



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

Proximity Principle

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

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

Consider creating multiple layers of abstraction

The Outline Rule

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

- 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

- Specific names encourage cohesion
- Avoid “magnet classes”

Classes are rarely “too small”

- Inappropriate intimacy, feature envy

Watch for primitive obsession

Proximity principle

- Place related code together
- Organize to read top down if possible

Outline rule

- Multiple layers of abstraction
- Should read like a high-level outline

Next up: Comments