Code Smells between classes (II)

CODESAI

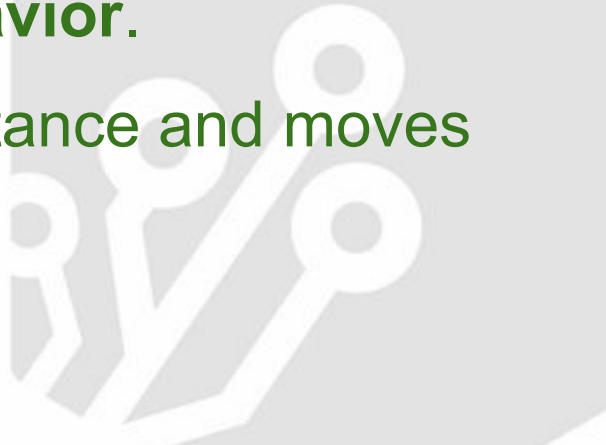
Smells between classes

Smells Between Classes

- Responsibility Smells
- Data Smells
- **Inheritance Smells**
- Accommodating Change Smells
- Library Classes smells

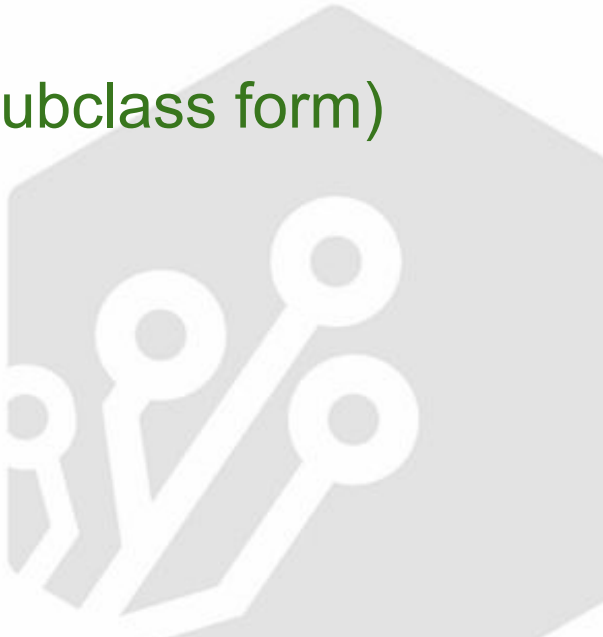
Inheritance Smells

- Inheritance produces a **stronger coupling than delegation** because **inheritance coupling is implicit**.
- Beware of **modelling the world**. Your code will be more robust if you **organize objects by behavior**.
- A class structure often starts with inheritance and moves to a more compositional style over time.



Inheritance Smells

- Refused Bequest
- Inappropriate Intimacy (subclass form)
- Lazy Class



Other inheritance Smells

- Simulated Inheritance (Case statement)
- Parallel Inheritance Hierarchies
- Combinatorial Explosion

Refused Bequest (OO Abuser)

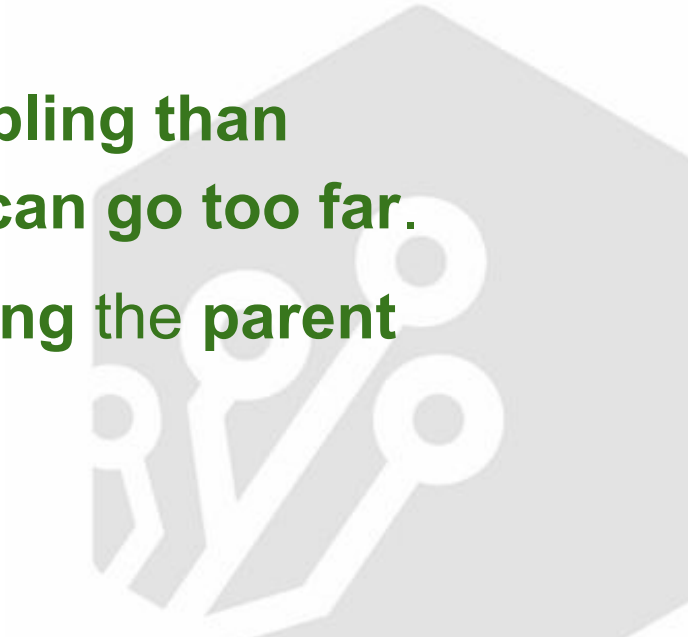
- **Honest refusal:** A class inherits a method from a parent but throws an exception instead of supporting the method. **A severe abuse of OO**.
- **Implicit refusal:** A class inherits a method from a parent but the method just doesn't work as expected. **Might be more dangerous!** (example).
- Both cases are **LSP violations**.
- Sometimes inheritance really makes no sense.

Refused Bequest (OO Abuser)

- Refactor tips:
  - ⚠️ If it's not confusing leave it.
  - The type of refactoring to apply depends on whether inheritance makes sense or not.
    - If it makes no sense, move to composition.
    - If it does make sense, rearrange the hierarchy.
- **Removal** 💰:
  - Improves communication and testability.

Inappropriate Intimacy (subclass form) (Coupler)

- A subclass accesses internal (“should-be-private”) parts of its parent.
- **Inheritance produces more coupling than delegation**, but **sometimes this can go too far**.
- We are talking about **compromising** the **parent class's encapsulation**.



Inappropriate Intimacy (subclass form) (Coupler)

- Refactoring tips:
  - Encapsulate parent's data.
  - The Template Method pattern might be useful when there is a general algorithm specialized for subclasses.
  - Sometimes more decoupling is needed, then better favour composition over inheritance.
- **Removal** 💰: Reduces duplication. Improves communication. May reduce size.

Lazy Class (Dispensable)

- A class is not doing much because all its responsibilities have been moved to other places in the course of previous refactorings.
- **Removal** 💰:
  - Improves Communication. Reduces size.

Smells Between Classes

- Responsibility Smells
- Data Smells
- Inheritance Smells
- **Accommodating Change Smells**
- Library Classes smells

Accommodating Change Smells

- Some problems become most apparent when you try to change your code.
- Ideally, one changed decision affects only one place in the code when it doesn’t is a sign of duplication in the code.
- Many times addressing these smells makes the code easier to test

Accommodating Change Smells

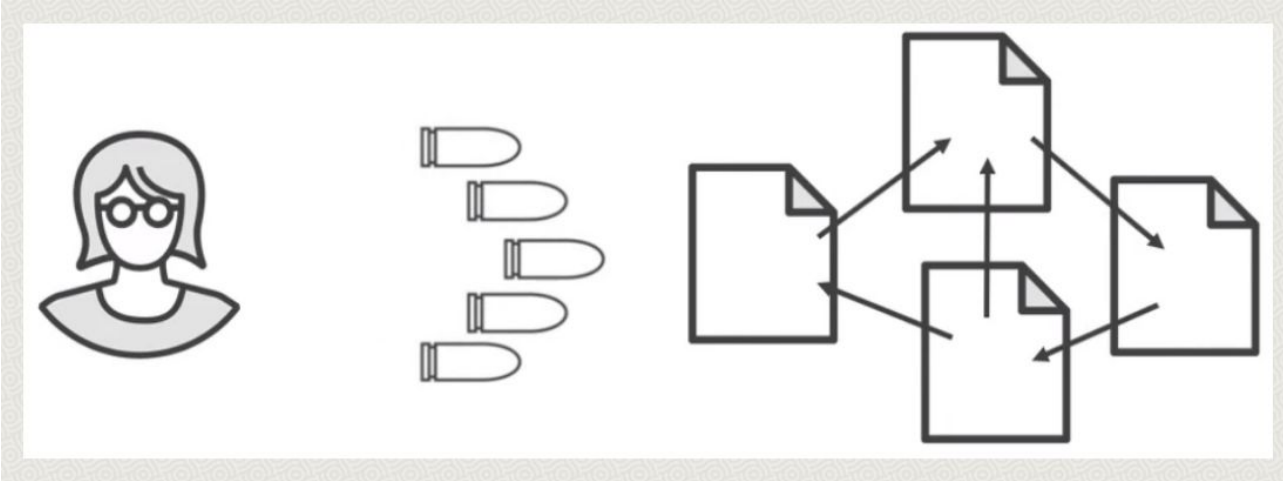
- Divergent Change
- Shotgun Surgery
- Parallel Inheritance Hierarchies
- Combinatorial Explosion

Divergent Change (Change Preventer)

- You find yourself **changing the same class for different reasons**. **SRP** violation.
- May result from overdoing **DRY** or from a **Large Class**.
- Refactoring tips:
  - Segregate the responsibilities (see **Large Class**).
- **Removal** 💰:
  - Improves communication (by expressing intent better).
  - Improves robustness to future changes.

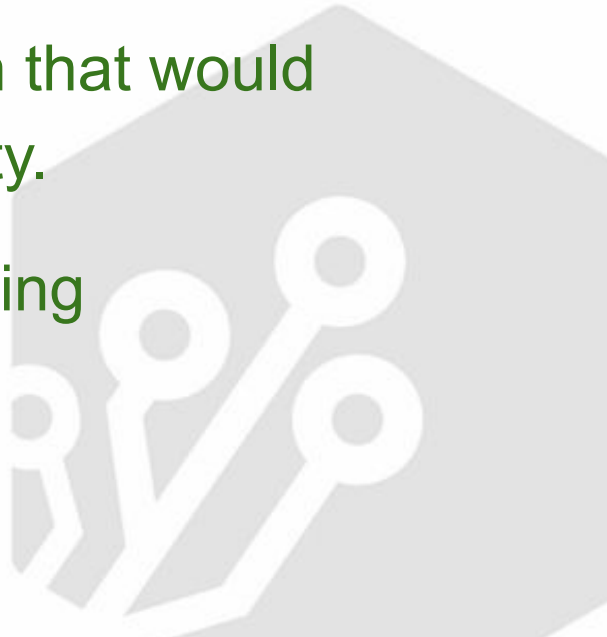
Shotgun Surgery (Change Preventer)

- Making **a simple change requires you to change several classes because the responsibility is split among them**.



Shotgun Surgery (Change Preventer)

- **SRP** violation.
- There maybe a missing abstraction that would encapsulate the whole responsibility.
- Or this can happen through overdoing **Divergent Change** elimination.

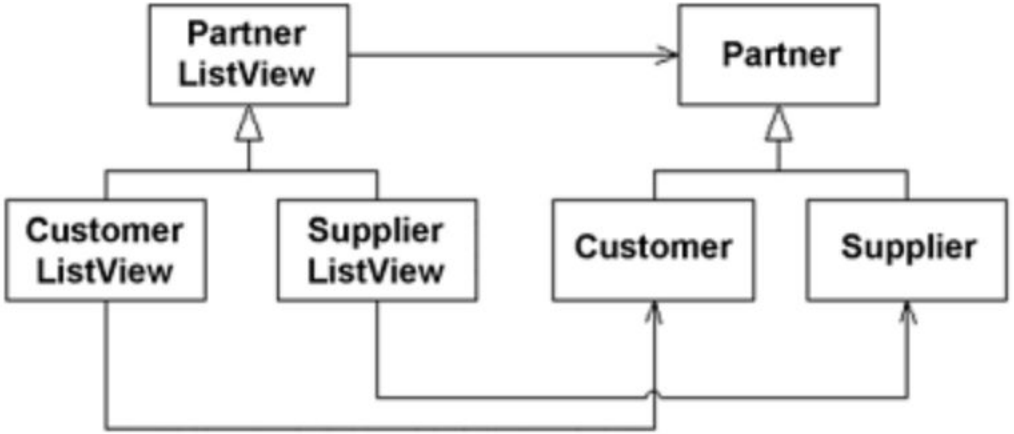


Shotgun Surgery (Change Preventer)

- Refactoring tips:
  - Identify the class that should own the responsibility. Maybe it already exists or maybe you have to extract it or create it. Then *move* the responsibility there.
- **Removal** 💰:
  - Reduce duplication.
  - Improves communication and maintainability.
- ⚠️ It might be difficult to remove...

Parallel Inheritance Hierarchy (Change Preventer)

- You make a new subclass in one hierarchy, and find yourself required to create a related subclass in another hierarchy.
- A special case of **Shotgun Surgery**.
- Sometimes the related subclasses have the same prefix in both hierarchies.



Parallel Inheritance Hierarchy (Change Preventer)

- Refactoring tips:
  - Redistribute the features in such a way that you can eliminate one of the hierarchies.
- **Removal** 💰:
  - Reduces duplication.
  - May improve communication and reduce size.

Combinatorial Explosion

- You want to introduce a single new class, but you have to introduce multiple classes at various points of the hierarchy.
- You notice that each layer of the hierarchy uses a common set of words.
- The problem is that **what should have been independent responsibilities instead got implemented via a hierarchy** (see example).

Combinatorial Explosion

- Refactoring tips:
  - If things haven’t got to far, use composition instead of inheritance for the problematic aspect (you might use the Decorator pattern).
  - If the situation has grown too complex, *remove inheritance completely*, (⚠️ big refactoring territory!!).
- **Removal** 💰:
  - Reduces duplication and size.

Smells Between Classes

- Responsibility Smells
- Data Smells
- Inheritance Smells
- Accommodating Change Smells
- **Library Classes smells**

Library Classes Smells

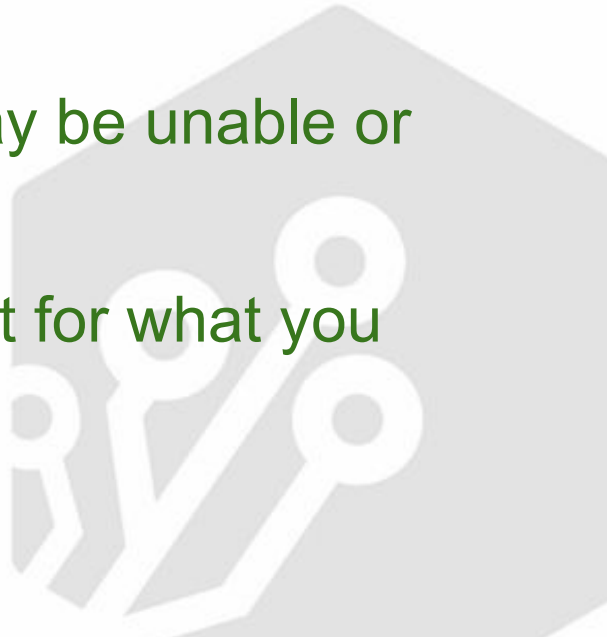
- Sometimes we want a library to be different, but we don’t want to change it.
- Even when it’s possible to change a library, it carries risks:
  - it can affect other clients
  - we have to redo our changes for future versions of the library

Library Classes Smells

- Incomplete Library Class

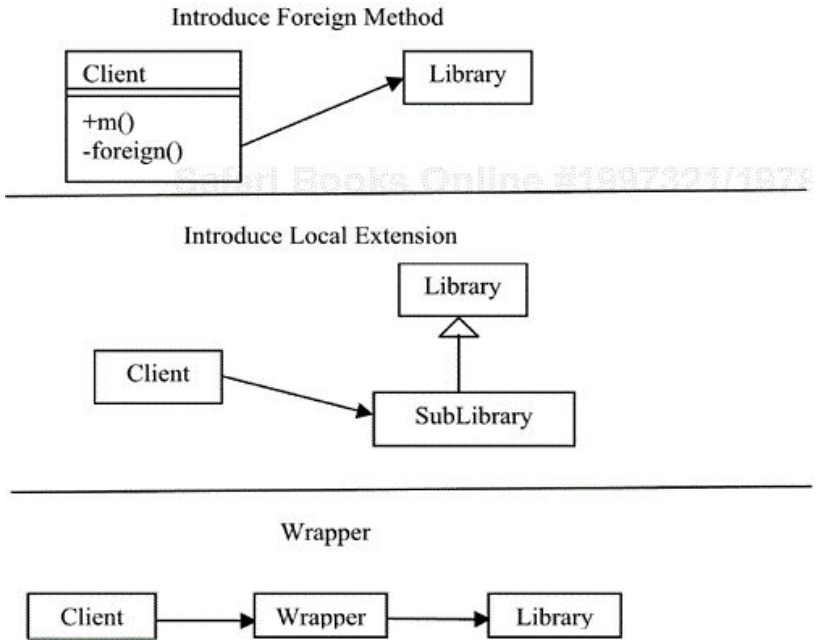
Incomplete Library Class

- You’re using a library class, and there’s a feature you wish were on that class, but it’s not.
- Since you don’t own the code, you may be unable or unwilling to change it.
- If the owner of the library adds support for what you need, the problem is solved.



Incomplete Library Class

- Refactoring:



  - **Removal** 💰: Reduces duplication.

Incomplete Library Class

- Refactoring tips:
  - We recommend **introducing a layer covering the library** (see Adapter Pattern and ACLs).
- **Additional 💰 of this option**:
  - The interface would talk about your domain.
  - A place to contain changes and avoid leaks in the domain.
  - Limits the surface of the interface.
  - “**Don’t mock interfaces that you don’t own**” compliant.