Agile Software Development

Lecturer: Raman Ramsin

Lecture 13

Refactoring – Part 3
Dealing with Generalization: *Pull Up Constructor Body*

- **Pull Up Constructor Body**
  - You have constructors on subclasses with mostly identical bodies.
  - Create a superclass constructor; call this from the subclass methods.

```java
class Manager extends Employee...
    public Manager (String name, String id, int grade) {
        _name = name;
        _id = id;
        _grade = grade;
    }

    public Manager (String name, String id, int grade) {
        super (name, id);
        _grade = grade;
    }
```
Dealing with Generalization: *Extract Subclass/Superclass*

**Extract Subclass**
- A class has features that are used only in some instances.
- *Create a subclass for that subset of features.*

**Extract Superclass**
- You have two classes with similar features.
- *Create a superclass and move the common features to the superclass.*
Dealing with Generalization: *Extract Interface*

- **Extract Interface**
  - Several clients use the same subset of a class's interface, or two classes have part of their interfaces in common.
  - *Extract the subset into an interface.*
Dealing with Generalization: *Collapse Hierarchy*

- **Collapse Hierarchy**
  - A superclass and subclass are not very different.
  - *Merge them together.*
Dealing with Generalization: *Form Template Method*

**Form Template Method**

- You have two methods in subclasses that perform similar steps in the same order, yet the steps are different.

- *Get the steps into methods with the same signature, so that the original methods become the same. Then you can pull them up.*
Dealing with Generalization: *Form Template Method*

![Diagram showing the relationship between Site, Residential Site, Lifeline Site, getBillableAmount, getBaseAmount, and getTaxAmount.](image)

- **Site**:
  - `double base = _units * _rate * 0.5;
  - double tax = base * Site.TAX_RATE * 0.2;
  - return base + tax;

- **Residential Site**:
  - getBillableAmount
  - getBaseAmount

- **Lifeline Site**:
  - getBillableAmount

- **Site**
  - getBillableAmount
  - getBaseAmount
  - getTaxAmount

- **Residential Site**
  - getBaseAmount
  - getTaxAmount

- **Lifeline Site**
  - getBaseAmount
  - getTaxAmount
Dealing with Generalization: *Replace Inheritance with Delegation*

- **Replace Inheritance with Delegation**
  - A subclass uses only part of a superclass’s interface or does not want to inherit data.
  - *Create a field for the superclass, adjust methods to delegate to the superclass, and remove the subclassing.*
Dealing with Generalization: *Replace Delegation with Inheritance*

- **Replace Delegation with Inheritance**
  - You're using delegation and are often writing many simple delegations for the entire interface.
  - *Make the delegating class a subclass of the delegate.*

![Diagram showing the transition from delegation to inheritance]
Big Refactorings: *Tease Apart Inheritance*

- **Tease Apart Inheritance**

  - You have an inheritance hierarchy that is doing two jobs at once.

  - *Create two hierarchies and use delegation to invoke one from the other.*
Big Refactorings: *Tease Apart Inheritance*
Big Refactorings: *Convert Procedural Design to Objects*

- **Convert Procedural Design to Objects**
  - You have code written in a procedural style.
  - *Turn the data records into objects, break up the behavior, and move the behavior to the objects.*
Big Refactorings: *Convert Procedural Design to Objects*
Big Refactorings: *Separate Domain from Presentation*

- **Separate Domain from Presentation**
  - You have GUI classes that contain domain logic.
  - *Separate the domain logic into separate domain classes.*
Big Refactorings: *Extract Hierarchy*

- **Extract Hierarchy**
  - You have a class that is doing too much work, at least in part through many conditional statements.
  - *Create a hierarchy of classes in which each subclass represents a special case.*
Big Refactorings: *Extract Hierarchy*
References
