Lecture 17: Refactoring

Inf2: SEPP

Adriana Sejfia

School of Informatics, University of Edinburgh

This Lecture

- Refactoring — seen by some processes (e.g. XP) as integral to development
  - The problem
  - Definitions
  - Why?
  - When?
  - What?
  - Refactoring in different IDEs: IntelliJ, Eclipse
  - Safe refactoring
  - Bad smells in code

The Problem

- As code evolves, its quality naturally decays

The Problem

- As code evolves, its quality naturally decays
    - Initially code implements a good design

The Problem

- As code evolves, its quality naturally decays
  - Initially code implements a good design
  - Changes are often local, without full understanding of the context

The Problem

- As code evolves, its quality naturally decays
  - Initially code implements a good design
  - Changes are often local, without full understanding of the context
  - With loss of structure, code becomes harder to follow, modify, and debug

The Problem

- As code evolves, its quality naturally decays
  - Initially code implements a good design
  - Changes are often local, without full understanding of the context
  - With loss of structure, code becomes harder to follow, modify, and debug
- Refactoring is about restoring good design in a disciplined way

The Problem

- As code evolves, its quality naturally decays
  - Initially code implements a good design
  - Changes are often local, without full understanding of the context
  - With loss of structure, code becomes harder to follow, modify, and debug
- Refactoring is about restoring good design in a disciplined way
  - Expertise captured in refactoring patterns
  - Enable rapid learning and tool support

Refactoring: Definition

- Refactoring (noun): a change made to the internal structure of software to make it…
  - easier to understand,

Refactoring: Definition

- Refactoring (noun): a change made to the internal structure of software to make it…
  - easier to understand, and
  - cheaper to modify

Refactoring: Definition

- Refactoring (noun): a change made to the internal structure of software to make it…
  - easier to understand, and
  - cheaper to modify
- …without changing its observable behaviour

Refactoring: Definition

- Refactoring (noun): a change made to the internal structure of software to make it…
    - easier to understand, and
    - cheaper to modify
- …without changing its observable behaviour
- Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behaviour — Fowler, 2000

Refactoring: Definition

- Refactoring (noun): a change made to the internal structure of software to make it…
  - easier to understand, and
  - cheaper to modify
- …without changing its observable behaviour
- Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behaviour — Fowler, 2000
- "Refactoring" also used to refer to the general activity

Why Refactor?

- Makes software easier to understand
  - Your code, by you
  - Your code, by others
  - Others' code, by you

Why Refactor?

- Makes software easier to understand
  - Your code, by you
  - Your code, by others
  - Others' code, by you
- Helps you make subsequent modifications quicker

Why Refactor?

- Makes software easier to understand
  - Your code, by you
  - Your code, by others
  - Others' code, by you
- Helps you make subsequent modifications quicker
- Helps you find bugs — design becomes clearer, bugs easier to see

Why Refactor?

- Makes software easier to understand
  - Your code, by you
  - Your code, by others
  - Others' code, by you
- Helps you make subsequent modifications quicker
- Helps you find bugs — design becomes clearer, bugs easier to see
- **The result:** refactoring helps you program faster

When to Refactor?

- Refactoring was once seen as a kind of maintenance…

When to Refactor?

- Refactoring was once seen as a kind of maintenance…
    - You've inherited legacy code that's a mess
    - A new feature requires a change in the architecture

When to Refactor?

- Refactoring was once seen as a kind of maintenance…
  - You've inherited legacy code that's a mess
  - A new feature requires a change in the architecture
- But can also be an integral part of the development process

When to Refactor?

- Refactoring was once seen as a kind of maintenance…
    - You've inherited legacy code that's a mess
    - A new feature requires a change in the architecture
- But can also be an integral part of the development process
    - Agile methodologies (e.g. XP) advocate continual refactoring
    - XP maxim: "Refactor mercilessly"

What Does Refactoring Do?

- A refactoring is a small transformation which preserves correctness

What Does Refactoring Do?

- A refactoring is a small transformation which preserves correctness
- Examples: Catalogue of over 90 refactorings by Martin Fowler: http://refactoring.com/catalog/

What Does Refactoring Do?

- A refactoring is a small transformation which preserves correctness
- Examples: Catalogue of over 90 refactorings by Martin Fowler: http://refactoring.com/catalog/
- A sample:
    - Add Parameter

What Does Refactoring Do?

- A refactoring is a small transformation which preserves correctness
- Examples: Catalogue of over 90 refactorings by Martin Fowler: http://refactoring.com/catalog/
- A sample:
  - Add Parameter
  - Change Bidirectional Association to Unidirectional

What Does Refactoring Do?

- A refactoring is a small transformation which preserves correctness
- Examples: Catalogue of over 90 refactorings by Martin Fowler: http://refactoring.com/catalog/
- A sample:
  - Add Parameter
  - Change Bidirectional Association to Unidirectional
  - Extract Variable (Introduce Explaining Variable)

What Does Refactoring Do?

- A refactoring is a small transformation which preserves correctness
- Examples: Catalogue of over 90 refactorings by Martin Fowler: http://refactoring.com/catalog/
- A sample:
  - Add Parameter
  - Change Bidirectional Association to Unidirectional
  - Extract Variable (Introduce Explaining Variable)
  - Replace Conditional with Polymorphism

Example: Extract Variable

```
Before:
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1) &&
     wasInitialized() && resize > 0 )
{
    // do something
}
```

Example: Extract Variable

```
Before:
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1) &&
     wasInitialized() && resize > 0 )
{
    // do something
}
```

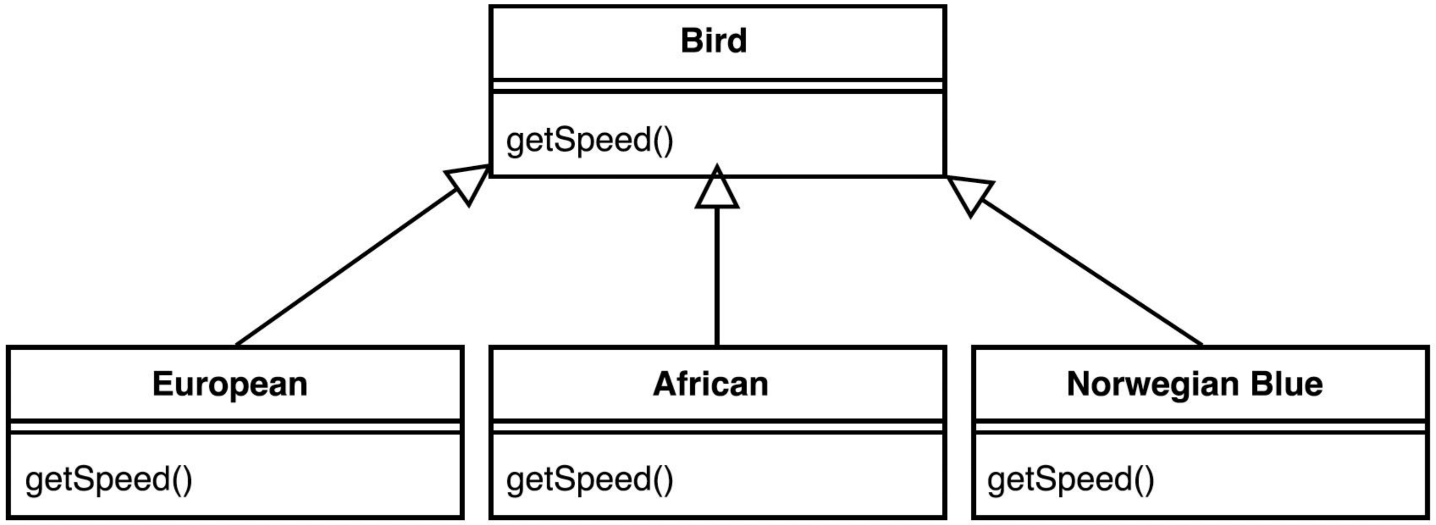
```
After:
final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;
final boolean wasResized = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
    // do something
}
```

Replace Conditional with Polymorphism — Before

```
Code:
double getSpeed() {
    switch (_type) {
        case EUROPEAN:
            return getBaseSpeed();
        case AFRICAN:
            return getBaseSpeed()- getLoadFactor() * _numberOfCoconuts;
        case NORWEGIAN_BLUE:
            return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException("Should be unreachable");
}
```

Replace Conditional with Polymorphism — After



IntelliJ Refactoring

- Press Ctrl+Alt+Shift+T to see all available refactorings
- Or use a keyboard shortcut for a specific refactoring

IntelliJ Refactoring

- Press Ctrl+Alt+Shift+T to see all available refactorings
- Or use a keyboard shortcut for a specific refactoring
- Other features:
  - Preview before applying (for some refactorings)
  - Conflicts displayed if there are problems
  - Ability to exclude or remove unnecessary changes

Most Popular IntelliJ Refactorings

- Safe delete: Alt+Delete
- Copy/Move: F5 / F6
- Extract method: Ctrl+Alt+M
- Extract constant: Ctrl+Alt+C
- Extract field: Ctrl+Alt+F

Most Popular IntelliJ Refactorings (cont.)

- Extract parameter: Ctrl+Alt+P
- Introduce variable: Ctrl+Alt+V
- Rename: Shift+F6
- Inline: Ctrl+Alt+N
- Change signature: Ctrl+F6

Eclipse Refactoring

Eclipse has a built-in refactoring tool (on the Refactor menu)

Eclipse Refactoring I: Renaming & Physical Reorganization

- A variety of simple changes, applied semantically (not syntactic search-and-replace):
  - Rename Java elements — classes, fields, methods, local variables
  - On class rename, import directives are updated automatically
  - On field rename, getter and setter methods are also renamed
  - Move classes between packages

Eclipse Refactoring II: Modifying Class Relationships

- Heavier-weight changes — less used, but seriously useful when they are
  - Move methods or fields up and down a class inheritance hierarchy

Eclipse Refactoring II: Modifying Class Relationships

- Heavier-weight changes — less used, but seriously useful when they are
    - Move methods or fields up and down a class inheritance hierarchy
    - Extract an interface from a class

Eclipse Refactoring II: Modifying Class Relationships

- Heavier-weight changes — less used, but seriously useful when they are
  - Move methods or fields up and down a class inheritance hierarchy
  - Extract an interface from a class
  - Turn an anonymous class into a nested class

Eclipse Refactoring III: Intra-Class Refactorings

- The most used types — rearranging code within a class to improve readability
  - Extract Method: pull a code block into a new method
    - Good for shortening a method or making a block reusable
    - Can also extract local variables and constants

Eclipse Refactoring III: Intra-Class Refactorings

- The most used types — rearranging code within a class to improve readability
  - Extract Method: pull a code block into a new method
    - Good for shortening a method or making a block reusable
    - Can also extract local variables and constants
  - Encapsulate fields in accessor methods

Eclipse Refactoring III: Intra-Class Refactorings

- The most used types — rearranging code within a class to improve readability
    - Extract Method: pull a code block into a new method
        - Good for shortening a method or making a block reusable
        - Can also extract local variables and constants
    - Encapsulate fields in accessor methods
    - Change the type of a method parameter or return value

Refactoring example

```
 public void printOrderSummary(String customerName, int[] itemPrices) {
   // Print header
   System.out.println("====================");
   System.out.println("Customer: " + customerName);
   System.out.println("====================");

   // Calculate and print total
   int total = 0;
   for (int price : itemPrices) {
      total += price;
   }

   System.out.println("Total: $" + total);
   // Print footer
   System.out.println("====================");
   System.out.println("Thank you for your order!");
   System.out.println("===================="); }
}
```

Safe Refactoring

- How do you know refactoring hasn't changed or broken something?
  - Formally: a refactoring operation may have been proved safe

Safe Refactoring

- How do you know refactoring hasn't changed or broken something?
  - Formally: a refactoring operation may have been proved safe
- More realistically:
  - test → refactor → test

Safe Refactoring

- How do you know refactoring hasn't changed or broken something?
    - Formally: a refactoring operation may have been proved safe
- More realistically:
    - test → refactor → test
- This works better the more tests you have
    - Ideally: unit tests for every class

Bad Smells in Code

- Suggest that the quality of your code is decaying

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
  - Duplicated code

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
  - Duplicated code
  - Long method

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
  - Duplicated code
  - Long method
  - Large class

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
  - Duplicated code
  - Long method
  - Large class
  - Long parameter list

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
  - Duplicated code
  - Long method
  - Large class
  - Long parameter list
  - Lazy class

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
  - Duplicated code
  - Long method
  - Large class
  - Long parameter list
  - Lazy class
  - Long message chains

Bad Smells in Code

- Suggest that the quality of your code is decaying
- Examples:
    - Duplicated code
    - Long method
    - Large class
    - Long parameter list
    - Lazy class
    - Long message chains

Catalogues of bad smells explain how to recognise them and what refactorings can help

Reading

- Essential: Browse Fowler's page at http://refactoring.com/
    - Some of his book Refactoring is available on Google Books
- Essential: Search 'code smells' — catalogue at https://refactoring.guru/refactoring/smells
- Recommended: IntelliJ refactoring docs — https://www.jetbrains.com/help/idea/refactoring-source-code.html
- Recommended (Eclipse): Eclipse Java development user guide
    - https://www.linuxtopia.org/online_books/eclipse_documentation/eclipse_java_development_guide/