

Inf2: SEPP

Lecture 17: Refactoring

Cristina Adriana Alexandru

School of Informatics
University of Edinburgh

Last two lectures:

Construction:

- ▶ High quality code
- ▶ Version control and system building

This lecture:

Refactoring, seen by some software development processes (e.g. XP) as integral part of the development process

- ▶ 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

- ▶ Initially code implementing a good design
- ▶ Changes often local, without full understanding of the context
- ▶ With loss of structure, code becomes harder to follow, modify, debug

Refactoring is about restoring good design in a disciplined way

- ▶ Expertise on refactoring captured in *refactoring patterns*
 - ▶ Enable rapid learning
 - ▶ Tool support

Refactoring definition

Refactoring (noun) is 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, *Refactoring*, 2000

Refactoring (noun) also used to refer to the general activity

Why refactor?

Refactoring

- ▶ 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 and bugs easier to see

The **result**: refactoring helps you program *faster*

When to refactor?

Refactoring was once seen as a kind of maintenance. . .

For example:

- ▶ You've inherited legacy code that's a mess.
- ▶ A new feature is required that necessitates 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.

There are many examples.

For a catalogue of over 90 assembled by Martin Fowler, see <http://refactoring.com/catalog/>.

A sample:

- ▶ Add Parameter
- ▶ Change Bidirectional Association to Unidirectional
- ▶ Extract Variable (Introduce Explaining Variable)
- ▶ Replace Conditional with Polymorphism

Extract Variable

Change

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

to

```
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
}
```

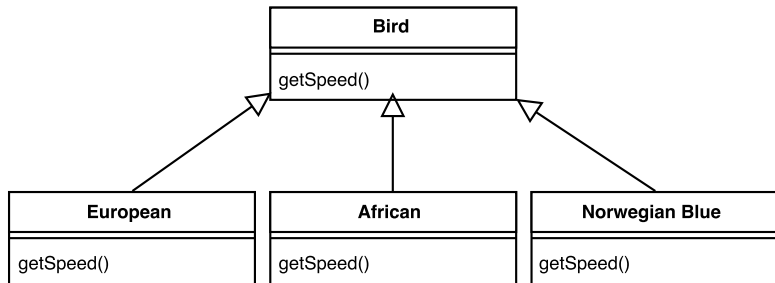
Replace Conditional with Polymorphism I

Change

```
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 II

to



IntelliJ Refactoring

To see available refactorings in IntelliJ IDEA, you need to select an item to refactor and press `Ctrl+ Alt+ Shift+ T`, or use a keyboard shortcut for a specific refactoring.

Other features:

- ▶ For some refactorings, previewing
- ▶ If there are problems with the refactoring, conflicts are displayed
- ▶ For both of the above, excluding or removing any 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
- ▶ Extract parameter: Ctrl+ Alt+ P
- ▶ Introduce variable: Ctrl+ Alt+ V
- ▶ Rename: Shift+ F6
- ▶ Inline: Ctrl+ Alt +N
- ▶ Change signature: Ctrl+ F6

See more here as well as in subpages: <https://www.jetbrains.com/help/idea/refactoring-source-code.html>

Eclipse Refactoring

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

Many of its refactoring operations can be grouped into three broad types ...

Eclipse Refactoring I:

Renaming and physical reorganization

A variety of simple changes.

For example:

- ▶ Rename Java elements (classes, fields, methods, local variables)
 - ▶ On class rename, `import` directives updated
 - ▶ On field rename, getter and setter methods also renamed
- ▶ Move classes between packages

Eclipse applies these changes *semantically*

- ▶ Much better than syntactic search-and-replace

Eclipse Refactoring II:

Modifying class relationships

Heavier weight changes. Less used, but seriously useful when they are used.

For example:

- ▶ 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 of refactoring: rearranging code within a class to improve readability etc.

For example:

- ▶ Extract Method: pull code block into new method.
 - ▶ Good for shortening method or making block reusable
 - ▶ Can also extract local variables and constants
- ▶ Encapsulating fields in accessor methods.
- ▶ Change the type of a method parameter or return value

Safe refactoring

How do you know refactoring hasn't changed/broken something?

Perhaps somebody has *proved* that a refactoring operation is 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.

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 through Fowler's page at <http://refactoring.com/>. Some of his book *Refactoring* is available on Google Books e.g., details of some of the refactorings in the catalogue.

Essential: Search *code smells*. One catalogue can be found at <https://refactoring.guru/refactoring/smells>.

Reading

Recommended: Browse through the Code Refactoring page and subpages of IntelliJ IDEA for full information on IntelliJ's current capabilities: <https://www.jetbrains.com/help/idea/refactoring-source-code.html>

Recommended: If you are using Eclipse, browse through the Eclipse *Java development user guide* for full information on Eclipse's current capabilities: https://www.linuxtopia.org/online_books/eclipse_documentation/eclipse_java_development_guide/topic/org.eclipse.jdt.doc.user/concepts/eclipse_java_concept-refactoring.htm