Refactoring in Small Steps

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

At the end of Today's Lecture you will be able to:

- Apply to Refactoring to Test Code

ROAR

Inspiration

“If you can get today’s work done today, but you do it in such a way that you can’t possibly get tomorrow’s work done tomorrow, then you lose.” – Martin Fowler

Idaho State University
Computer Science

Let's Remember to Refactor

- Refactoring applies to:
  - **functional code**
  - **test code**

ROAR

Idaho State University
Computer Science

Problems with Current Tests

```
@Test public void oneVariable() throws Exception {
    Template template = new Template("Hello, ${name}");
    template.set("name", "Reader");
    assertEquals("Hello, Reader", template.evaluate());
}
@Test public void differentTemplate() throws Exception {
    Template template = new Template("Hi, ${name}");
    template.set("name", "someone else");
    assertEquals("Hi, someone else", template.evaluate());
}
@Test public void multipleVariables() throws Exception {
    Template template = new Template("${one}, ${two}, ${three}");
    template.set("one", "1");
    template.set("two", "2");
    template.set("three", "3");
    assertEquals("1, 2, 3", template.evaluate());
}
@Test public void unknownVariablesAreIgnored() throws Exception {
    Template template = new Template("Hello, ${name}");
    template.set("name", "Reader");
    template.set("doesnotexist", "Hi");
    assertEquals("Hello, Reader", template.evaluate());
}
```


ROAR

Idaho State University
Computer Science

Problems

- Redundant creation of Template instances
- Redundant assertions
- 1st and 2nd tests are basically the same
- Test 1 and 4 have same values

```
@Test public void oneVariable() throws Exception {
   Template template = new Template("Hello, ${name}");
   template.set("name", "Reader");
   assertEquals("Hello, Reader", template.evaluate());
}
@Test public void differentTemplate() throws Exception {
   Template template = new Template("Hi, ${name}");
   template.set("name", "someone else");
   assertEquals("Hi, someone else", template.evaluate());
}
@Test public void multipleVariables() throws Exception {
   Template template = new Template("${one}, ${two}, ${three}");
   template.set("one", "1");
   template.set("two", "2");
   template.set("three", "3");
   assertEquals("1, 2, 3", template.evaluate());
}
@Test public void unknownVariablesAreIgnored() throws Exception {
   Template template = new Template("Hello, ${name}");
   template.set("name", "Reader");
   template.set("doesnotexist", "Hi");
   assertEquals("Hello, Reader", template.evaluate());
}
```

ROAR

Refactor Test Code

```
public class TestTemplate {
  private Template template;
  @Before
  public void setUp() throws Exception {
    template = new Template ("${one}, ${two}, ${three}");
    template.set("one", "1");
    template.set("two", "2");
    template.set("three", "3");
  }

  @Test
  public void multipleVariables() throws Exception {
    assertTemplateEvaluatesTo("1, 2, 3");
  }

  @Test
  public void unknownVariablesAreIgnored() throws Exception {
    template.set("doesnotexist", "whatever");
    assertTemplateEvaluatesTo("1, 2, 3");
  }

  private void assertTemplateEvaluatesTo(String expected) {
    assertEquals(expected, template.evaluate());
```

Idaho State University
Computer Science

Adding Some Error Handling

- A variable without a value?
- Adding exception test
- Note different approaches to testing exceptions
  - try-catch block with fail() vs. @Test(expected = ...)

```
@Test
public void missingValueRaisesException() throws Exception {
    try {
        new Template ("${foo}").evaluate();
        fail("evaluate() should throw an exception if "
                + "a variable does not have a value!");
    } catch(MissingValueException expected) {
    }
}
```


ROAR

Idaho State University
Computer Science

Extract Method Refactoring

```
public String evaluate() {
  String result = templateText;
  for (Entry<String, String> entry : variables.entrySet()) {
    String regex = "\\$\\{" + entry.getKey() + "\\}";
    result = result.replaceAll (regex, entry.getValue());
  }
  if (result.matches(".*\\$\\{.+\\}.*")) {
    throw new MissingValueException();
  }
  return result;
}
```

- The `if` block checks if `result` still has a variable with no value

ROAR

Extract Method Refactoring

- We refactor so `evaluate()` does only one thing

```
public String evaluate() {
  String result = templateText;
  for (Entry<String, String> entry : variables.entrySet()) {
    String regex = "\\$\\{" + entry.getKey() + "\\}";
    result = result.replaceAll (regex, entry.getValue());
  }
  checkForMissingValues(result);
  return result;
}

private void checkForMissingValues (String result) {
  if (result.matches(".*\\$\\{.+\\}.*")) {
    throw new MissingValueException();
  }
}
```

More Refactoring

```
public String evaluate() {
  String result = replaceVariables();
  checkForMissingValues(result);
  return result;
}
```

- `evaluate()` method's internals become better balanced

More Refactoring

```
private String replaceVariables() {
  String result = templateText;
  for (Entry<String, String> entry : variables.entrySet()) {
    String regex = "\\$\\{" + entry.getKey() + "\\}";
    result = result.replaceAll(regex, entry.getValue());
  }
return result;
}
```

- New method `replaceVarialbes()` is simple and has a single, clear purpose

More Refactoring

```
private void checkForMissingValues(String result) {
  if (result.matches(".*\\$\\{.+\\}.*")) {
    throw new MissingValueException();
  }
}
```

- Must re-run all the tests to ensure nothing broke

Idaho State University
Computer Science

A Truly Difficult Special Case

- What happens in the special case that a value has a special character such as $, {, or }?
  - These are the kinds of non-happy path tests TDD often skips
- Implementing this test breaks the current implementation:

```
@Test
public void variablesGetProcessedJustOnce() throws Exception {
    template.set("one", "${one}");
    template.set("two", "${three}");
    template.set("three", "${two}");
    assertTemplateEvaluatesTo("${one}, ${three}, ${two}");
}
```

- regexp throws an `IllegalArgumentException`
  - Requiring a major design change

ROAR

What is a spike?

- A detour to **learn something** new
  - Package, details of an API, etc.
  - Whether a proposed design will work
- Spikes are **experimental** in nature
- **Self education**—increase knowledge, skills, or abilities

The Problem

- Existing design replace variables via simple matching
  - for all variables v, replace `${v}` with its value: `result = result.replaceAll(regex, entry.getValue())`
- Failing test from Chapter 2: Sets the value to "${one}, ${two}, ${three}"

```
@Test
public void variablesGetProcessedJustOnce() throws Exception {
  template.set("one", "${one}");
  template.set("two", "${three}");
  template.set("three", "${two}");
  assertTemplateEvaluatesTo("${one}, ${three}", ${two});
}
```

Tweaking the current design won't make this test pass

Idaho State University
Computer Science

Exploring a potential solution

- Break the templates into “**segments**”
- **Prototyping** with spikes
  - A spike is a detour to learn
  - In the template example, we learn more about using regex
- Learn by writing tests (**learning tests**)
  - Need to figure out an API?
    - Write some tests that use the API

Learn on a short detour, then apply

ROAR

Core Idea

- Use regexp to **break** the following string:

```
"${greeting} ${fname},
 thank you for your interest in ${product}"
```

- Into the following **5 pieces**:
  - `"${greeting}"`
  - `"${fname}"`
  - `", Thank you for your interest in"`
  - `"${product}"`
  - `"."`
- Now the variables can easily be identified and **replaced**
  - regexp will **not** explode if values have `'$'`, `'{'`, or `'}'`

Are there any questions?