Declarative Mocking

Hesam Samimi, Rebecca Hicks, Ari Fogel, Todd Millstein. ISSTA 2013.

Presented by: Kim Phan
What is Declarative Mocking?

• The creation of expressive and reliable mock objects with relatively little effort.

• Mock Objects
  • Serves as a dummy or stub of the actual functionality of the code it is mocking.
  • Limited benefits
    • Does not contain actual functionality.
    • Requires explicit results when indirectly testing.
    • Tests are fragile, error-prone, and difficult to understand or reuse.
class MySet implements Set {
    List elems;
    void add(Object o) {
        if (!elems.contains(o))
        
            elems.add(o);
    }
    void testAdd() {
        List mockList = mock(List.class);
        MySet s = new MySet(mockList);
        when(mockList.contains(0))
        .thenReturn(false);
        s.add(0);
        verify(mockList, times(1)).add(0);
        when(mockList.contains(0))
        .thenReturn(true);
        s.add(0);
        // shouldn't add duplicates:
        verify(mockList, times(1)).add(0);
    }
}
Goal

1. Easily build mock objects that directly reflect important parts of the functionality.
2. Have mocks that are less coupled with specific implementation.
Declarative Mocking

• Executable Specifications
  • Removes the need for imperative code.
  • Allows the tester to specify the intended precondition.

• Declarative Execution

• Specifications
  • Directly expresses what behavior is desired without specifying how that behavior is achieved.

• Constraint solver
Executable Specification

```java
class MockList implements List {
    Object[] elems, int size;
    spec int size() { return size; }
    pure boolean contains(Object o)
        ensures result <=>
        some int i: 0 .. size - 1 |
            elems[i].equals(o);
    void add(Object o)
        modifies fields
            MockList:elems, MockList:size
        ensures size == old(size) + 1
        && elems[old(size)] == o
        && all int i: 0 .. old(size) - 1 |
            elems[i] == old(elems[i]);
}
```

- PBnJ, a java extension that supports executable specifications.
- Propositional Satisfiability (SAT) solver called Kodkod.
- Data mocking
Testing

class MySet implements Set {
    List elems;
    int size() { return elems.size(); }
    void add(Object o) {
        if (!elems.contains(o))
            elems.add(o);
    }
    void testAdd() {
        List mockList = new MockList();
        MySet s = new MySet(mockList);
        s.add(0);
        s.add(0);
        // shouldn't add duplicates:
        assert (s.size() == 1);
    }
}

Mocking

class MySet {
    List elems;
    void test1() {
        assume elems.size() > 0;
        // now run the test ...
    }
    void test2() {
        assume elems.contains(null);
        // now run the test ...
    }
}
Evaluating Declarative Mocking

- Tools: PBNJ
- Tests:
  - Using declarative mocking on 4 existing applications to see if it would benefit over traditional mocks.
  - Porting over 6 existing applications from google code that are currently using Mockito to instead use PBNJ.
Study 1

- Jstock – Mocking Webserver Data
- JDBC – Mocking Database behavior and Data
- TFTP – Mocking Errors and Network Nondeterminism in a Client-Server Protocol
- Hadoop – Mocking Cloud Behavior and Environment

- Advantages:
  - Integrity constraints only need to be stated once.
  - Declarative mocking is useful when the mock object is frequent to modifications.

- Disadvantages:
  - Stub’s have low overhead.
  - Specifications can be error prone and hard to debug.
  - Constraint solving is limited in its efficiency and scalability.
Study 2

- Research Questions:
  - Phase A:
    1. What is the overhead for a developer to use declarative mocks, when used to replicate the exact behavior of traditional mocking approaches.
    - Considerable effort overhead.
  - Phase B:
    2. How often and under what scenarios do declarative mocks offer advantages beyond the traditional approaches, with a justifiable amount of additional effort.
    - Reusability
      - Was able to reuse initialization conditions, which reduced effort.
    - Underspecification
      - The specifications increased the coverage of each test while requiring the same amount of developer effort.
Study 2: Results

- Compared the number of lines between the stub and the spec version.
- Other than LOC, is there another metric that could be used to measure the relative effort between declarative mocking and stubs?
Conclusion

• Declarative Mocking is a new approach to creating mock objects that implements executable specifications.
• Is very beneficial for unit testing and can increase test coverage.
• Useful for complex code.
Thank You
Conclusion

• Declarative Mocking is a new approach to creating mock objects that implements executable specifications.
• Is very beneficial for unit testing and can increase test coverage.
• Useful for complex code.