Testing Frameworks (MiniTest: Assert & Spec)

Lecture 21

MiniTest and RSpec

- Many popular testing libraries for Ruby
- MiniTest (replaces older Test::Unit)
  - Comes built-in
  - Looks like JUnit (mapped to Ruby syntax)
  - Well-named!
- RSpec
  - Installed as a library (*i.e.* a gem)
  - Looks different from JUnit (and even Ruby!)
  - Most unfortunate name!
- RSpec view is that test cases *define* expected behavior—they *are* the spec!
  - What is wrong with that view?

Writing MiniTest Tests

- Require runner and UUT

```
require 'minitest/autorun' # the test runner
require 'card'             # the UUT
```

- *Test fixture:* subclass of **`MiniTest::Test`**

```
class TestCard < MiniTest::Test
```

- *Test case:* a method in the fixture
  - Method name *must* begin with **`test_`**

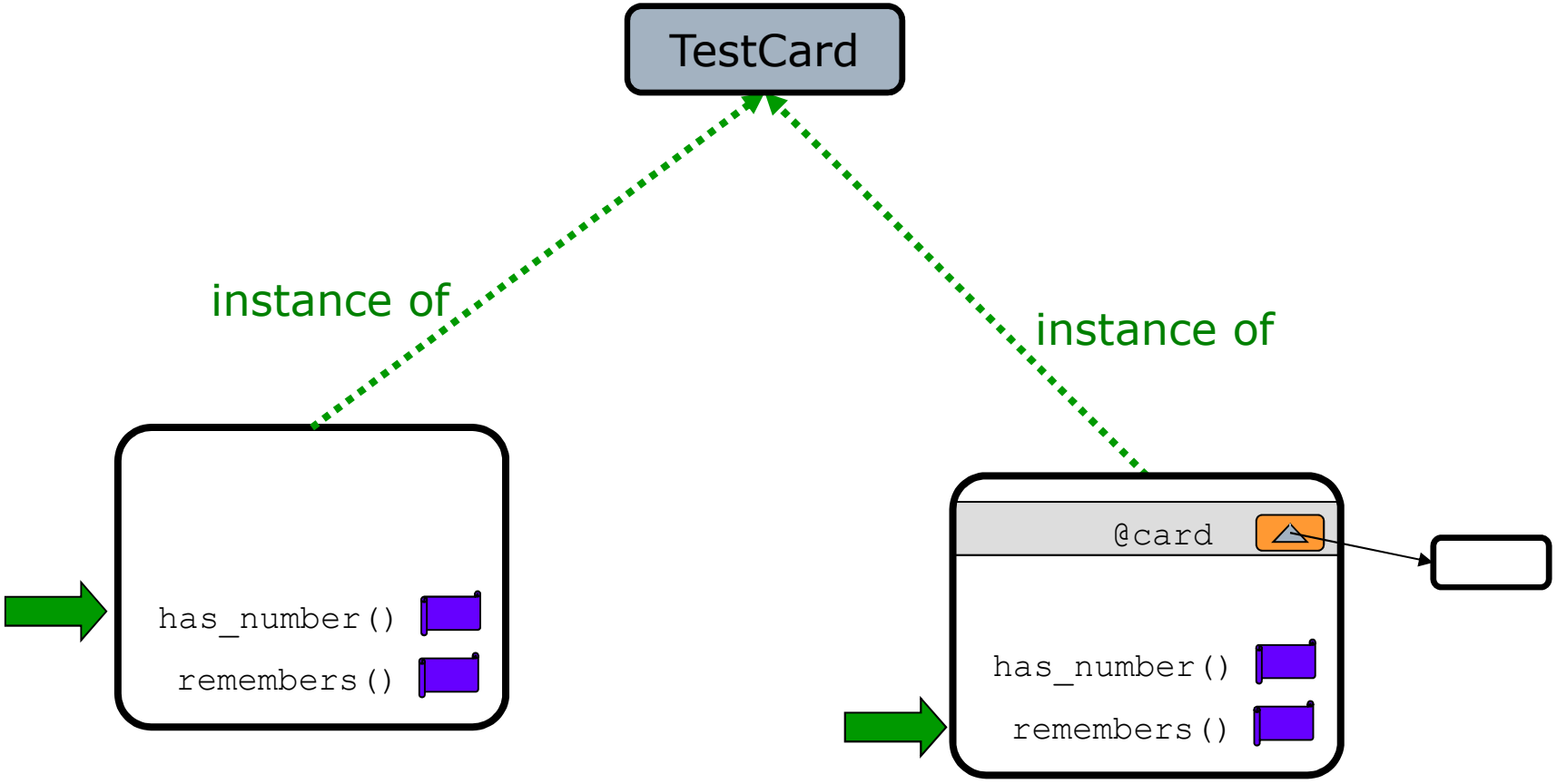
```
def test_identifies_set  …  end
```

  - Contains assertion(s) exercising a single piece of code / behavior / functionality
  - Should be *small* (*i.e.* test one thing)
  - Should be *independent* (*i.e.* of other tests)
- *Test Suite:* a collection of fixtures

Example: test_card.rb


```
require 'minitest/autorun'
require 'card' # assume card.rb on load path

class TestCard < MiniTest::Test

  def test_has_number
    assert_respond_to Card.new, :number
  end

  def test_remembers_number
    @card = Card.new 1, "oval", "open", "red"
    assert_equal 1, @card.number
  end
end
```

Execution Model



Execution Model: Implications

- Separate instances of test class created
  - One instance / test case
- Test cases don't have side effects
  - Passing/failing one test does not affect others
- Can not rely on order of tests
  - Randomized order of execution
  - Controllable with --seed command-line option
  - Also controllable by invoking, in test fixture:
    `i_suck_and_my_tests_are_order_dependent!`
- Fixture: common set-up to all test cases
  - Field(s) for instance(s) of class being tested
  - Factor initialization code into its own method
  - This method must be called **`setup`**

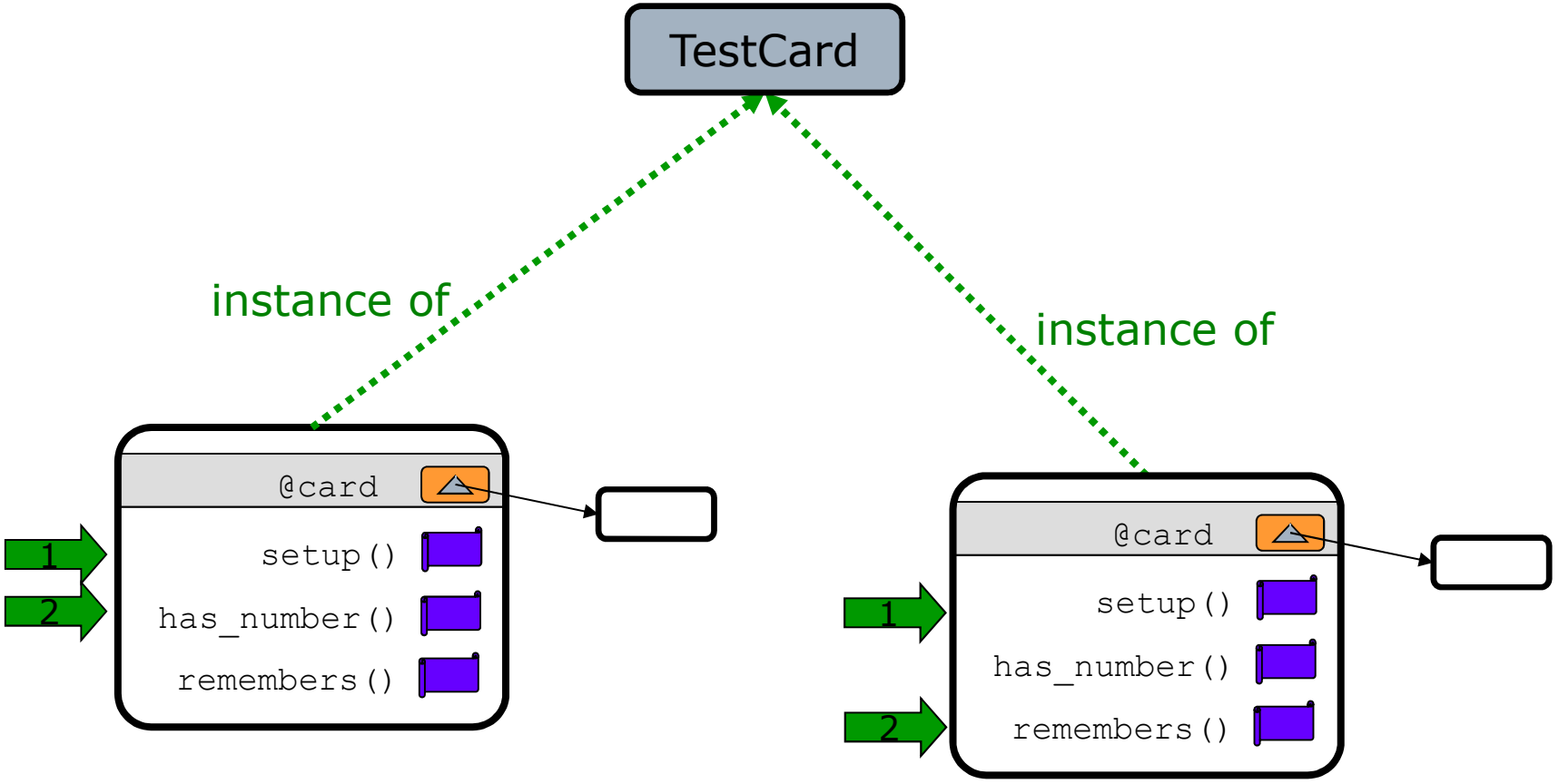
Good Practice: `setup`

- Initialize a fixture with a **`setup`** method (rather than initialize method)
- Reasons:
  - If the code being tested throws an exception *during the setup*, the output is much more meaningful
  - Symmetry with teardown method for cleaning up after a test case

Example: test_card.rb


```
require 'minitest/autorun'
require 'card' # assume card.rb is on load path

class TestCard < Minitest::Test

  def setup
    @card = Card.new 1, "oval", "open", "red"
  end

  def test_has_number
    assert_respond_to @card, :number
  end

  def test_remembers_number
    assert_equal 1, @card.number
  end
end
```

Execution Model



MiniTest Assertion Methods

- Most have two versions: **`assert`** & **`refute`**
  - Example: **`assert_nil`**, **`refute_nil`**
  - No need for negation (use **`refute`** instead)
- Most take an optional message

```
assert_empty Library.new,
                "A new library contains no books"
```

  - Message appears when assertion *fails*
- Specials:
  - **`pass/flunk`** – always passes/fails
  - **`skip`** – skips the rest of the test case
- Performance benchmarking also available

Asserting Equality

- Assert two objects are == equal

```
assert_equal expected, actual
```

  - Compares *object* values (*i.e.* == in Ruby)
  - Failure produces useful output

```
TestCard#test_total_number_of_cards
Expected: 81
Actual: 27
```

  - Compare with **`assert exp == actual`**

```
TestCard#test_shuffle_is_permutation
Failed assertion, no message given
```

- Assert two objects are aliased

```
assert_same @table.north, @players.first
```

  - Compares *reference* values (*i.e.* **`.equal?`**)

Good Practice: Comparing Floats

- Never compare floating point numbers directly for equality

```
assert_equal 1.456, calculated,
  "Low-density experiment"
```

  - Numeric instabilities make exact equality problematic for floats

- Better: Equality with tolerance

```
assert_in_delta Math::PI, (22.0 / 7.0),
  0.01, "Archimedes algorithm"
assert_in_epsilon Math::PI, (22.0 / 7.0),
  0.1, "Archimedes algorithm"
```

  - Delta for absolute error, epsilon for relative error

Common Assertions

- Boolean condition: **assert** (**refute**)

```
assert @books.all {|b| b.available?}
```

- Is nil: **assert_nil** (**refute _nil**)
  - Checks the result of **#nil?**

```
refute_nil @library.manager
# ie refute @library.manager.nil?
```

- Is empty: **assert_empty** (**refute_emp**)
  - Checks the result of **#empty?**

```
assert_empty Library.new
# ie assert Library.new.empty?
```

More Assertions

- String matches a regular expression

```
assert_match /CSE.*/, @course.name
```

- Collection includes a particular item

```
assert_includes @library, @book
```

- Object is of a particular type

```
assert_instance_of String, @book.title
```

- Object has a method

```
assert_respond_to @student, :alarm
```

- Block raises an exception

```
assert_raises ZeroDivisionError do
  @library.average_book_cost
end
```

Good Practice: Organization

- Keep tests in the same *project* as the code
  - They are part of the build, the repo, *etc.*
  - Helps to keep tests current
- Separate tests and implementation
  - /set/lib – contains card.rb (implementation)
  - /set/tests – contains test_card.rb (tests)
- Name test classes consistently
  - **`TestCard`** tests **`Card`**
- Test fixture is a Ruby program

```
[setapp] $ ruby tests/test_card.rb
```

  - Test needs to be able to find UUT (require)
  - Add location of UUT to load path

```
[setapp] $ ruby -I lib tests/test_card.rb
```

Alternative Syntax:

- Problem: Cumbersome method names

```
test_shuffle_changes_deck_configuration
```

- Solution: exploit Ruby language flexibility in API of testing library
  - Methods are available that change the syntax and structure of test cases
  - *Domain-specific language* (DSL) for tests
- Result: MiniTest::Spec
  - Notation inspired by RSpec

Writing MiniTest::Spec Tests

- Require runner and UUT as usual
- Test fixture ("example group") is a **`describe`** block

```
describe Card "noun being described" do … end
```

  - Can be nested, and identified by string
  - The block contains *examples*
- Test case ("example") is an **`it`** block

```
it "identifies a set"  …  end
```

  - Contains *expectation*(s) on a single piece of code / behavior / functionality
- Expectations are methods on *values* of objects

```
value(@card.number).must_equal 1
expect(@card.number).must_equal 1 # equivalent
_(@card.number).must_equal 1      # equivalent
```

Example: test_card.rb


```
require 'minitest/autorun'
require 'card' # assume card.rb is on load path

describe Card, "game of set" do

  it "has a number" do
    _(Card.new).must_respond_to :number
    # value(Card.new).must_respond_to :number
    # expect(Card.new).must_respond_to :number
  end

  it "remembers its original number" do
    @card = Card.new 1, "oval", "open", "red"
    _(@card.number).must_equal 1
  end
end
```

Expectations vs. Assertions

- Similarity: Positive and negative form

```
must_be_empty # like assert_empty
wont_be_empty # like refute_empty
```

- Difference: Argument order

```
assert_equal expected, actual
_actual.must_equal expected
```

- Difference: No string argument
  - Meaningful output comes from group name and example name

```
Card::game of set#test_0001_has a number
[test_card.rb:14]:
Expected #<Card:0x00564f9a00> (Card) to
respond to #number.
```

(object).must + …

- General expectation: Must be

```
_x.must_be :<=, 10
```

- Many other flavors of expectation…

```
_x.must_equal y
_x.must_be_same_as y
_(@library.manager).must_be_nil
_@shelf.must_be_empty
_@library.must_include @book
_PI.must_be_within_delta (22.0 / 7.0), .01
_(@book.title).must_be_instance_of String
_(@course.name).must_match /CSE.*/
_@student.must_respond_to :alarm
proc {
  @library.average_book_cost
}.must_raise ZeroDivisionError
```

Setup/Teardown

- Methods **before**, **after**

```
describe Student do
  before do
    @buck_id = BuckID.new "4328429"
    @s = Student.new buck_id
  end

  it 'should come to class' do … end
end
```

- Independence is good, but

Let: Lazy Initialization


```
describe Student do
  # both defines a method (student)
  # and memoizes its return value!
  let(:student) { Student.new 1234 }

  describe "sleep deprivation" do
    it "misses class" do
      _(student.awake?).must_equal false
    end
  end
end
```

RSpec: Set up and Use

- Install the rspec gem locally

```
[~] $ gem install rspec
```

- Set up your program to use rspec

```
[myapp] $ rspec --init
```

- Init creates several things in myapp/

```
spec/ # put tests (foo_spec.rb) here
spec/spec_helper.rb # configures paths
.rspec  # default command-line args
```

- Run tests

```
[myapp] $ rspec spec/foo_spec.rb
```

Example Groups and Examples


```
require_relative '../student'

describe Student do                    # example group
  it "can drop a class" do             # example
    ...
  end
  context "when attending lecture" do
    before :each do … end
    it "stays awake during lecture" do
      ...
    end
    it "stores info until exam" do
      ...
    end
  end
end
```

RSpec Expectations

- Verb is "should" (or "should_not")

```
target.should condition # notice space
```

- Examples of condition
  - ==, equal,

```
factor.should equal 34
```

  - be_true, be_false, be_nil, be_empty

```
list.emtpy?.should be_true
```

  - have(n).items, have_at_most(n).items
  - include(item)

```
list.should include(name)
```

  - match(regex)
  - respond_to(method_name)
- Preferred form: expect().to (or not_to)

```
expect(a_result).to eq "OSU"
```

Stubs

- Top-down: testing a class that uses A, B, C
- Problem: We don't have A, B, C
  - Want quick approximations of A, B, C
  - Behave in certain way, returning canned answers
- Solution: Stub method
  - Takes a hash of method names & return values
  - Returns an object with those methods

```
stub_printer = stub :available? => true,
                    :render => nil
```

- Another form adds (or changes) a method/return value of an *existing* object

```
long_str = 'something'
long_str.stub (:length).and_return(1000000)
```

Mocks

- Stubs passively allow the test to go through
- Mocks *monitor* how they are used (and will fail if they aren't used right)

```
it 'should know how to print itself' do
  mock_printer = mock('Printer')
  mock_printer.should_receive
    (:available?).and_return(true)
  mock_printer.should_receive
    (:render).exactly(3).times
  @doc.print (mock_printer).should
    == 'Done'
end
```

Summary

- MiniTest
  - Test fixture: class extending Minitest::Test
  - Test case: method named test_
- Execution model: multiple instances
  - Independence of test cases
- MiniTest::Spec
  - Examples and expectations
  - String descriptions
- RSpec
  - Stubs and mocks