Fetching contributors…

Cannot retrieve contributors at this time

27 lines (20 sloc) | 608 Bytes

```
require_relative 'plane'
require_relative 'transit'

class Airport

  DEFAULT_CAPACITY = 10

  def initialize(capacity = DEFAULT_CAPACITY)
    @planes_at_rest = []
    @capacity = capacity
  end

  attr_accessor :planes_at_rest, :capacity

  def land(plane, transit)
    raise "Too stormy for landing" if transit.stormy?
    raise "The airport is currently full" if @capacity == planes_at_rest.count
    @planes_at_rest << plane
  end

  def take_off(plane, transit)
    raise "Too stormy for take off" if transit.stormy?
    @planes_at_rest.delete(plane)
    transit.planes_in_transit.push(plane)
  end
end
```

makersacademy / airport_challenge

Tree: a438a68dcd airport_challenge / lib / plane.rb

Find file Copy path

Fetching contributors…

Cannot retrieve contributors at this time

7 lines (6 sloc) | 71 Bytes

```ruby
class Plane
  def initialize
    @plane
  end
  attr_reader :plane
end
```

makersacademy / airport_challenge

Tree: a438a68dcd airport_challenge / lib / transit.rb

Find file Copy path

Fetching contributors…

Cannot retrieve contributors at this time

16 lines (10 sloc) | 189 Bytes

```
require_relative 'airport'

class Transit

  attr_accessor :planes_in_transit

  def initialize
    @planes_in_transit = []
  end

  def stormy?
    rand(10) >= 9 ? true : false
  end

end
```

Fetching contributors…

Cannot retrieve contributors at this time

83 lines (73 sloc) | 2.92 KB

```ruby
require 'airport'

describe Airport do

  context 'the airport' do
    it 'should be an empty array to start with' do
      airport = Airport.new
      expect(airport.planes_at_rest).to eq []
    end
    it 'should have a defualt capactiy equal to DEFAULT_CAPACITY if no args called' do
      airport = Airport.new
      expect(airport.capacity).to eq Airport::DEFAULT_CAPACITY
    end
    it 'should have a set capacity equal to the argument when called' do
      airport = double(:airport, capacity: 5)
      expect(airport.capacity).to eq 5
    end
  end

  context 'landing' do
    it 'should respond to land with one argument' do
      airport = Airport.new
      expect(airport).to respond_to(:land).with(2).argument
    end
    it 'should return an array with a plane in it once a plane has landed' do
      airport = Airport.new
      plane = double(:plane)
      clear_transit = double(:transit, stormy?: false)
      expect(airport.land(plane, clear_transit)).to eq [plane]
    end
    it 'should not be able to land if the weather is stormy' do
      plane = double(:plane)
      airport = Airport.new
      stormy_transit = double(:transit, stormy?: true)
      expect { airport.land(plane, stormy_transit) }.to raise_error('Too stormy for landing')
    end
    it 'should not be able to take place if the airport is full' do
      airport = Airport.new(1)
      plane = Plane.new
      plane1 = Plane.new
      clear_transit = double(:transit, stormy?: false)
      airport.land(plane, clear_transit)
      expect { airport.land(plane1, clear_transit) }.to raise_error('The airport is currently full')

      # Using doubles, didn't quite work

      # plane = double(:plane)
      # airport = double(:airport, planes_at_rest: [plane], capacity: 1)
      # clear_transit = double(:transit, stormy?: false)
      # plane1 = double(:plane)
      # allow(airport).to receive(:land)
      # expect { airport.land(plane1, clear_transit) }.to raise_error('The airport is currently full')

    end
  end

  context 'take off' do
    it 'should respond to take off with two arguments' do
      airport = Airport.new
      expect(airport).to respond_to(:take_off).with(2).argument
    end

    it 'the plane should not be at airport after the plane takes off' do
      plane = Plane.new
      airport = Airport.new(5)
      clear_transit = double(:trasnit, stormy?: false, planes_in_transit: [])
      clear_transit = Transit.new
      airport.land(plane, clear_transit)
      airport.take_off(plane, clear_transit)
      expect(airport.planes_at_rest).to_not include(plane)
    end

    it 'plane should not be able to take off if weather in transit is stromy' do
      plane = Plane.new
      airport = Airport.new
      clear_transit = double(:transit, stormy?: false)
      airport.land(plane, clear_transit)
      stormy_transit = double(:transit, stormy?: true)
      expect { airport.take_off(plane, stormy_transit) }.to raise_error('Too stormy for take off')
    end
  end
end
```

makersacademy / **airport_challenge**

Tree: **a438a68dcd** **airport_challenge** / spec / **plane_spec.rb** Find file Copy path

Fetching contributors…

Cannot retrieve contributors at this time

0 lines (0 sloc) | 0 Bytes

Fetching contributors…

Cannot retrieve contributors at this time

28 lines (24 sloc) | 783 Bytes

```
require 'transit'
require 'airport'

describe Transit do
  context 'should get a plane in transit if a plane has taken off' do
    it 'should return a plane' do

      plane = double(:plane)
      airport = Airport.new
      clear_transit = double(:trasnit, stormy?: false, planes_in_transit: [])
      airport.land(plane, clear_transit)
      airport.take_off(plane, clear_transit)
      expect(clear_transit.planes_in_transit).to eq [plane]

    end
  end
  context 'weather' do
    it 'should return not stormy' do
      clear_transit = double(:transit, stormy?: false)
      expect(clear_transit.stormy?).to eq false
    end
    it 'should return stormy' do
      stormy_transit = double(:transit, stormy?: true)
      expect(stormy_transit.stormy?).to eq true
    end
  end
end
```

User stories

As an **air traffic controller** So I can get **passengers** to a **destination** I want to instruct a **plane** to land at an **airport**

As an **air traffic controller** So I can get **passengers** on the way to their **destination** I want to instruct a **plane** to take off from an airport and confirm that it is no longer in the airport

As an **air traffic controller** To **ensure safety** I want to prevent **takeoff** when **weather** is stormy

As an **air traffic controller** To **ensure safety** I want to **prevent landing** when weather is stormy

As an **air traffic controller** To **ensure safety** I want to **prevent landing** when the **airport** is full

As the **system designer** So that the software can be used for many different airports I would like **a default airport capacity** that can be **overridden** as appropriate

Object | Message

As an ATC | I want to instruct a plane to land at an airport, so i can get passengers to a destination As an ATC | I want to instruct a plane to take off from an airport and confirm that it is no longer at the airport, so passengers get to destination

ATC --- instruct plane ---> land at airport ATC --- instruct plane ---> take off from airport --- no longer plane at airport ---> deliver passengers ATC --- instruct plane ---> not to take off if weather is stormy ATC --- instruct plane ---> not to land if weather is stormy ATC --- instruct plane ---> not land at airport --- if airport full