Cucumber

- Introduction to Cucumber
  - Start using Cucumber
  - Along the way intro concepts

Goal

- write a Java library that calculates the cost of groceries at the supermarket
- call this a checkout
- the first release has to be as simple as possible
- It will take two inputs
  1. the prices of available items
  2. the notification of items as they are scanned at the checkout
- The checkout will keep track of the total cost

An Example

- banana 40c
- apple 25c
  - and the only item you scan at the checkout is this:
    - 1 banana -> then the output will be 40c
  - if you scan multiple items:
    - 3 apple -> then the output will be 75c

Create feature

- Cucumber tests are grouped into features
- Features describe what a user will be able to enjoy when using our program

Create appropriate directory structure

```
$ mkdir checkout
$ cd checkout

$ mkdir jars
```

- Download inside jars the latest versions of:
  1. cucumber-core
  2. cucumber-java
  3. cucumber-jvm-deps
  4. gherkin

Starting cucumber

```
$ java -cp "jars/*" cucumber.api.cli.Main -p pretty .
No features found at [.]
0 Scenarios
0 Steps
0m0.000s
```

- cucumber.api.cli.Main -> starts the Cucumber cli
- allows control over how Cucumber searches for tests to run
  1. -p pretty tells cucumber to use the pretty formatter plugin
  2. . is a path that points to where our feature files are located

Creating shell script

```
$ nano cucumber.sh
```

- insert: java -cp "jars/*" cucumber.api.cli.Main -p pretty .

```
$ chmod u+x cucumber
$ ./cucumber.sh
```

- to make it an executable script

A feature file

```
$ mkdir features
# create an empty feature file in the features
$ cd features
$ touch checkout.feature
# on windows
C:\checkout> cd features
C:\checkout\features> type nul > checkout.feature
```

Running feature

- change content cucumber.sh into

```
java -cp "jars/*" cucumber.api.cli.Main -p pretty fea
```

- running cucumber.sh

```
$ ./cucumber.sh
No features found at [features]
0 Scenarios
0 Steps
0m0.000s
```

The output

```
$ ./cucumber.sh
No features found at [features]
0 Scenarios
0 Steps
0m0.000s
```

- Each Cucumber test is called a scenario
- A scenario contains steps -> tells Cucumber what to do
- This output means that Cucumber is scanning the features directory -> it didn’t find any scenarios to run

Create a scenario

```
nano checkout.feature
```

- Insert into this feature file:

```
Feature: Checkout
  Scenario: Checkout a banana
    Given the price of a "banana" is 40c
    When I checkout 1 "banana"
    Then the total price should be 40c
```

Gherkin

- Cucumber expects some structure:
- keywords Feature, Scenario, Given, When, and Then offer structure
- everything else is documentation
- the feature file is just a plain-text file
- This structure is called Gherkin

```
$ ./cucumber
Feature: Checkout

  Scenario: Checkout a banana            # checkout.feature:2
    Given the price of a "banana" is 40c
    When I checkout 1 "banana"
    Then the total price should be 40c

1 Scenarios (1 undefined)
3 Steps (3 undefined)
0m0.000s

You can implement missing steps with the snippets below:

@Given("^the price of a \"(.*?)\" is (\\d+)c$")
public void the_price_of_a_is_c(String arg1, int arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I checkout (\\d+) \"(.*?)\"$")
public void i_checkout(int arg1, String arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^the total price should be (\\d+)c$")
public void the_total_price_should_be_c(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Look at output

```
Feature: Checkout #-> Cucumber has found a feature
Scenario: Checkout a banana # checkout.feature:2
Given the price of a "banana" is 40c
When I checkout 1 "banana"
Then the total price should be 40c
1 Scenarios (1 undefined) # -> Cucumber found scenario but could
3 Steps (3 undefined) #-> should be mapped on methods
0m0.000s
You can implement missing steps with the snippets below:
#-> Cucumber offers suggestions to mep steps to methods

@Given("^the price of a \"(.*?)\" is (\\d+)c$")
public void the_price_of_a_is_c(String arg1, int arg2) throws Thr
// Write code here that turns the phrase above into concrete acti
throw new PendingException();
}
```

Change name generation

```
@Given("^the price of a \"(.*?)\" is (\\d+)c$")
public void the_price_of_a_is_c(String arg1, int arg2) {
  // Write code here that turns the phrase above into concrete ac
  throw new PendingException();
}
```

- _ underscores in method names are not preferred
- add --snippets camelcase to cucumber.sh

```
java -cp "jars/*" cucumber.api.cli.Main -p pretty --snippets came
```

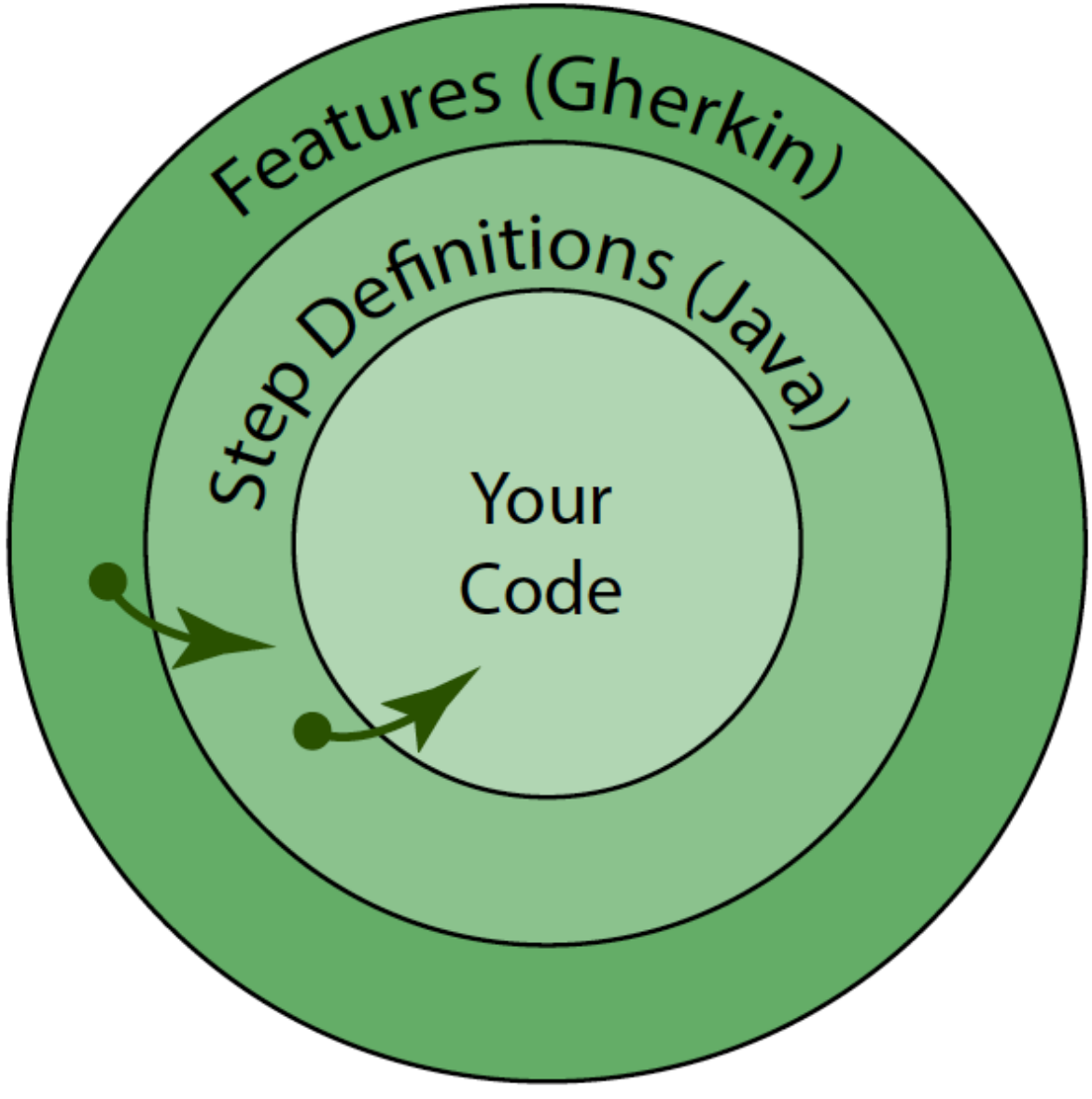


Figure 1—The main layers of a Cucumber test suite

Step definitions

```
$ mkdir step_definitions
```

- create inside this directory the CheckoutSteps.java file

```
package step_definitions;
import cucumber.api.java.en.*;
import cucumber.api.PendingException;
public class CheckoutSteps {
  // paste the snippets from Cucumber's
  //last output into a new Java file
}
```

Running the steps

- The java code inside CheckoutSteps.java must be compiled:
- Change the cucumber.sh script to include javac

```
javac -cp "jars/*" step_definitions/CheckoutSteps.java
java -cp "jars/*:." cucumber.api.cli.Main -p pretty --snippets ca
```

- Remark the : (colon) inside the -cp argument "jars:." -> the current direcotry is also considered (windows ;)
- -g step_definitions tells Cucumber where to look for the step definitions

Running ./cucumber.sh again

Looking at the output

```
1 Scenarios (1 pending)
3 Steps (2 skipped, 1 pending)
0m0.138s
```

- The scenario has graduated from undefined to pending
- -> it means Cucumber is now running the first step
- The PendingException tells Cucumber that this scenario is still a work in progress
- As soon as Cucumber encounters a failed or pending step, it will stop the scenario and skip the remaining steps

The First Step Definition

- Implement the step definition for Given the price of a banana is 40c
- This step must remember the price of bananas

```
@Given("^the price of a \"(.*?)\" is (\\d+)c$")
public void thePriceOfAIsC(String name, int price) throws Throwab
    int bananaPrice = price;
}
```

- the only items we care for now are bananas so forget about the name

Running ./cucumber.sh again

The verbose feedback

- The feedback of cucumber is often too detailed
- Tailor the feedback into more succint

```
java -cp "jars/*:." cucumber.api.cli.Main -p progress --snippets

# changed -p pretty -> -p progress
```

Output compressed

```
.P-

1 Scenarios (1 pending)
3 Steps (1 skipped, 1 pending, 1 passed)
0m0.081s

cucumber.api.PendingException: TODO: implement me
        at step_definitions.CheckoutSteps.iCheckout(CheckoutSteps.java:17)
        at *.When I checkout 1 "banana"(checkout.feature:4)
```

- Where:
  1. . (dot) -> step passed
  2. P -> step is pending
  3. - -> step is skipped

Implementing second step

```
@When("^I checkout (\\d+) \"(.*?)\"$")
public void iCheckout(int itemCount, String itemName) throws Thro
  Checkout checkout = new Checkout();
  checkout.add(itemCount, bananaPrice);
}
```

- Running ./cucumber.sh results in a compile error -> the Checkout class does not yet exist

Creating Checkout.java

```
$ mkdir implementation
# the implementation folder holds our Checkout class

package implementation;
public class Checkout {
    public void add(int count, int price) {

    }
}

# change cucumber.ps
javac -cp "jars/*:." step_definitions/CheckoutSteps.java implemen
```

Running again

```
step_definitions/CheckoutSteps.java:19: error: cannot find symbol
        checkout.add(itemCount, bananaPrice);
                                ^
  symbol:   variable bananaPrice
  location: class CheckoutSteps
1 error
```

- The variable bananaPrice is local to thePriceOfAIsC() -> but we're trying to use it in iCheckout()
- Move it to be an instance variable so that it can be shared by all step definitions

Adding Assertion

```
@Then("^the total price should be (\\d+)c$")
public void theTotalPriceShouldBeC(int total) throws Throwable {
    assertEquals(total, checkout.total());
}
```

- Add the junit jar to the jars folder
- Add static import
- Create the total() method

```
public int total() {
  return 0;
}
```

Running again

```
step_definitions/CheckoutSteps.java:31: error: cannot find symbol
        assertEquals(total, checkout.total());
                            ^
  symbol:   variable checkout
  location: class CheckoutSteps
1 error
```

- The local variable checkout in iCheckout() has gone out of scope
- Make checkout an instance variable of CheckoutSteps

```
public class CheckoutSteps {
  Checkout checkout;
  @When("^I checkout (\\d+) \"(.*?)\"$")
  public void iCheckout(int itemCount, String itemName) {
     checkout = new Checkout();
     checkout.add(itemCount, bananaPrice);
  }
}
```

Finally a failing test

```
..F

1 Scenarios (1 failed)
3 Steps (1 failed, 2 passed)
0m0.099s

java.lang.AssertionError: expected:<40> but was:<0>
        at org.junit.Assert.fail(Assert.java:92)
        at org.junit.Assert.failNotEquals(Assert.java:646)
        at org.junit.Assert.assertEquals(Assert.java:127)
        at org.junit.Assert.assertEquals(Assert.java:471)
        at org.junit.Assert.assertEquals(Assert.java:455)
        at step_definitions.CheckoutSteps.theTotalPriceShouldBeC
        at *.Then the total price should be 40c(checkout.feature:5)
```

- It fails for the right reason
  1. The asserting uses Checkout to examine the total
  2. It tells what the total should look like

Joe asks:

I Feel Weird: You're Making Tests Pass but Nothing Works!

We've implemented a step that uses the `Checkout` class and passes, even though the "class" just contains method implementations that don't do anything useful. What's going on here?

Remember that a step isn't a test in itself. The test is the whole scenario, and that isn't going to pass until all of its steps do. By the time we've implemented all of the step definitions, there's going to be only one way to make the whole scenario pass, and that's to build a checkout that can total items!

When we work outside-in like this, we often use temporary *stubs* like the empty `Checkout` class as placeholders for details we need to fill in later. We know that we can't get away with leaving that as an empty file forever, because eventually Cucumber is going to tell us to come back and make it do something useful in order to get the whole scenario to pass.

This principle, *deliberately doing the minimum useful work the tests will let us get away with*, might seem lazy, but in fact it's a discipline. It ensures that we make our tests thorough: if the test doesn't drive us to write the right thing, then we need a better test.

Make it pass

```
package implementation;
public class Checkout {
    public void add(int count, int price) {

    }
    public int total() {
        return 40;
    }
}
```

```
...

1 Scenarios (1 passed)
3 Steps (3 passed)
0m0.116s
```

What's wrong with return 40;

- return 40; is the most simple solution, does it suffice?
- we want to do the minimum ***useful work*** that the tests will let us get away with
- No inputs are used
- No total is added up
- Is this ok?

Alistair Cockburn

- In Crystal Clear: A Human-Powered Methodology for Small Teams
  - is advocated to build a walking skeleton as early as possible to flush out any potential problems with your technology choices
- why not building something more useful that passes this scenario and helps to learn more about the planned implementation?

An other argument

- code duplication
  - a hard-coded value of 40 in two places
- In a more complex system, this kind of duplication might go unnoticed, and results in a brittle scenario

Kent Beck

- Use what Kent Beck calls triangulation (Test Driven Development: By Example)

```
Feature: Checkout
    Scenario Outline: Checkout bananas
        Given the price of a "banana" is 40c
        When I checkout <count> "banana"
        Then the total price should be <total>c

        Examples:
        | count  | total |
        | 1      | 40    |
        | 2      | 80    |
```

- Change the Scenario into a Scenario Outline -> specify multiple scenarios using a table

Running again

```
$ ./cucumber
.....F

2 Scenarios (1 failed, 1 passed)
6 Steps (1 failed, 5 passed)
0m0.155s

java.lang.AssertionError: expected:<80> but was:<40>
        at org.junit.Assert.fail(Assert.java:92)
        . . .
        at step_definitions.CheckoutSteps.theTotalPriceShouldBeC
        at *.Then the total price should be 80c(checkout.feature:6)
```

Simple -> shows direction solution

```
package implementation;
public class Checkout {

  int runningTotal=0;

  public void add(int count, int price) {
    runningTotal+=count*price;
  }

  public int total() {
    return runningTotal;
  }
}
```

What We Just Learned

1. Directory Structure
    - Cucumber needs to know where features and step definition are kept
2. Baby Steps
    - When working outside-in with Cucumber it helps to stay focused
    - Every time we make a change, any mistakes we make are found and resolved quickly
3. Gherkin
    - Cucumber tests are written in Gherkin
        1. Gherkin files are plain text
        2. have a .feature extension

Try this

```
Scenario: Two bananas scanned separately
    Given the price of a "banana" is 40c
    When I checkout 1 "banana"
    And I checkout 1 "banana"
    Then the total price should be 80c
```

An other scenario

```
Scenario: A banana and an apple
    Given the price of a "banana" is 40c
    And the price of a "apple" is 25c
    When I checkout 1 "banana"
    And I checkout 1 "apple"
    Then the total price should be 65c
```

Solution to scenario on preceding page

```
public class CheckoutSteps {
    Checkout checkout=new Checkout();
    Map<String,Integer> productPrices= new HashMap<>();

    @Given("^the price of a \"([^\"]*)\" is (\\d+)c$")
    public void thePriceOfAIsC(String itemName, int price) throws
        productPrices.put(itemName, price);
    }
    @When("^I checkout (\\d+) \"([^\"]*)\"$")
    public void iCheckout(int itemCount, String itemName) throws
        checkout.add(itemCount, productPrices.get(itemName));
    }
    @Then("^the total price should be (\\d+)c$")
    public void theTotalPriceShouldBeC(int total) throws Throwabl
        assertEquals(total, checkout.total());
    }
```

Are the regexp's similar?

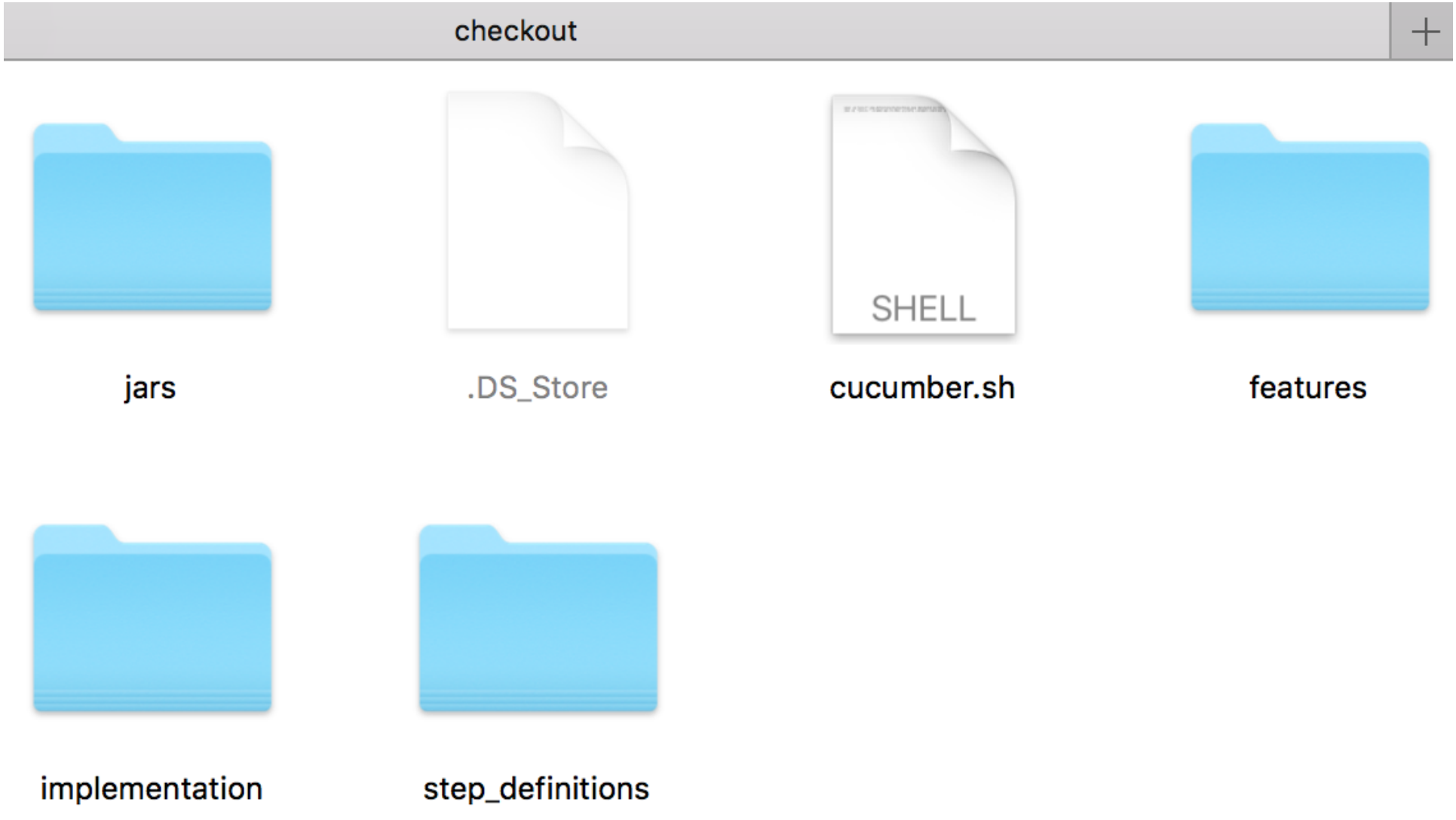
```
@When("^I checkout (\\d+) \"([^\"]*)\"$")
public void iCheckout(int itemCount, String itemName) throws Thro
    checkout.add(itemCount, productPrices.get(itemName));
}

@When("^I checkout (\\d+)  \"([^\"]*)\"$")
public void iCheckout(int itemCount, String itemName) throws Thro
    checkout.add(itemCount, productPrices.get(itemName));
}
```

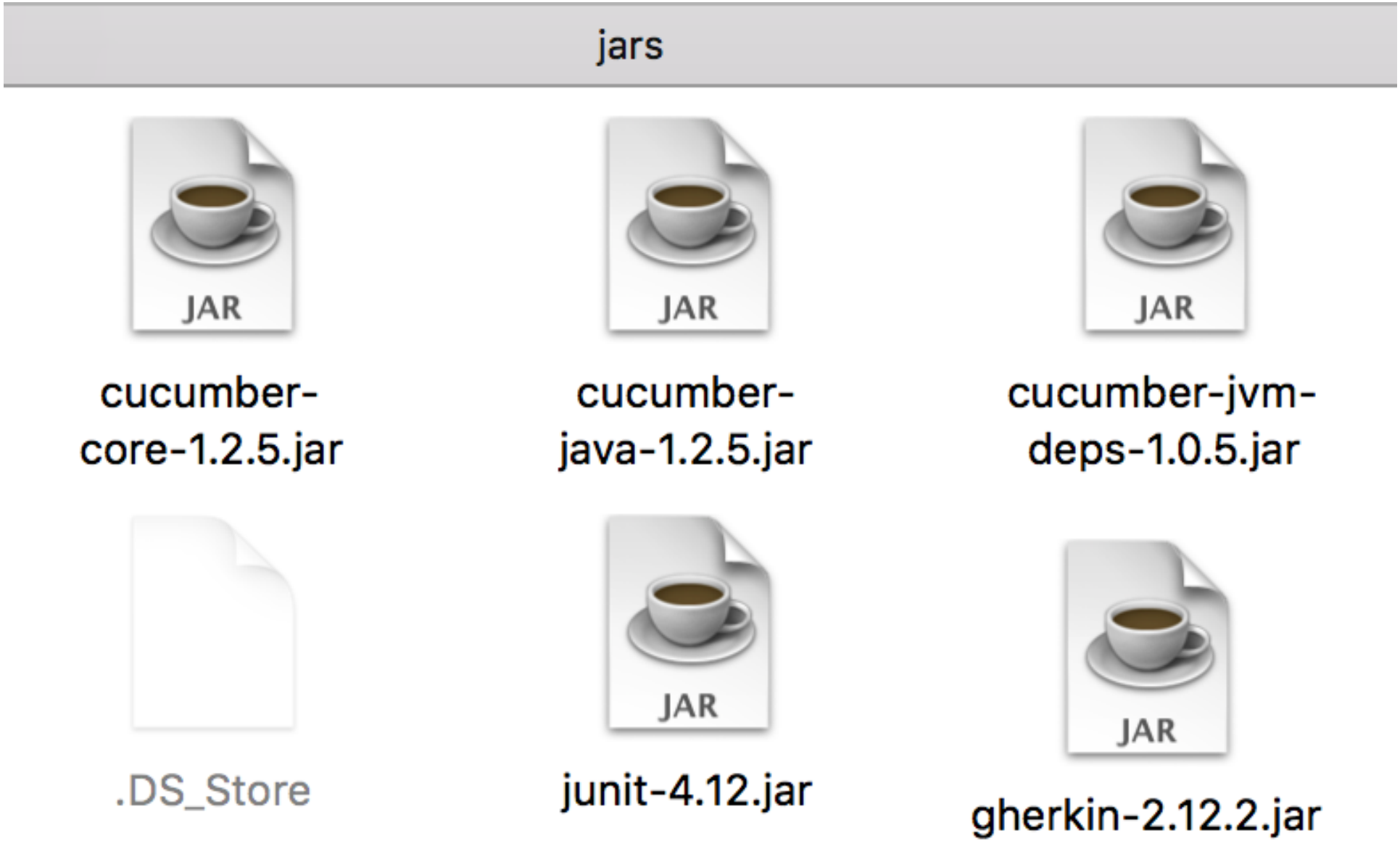
Are the regexp's similar? continued

- The gherkin files are mapped, more precisely the steps through the regexp's
- Regexp's are very sensitive to small sometimes difficult to see differences
- Place regexp's above each other to compare them:

```
@When("^I checkout (\\d+) \"([^\"]*)\"$")
@When("^I checkout (\\d+)  \"([^\"]*)\"$")
```

- Try to find the difference!


checkout
jars
.DS_Store
SHELL
cucumber.sh
features
implementation
step_definitions


jars
JAR
cucumber-core-1.2.5.jar
JAR
cucumber-java-1.2.5.jar
JAR
cucumber-jvm-deps-1.0.5.jar
.DS_Store
JAR
junit-4.12.jar
JAR
gherkin-2.12.2.jar