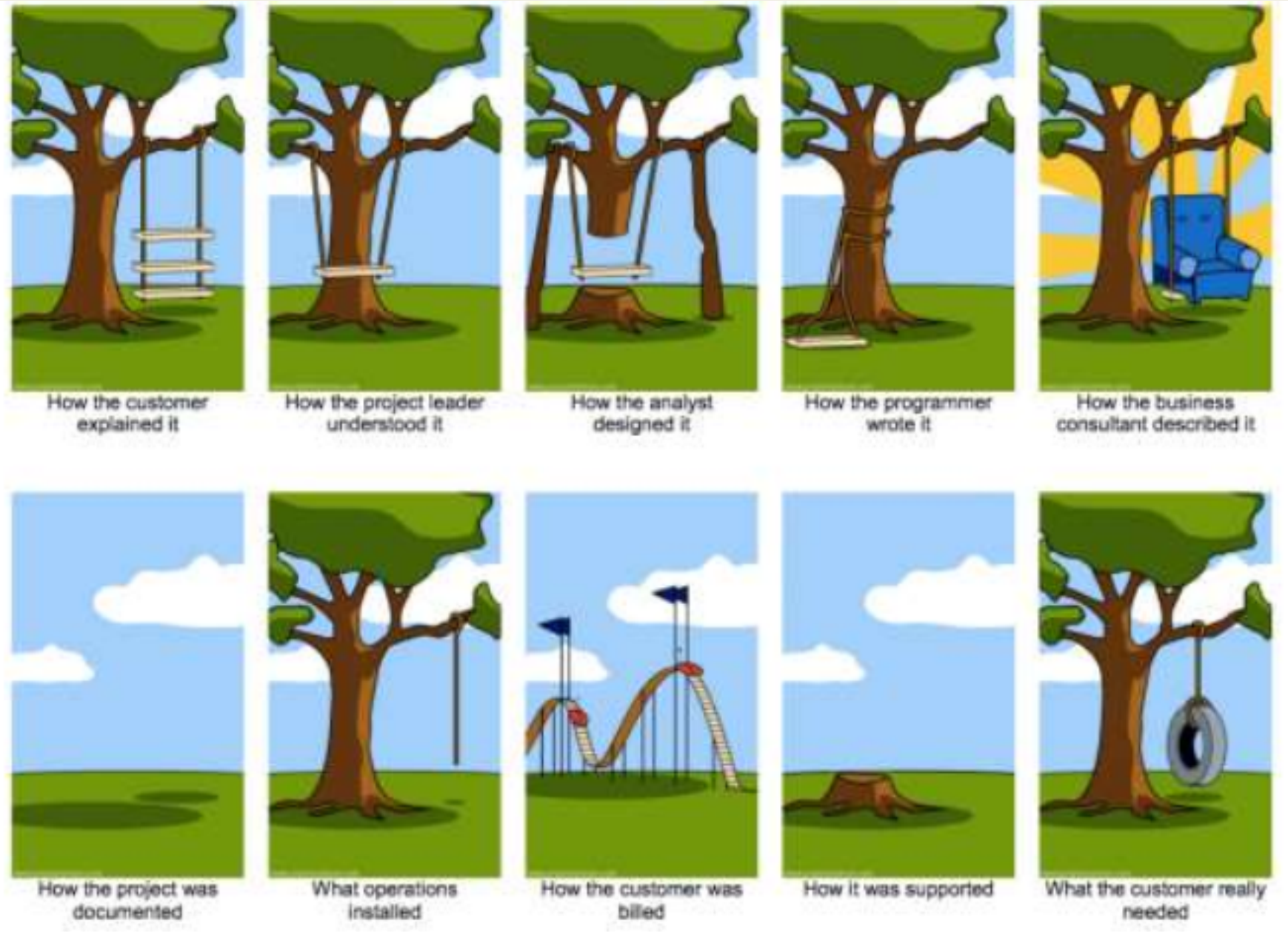
BDD USING CUCUMBER

Amit Kumar

Agenda

- ❑ Introduction to BDD
- ❑ Basics of BDD
- ❑ Understanding Acceptance Criterias and Scenarios
- ❑ Using Gherkin
- ❑ Creating Feature files
- ❑ Roles & Responsibilities in a BDD team
- ❑ Introduction to Cucumber
- ❑ Setting up Cucumber
- ❑ Cucumber features
- ❑ Writing Cucumber tests
- ❑ Running Cucumber tests
- ❑ Cucumber reports
- ❑ Integration with Selenium
- ❑ Building a Cucumber-Selenium framework

Behaviour-Driven Development – Why?



Behaviour-Driven Development - Overview

- BDD uses examples in conversations to illustrate behaviour
- In its core, BDD is simply the idea that software development should be governed by both technical proficiencies and business interests alike.
- The main tool of the method is simple domain-specific language (also known as DSL). Instead of complex lines of code, this language uses normal English words and logical constructs to express how the software should behave.
- BDD is a branch of the test-driven development method, which uses domain-specific language to convert natural language phrases and statements into executable tests.
- BDD focuses on implementing the minimum marketable feature that will yield the most value
- **“Behaviour” is a more useful word than “test”**

Behaviour Specifications

- Written in everyday business language, they comprehensively and unambiguously specify the required functionality to be delivered, as agreed by the team;
- They are the specifications that developers should refer to when they write code. The only code that should be written by a developer is code to satisfy the Behaviour Specifications;
- They also represent the complete set of test cases that are needed for the given functionality;
- They are the living, up-to-date, as-built documentation, for the entire team, about the functionality of each software component; and
- When implemented as automated tests, they are the executable documentation that becomes part of a suite of test assets.
- **No Other Test Cases Are Necessary!**
- **No Other Automated Tests!**
- You should not write any automated tests other than those that are specified by the Behaviour Specifications.

BDD - Benefits

- The team agrees, up-front, on the behaviours to be implemented and afterwards there is no room for a debate about the value of any of the tests;
- The team decides, up-front, which behaviours are implemented by the developers and which behaviours are performed by the testers;
- The team can more accurately provide estimates to management — since the tests are not dependent on the number of classes and methods coded;
- There typically are fewer tests and they are less brittle. Even though there are less tests, there still can be complete test coverage — because overall behaviour is being tested — not individual method functionality; and
- Test analysts (and everyone) have visibility of all the tests of each component.

Gherkin – the BDD language

- is the most widely used and recognized business-readable, domain-specific language (DSL) for specifying behaviours

Feature (logical grouping of behaviours)

---- *Scenario (a behavior)*

---- *Given (pre-requisite)*

---- *And*

---- *When (how to initiate the behaviour)*

---- *And*

---- *Then (how to verify the behaviour)*

---- *And*

---- *But (or)*

----- *Scenario*

---- *Given*

---- *When*

---- *Then*

Gherkin – the BDD language

Behaviour Description Rules:

- *Start with 'Should'*
- *Indicate Positive/Negative Behaviour*
- *Indicate Primary/Secondary Behaviour*
- *Unique*

[Primary/Secondary]/[Positive/Negative]-Should….

Gherkin – the BDD language

Behaviour Specification Steps Rules:

- *Avoid the word "I"*
- *Avoid technical terms*
- *Re-use Steps, If possible* - Behaviour Specification steps that refer to and perform the exact same underlying functionality should be implemented once and only once!
- *Single "When" as much as possible*
- *"Given" and "Then" steps with multiple "And" steps are fine*
- *Proper use of "Data Tables", "Examples", "Scenario Outline"*

Feature file

Feature: <short description>
As a <role>
I want <feature>
So that <business value>
Scenario: <Behaviour description>
Given…
When…
Then…

- Use the Agile User Story and Acceptance Criteria to form Feature files

Cucumber

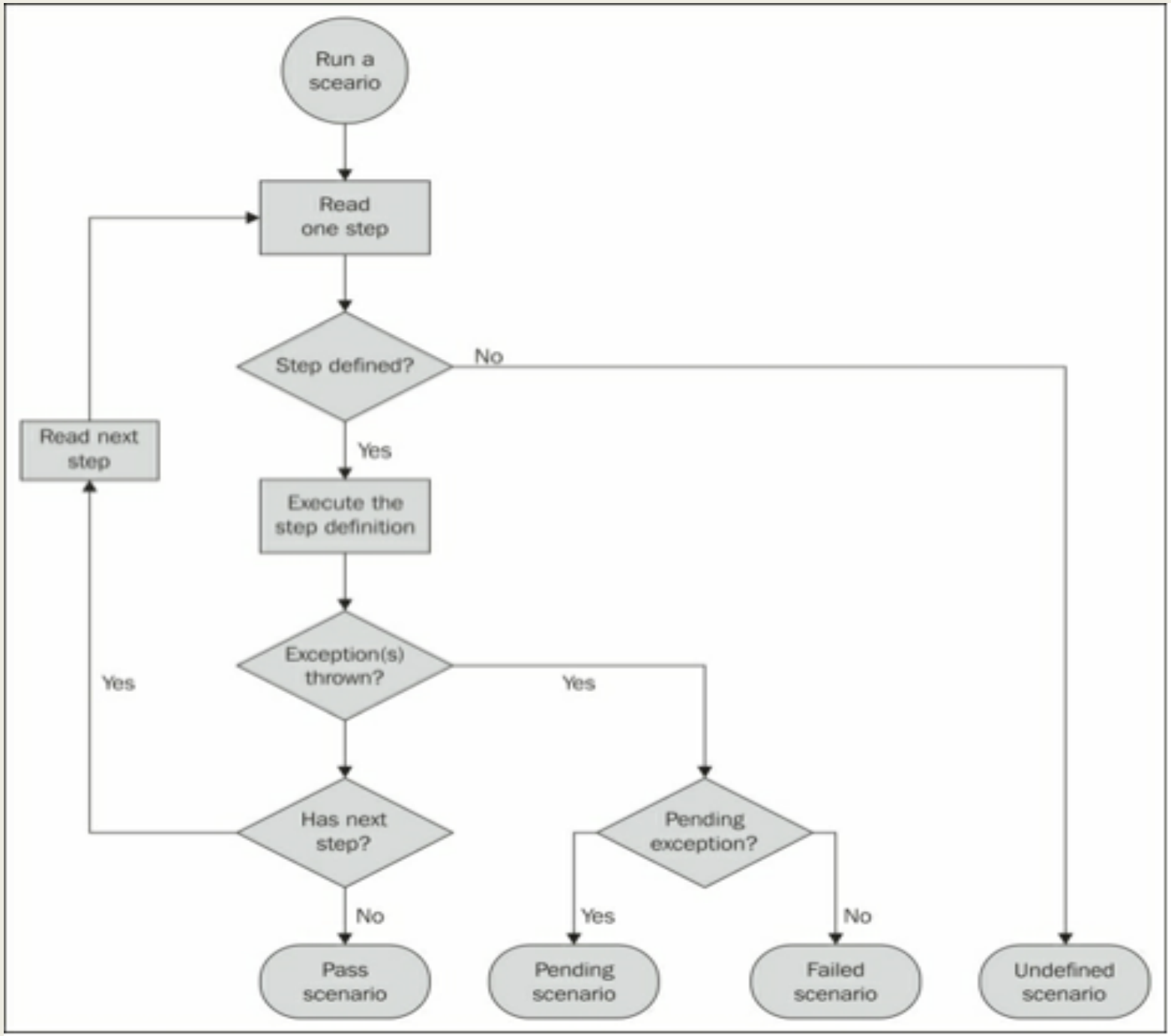
- is a popular BDD test automation tool.
- was initially implemented in Ruby and then extended to Java framework. Both the tools support native JUnit.
- **Cucumber-JVM** is the Java implementation of Cucumber
- Is a testing framework, driven by plain English text (Gherkin)
- How does it work?
  - *Cucumber reads the code written in plain English text (i.e. Gherkin) in the feature file.*
  - *finds the exact match of each step in the step definition*
  - *The piece of code to be executed can be different software frameworks like Selenium, Ruby on Rails etc.*

Pre-requisites & Setup

- Java 7+
- Maven 2/3
- Eclipse / IntelliJ Idea
- Cucumber Eclipse plug-in
- Natural Eclipse plug-in
- Selenium
- Maven configurations/dependencies:
  - *cucumber-java – v1.2.5*
  - *Cucumber-jvm (pom) –v1.2.5*
  - *cucumber-junit – v1.2.5*
  - *junit – v4.12*
  - *selenium-java – 2.5.3.1*

Cucumber Annotations

- Given
- When
- Then
- And
- But
- Scenario
- Scenario Outline – Parametrized Scenario
- Examples – used for Scenario Outline
- Background – Runs before each scenario but after @Before


Run a sceario
Read one step
Step defined?
No
Yes
Read next step
Execute the step definition
Exception(s) thrown?
Yes
Yes
Has next step?
Pending exception?
No
Yes
No
Pass scenario
Pending scenario
Failed scenario
Undefined scenario

Managing Scenarios

- Using @Tags you can let Cucumber know which scenarios to execute and which ones to ignore
- You can group your scenarios within and across feature files
- You can add multiple @Tags to a scenario (separated by a space)
- Feature files can be tagged too by tagging the ‘Feature:’ keyword. A feature tag is inherited by all the scenarios in the feature file
- Various tag expressions (in Runner):

– *@tag*

– *~@tag*

– *@tag1,@tag2*

Data Table

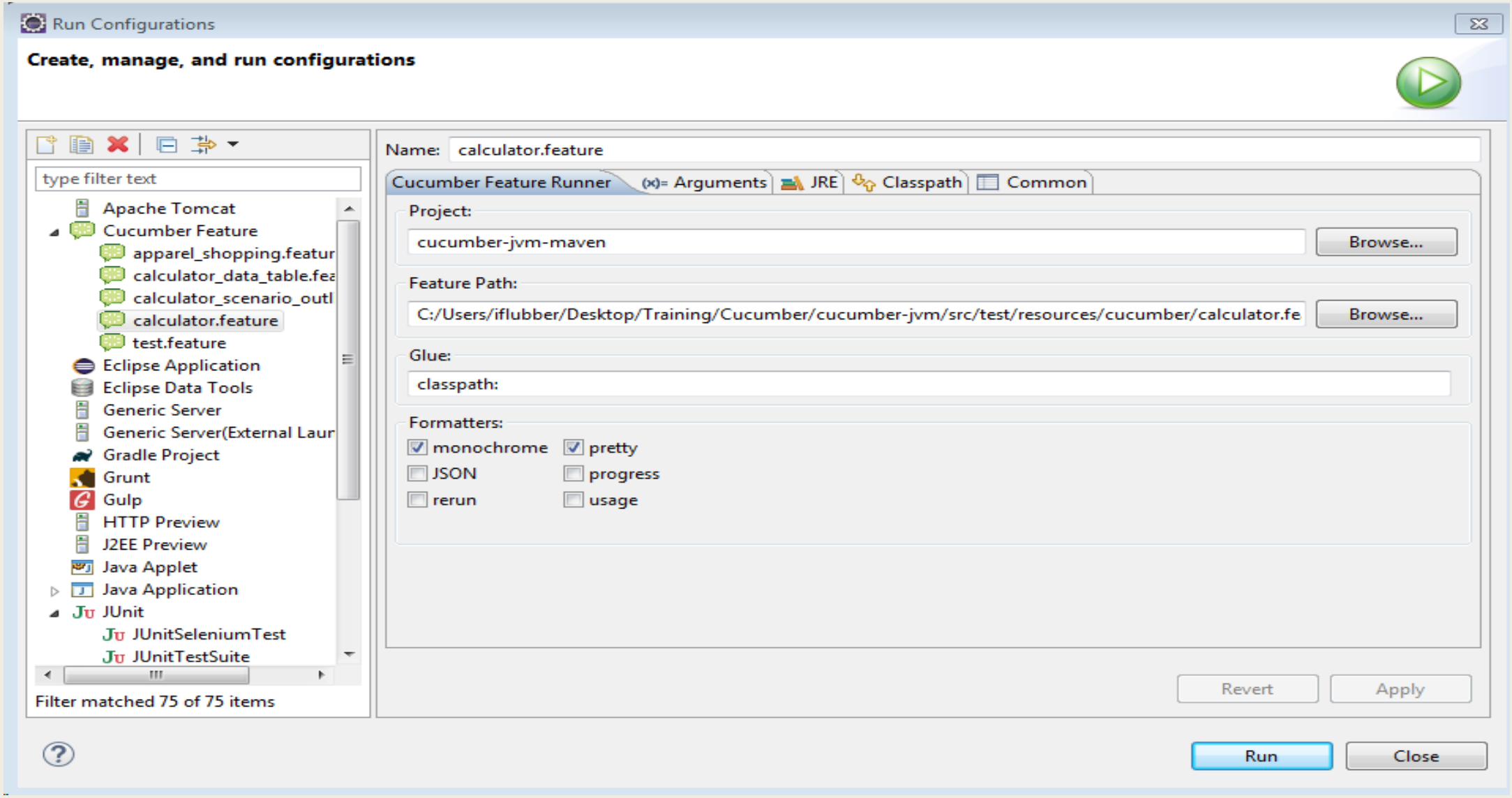
- Table without headers
  - *List<List<String>> data = table.raw(); table.get(row).get(col);*
- Table with headers
  - *List<Map<String,String>> data = table.asMaps(String.class,String.class); data.get(0).get("header");*
- Parsing DataTables as user-defined objects
  - *List<UserObject> object = table.get(0);*
  - *Define the UserObject class with fields having the same name as that of the table headers*
  - *Useful to pass a large set of data to steps, by creating domain object classes and use the same to do a data table transformation*

Cucumber Hooks

- @Before – any scenario applicable to all scenarios across all features
- @After – every scenario applicable to all scenarios across all features
- Tagged Hooks
  - *@Before("@tag") – before only the @tag scenario*
  - *@After("@tag1","~@tag2") – run only after the @tag1 scenario and don't run after the @tag2 scenario*
- Ordered Hooks
  - *@Before(order=0) – first before*
  - *@After(order=0) – last after*

Running Cucumber Feature File

Run the feature file directly from the IDE using the Cucumber plugin.



Using a Runner Class

This is a JUnit extension.

```
@RunWith(Cucumber.class)
@CucumberOptions (
        features =
        glue =
        ...
)
public class CucumberRunner {
}
```

Cucumber Runner Options

- dryRun – just checks if all Step definitions are present (default=false)
- features – sets the path of the feature files
- glue – sets the path of the definition files
- tags – specify the tags basis which the scenarios will be executed
- monochrome – display readable console output (default=false)
- format – set the report formatters
- strict – fail execution if there are undefined or pending steps (default=false)

Running Cucumber tests using Maven

- Create JUnit Test Runner(s) - *Test.java
- mvn test
- Check the cucumber reports

Cucumber in Jenkins

- Install Cucumber plugins
- Create & Configure a Maven job and setup Cucumber reports
- Run the job as mvn test
- Check the Cucumber reports

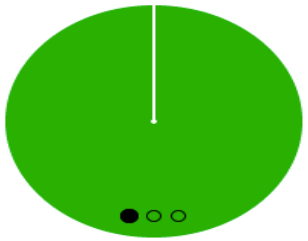
Cucumber Report | Jenkins | Previous results | Latest results | Features | Tags | Steps | Trends | Failures

Project	Number	Date
cucumber-test	17	06 Nov 2017, 21:49

Features Statistics

The following graphs show passing and failing statistics for features

Features



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Apparel Shopping	4	0	0	0	0	4	1	0	1	13s 808ms	Passed
Calculator	7	0	0	0	0	7	2	0	2	002ms	Passed
Calculator	3	0	0	0	0	3	1	0	1	016ms	Passed
Calculator	6	0	0	0	0	6	2	0	2	000ms	Passed

Selenium-Cucumber Framework