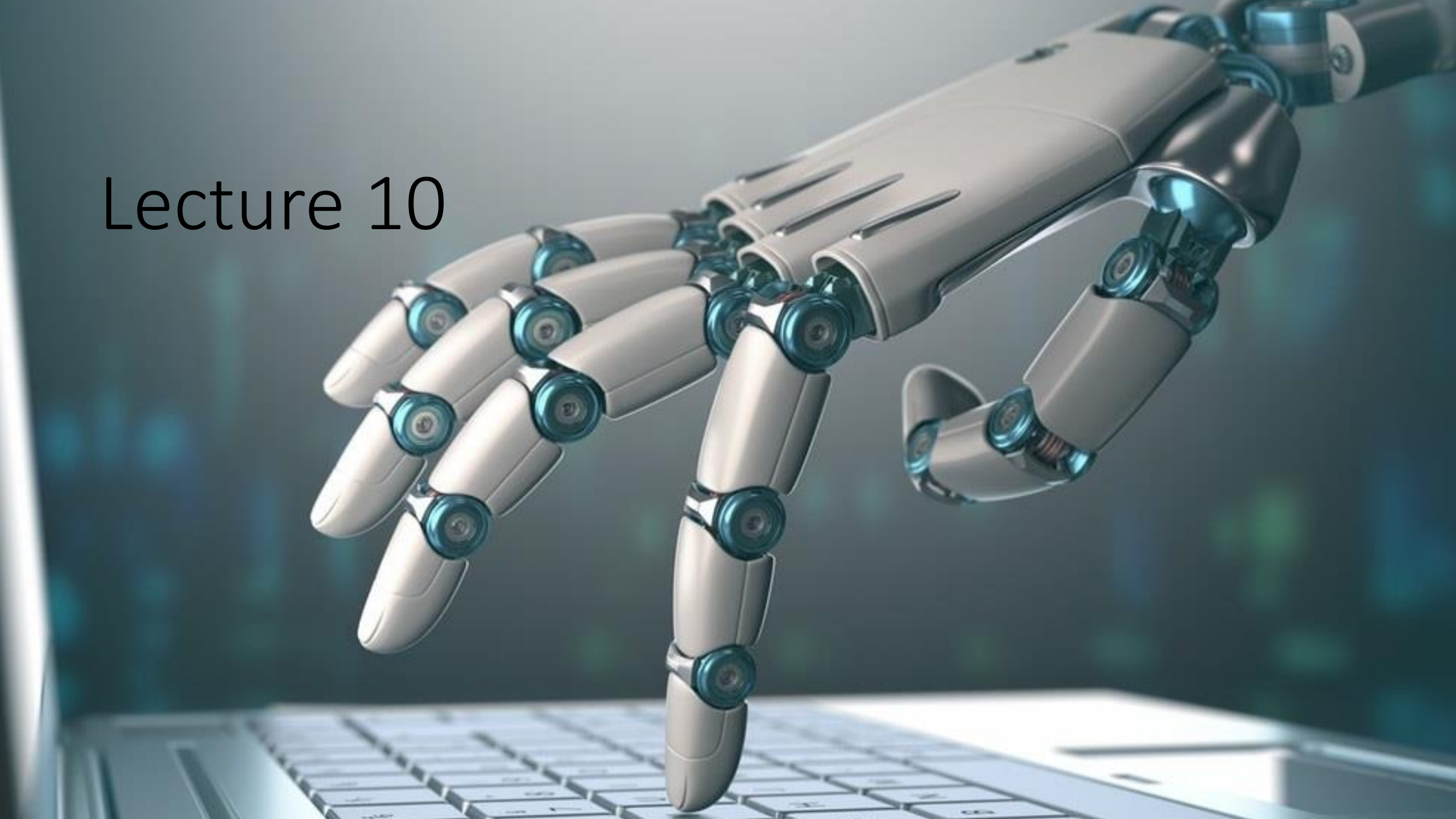
Lecture 10



Agenda

- **Cucumber**
- Gherkin
- Architecture
- How to configure
- Practice

What is Cucumber?

What is Cucumber?

- A **Feature File** is an entry point to the **Cucumber** tests. This is a **file** where you will describe your tests in Descriptive language (Like English). ... A **feature file** can contain a **scenario** or can contain many scenarios in a single **feature file** but it usually contains a list of scenarios.

What is Cucumber?

GoogleHomepage.feature

```
1  Feature: Google Homepage
2  This feature verifies the functionality on Google Homepage
3
4  Scenario: Check that main elements on Google Homepage are displayed
5  Given I launch Chrome browser
6  When I open Google Homepage
7  Then I verify that the page displays search text box
8  And the page displays Google Search button
9  And the page displays Im Feeling Lucky button
```

Agenda

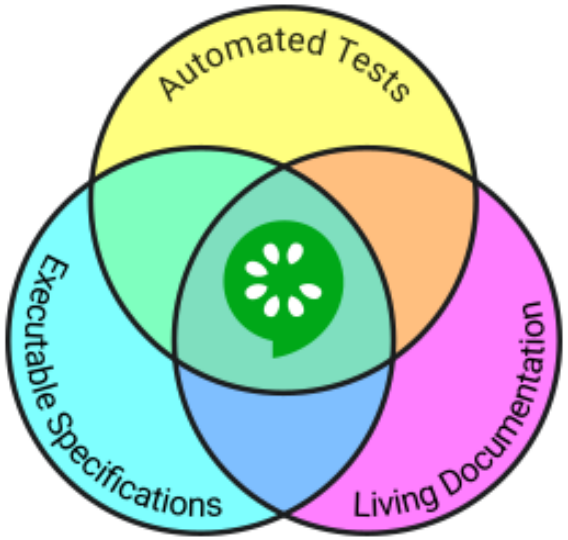
- Cucumber
- **Gherkin**
- Architecture
- How to configure
- Practice

Gherkin

Gherkin is a simple set of grammar rules that makes plain text structured enough for Cucumber to understand. The scenario above is written in Gherkin.

Gherkin serves multiple purposes:

- Unambiguous executable specification
- Automated testing using Cucumber
- Document how the system *actually* behaves



Gherkin commonly used phrases

- Feature
- Scenario
- Given, When, Then, And, But (Steps)
- Background
- Scenario Outline
- Examples

Feature file

Features file contain high level description of the Test Scenario in simple language. It is known as Gherkin. Gherkin is a plain English text language

Feature File consist of following components -

- **Feature**: A feature would describe the current test script which has to be executed.
- **Scenario**: Scenario describes the steps and expected outcome for a particular test case.
- **Scenario Outline**: Same scenario can be executed for multiple sets of data using scenario outline. The data is provided by a tabular structure separated by (| |).
- **Given**: It specifies the context of the text to be executed. By using datatables "Given", step can also be parameterized.
- **When**: "When" specifies the test action that has to performed
- **Then**: The expected outcome of the test can be represented by "Then"

Scenario

GoogleHomepage.feature

```
1  Feature: Google Homepage
2  This feature verifies the functionality on Google Homepage
3
4  Scenario: Check that main elements on Google Homepage are displayed
5  Given I launch Chrome browser
6  When I open Google Homepage
7  Then I verify that the page displays search text box
8  And the page displays Google Search button
9  And the page displays Im Feeling Lucky button
```

Agenda

- Cucumber
- Gherkin
- **Architecture**
- How to configure
- Practice

Architecture

- Feature file
- Steps definition
- Test Runner

Feature file

GoogleHomepage.feature

```
1  Feature: Google Homepage
2  This feature verifies the functionality on Google Homepage
3
4  Scenario: Check that main elements on Google Homepage are displayed
5  Given I launch Chrome browser
6  When I open Google Homepage
7  Then I verify that the page displays search text box
8  And the page displays Google Search button
9  And the page displays Im Feeling Lucky button
```

Feature file contains

- Steps
- Examples
- Background

Step definition

```
@Given("^User is on Home Page$")
public void user_is_on_Home_Page() throws Throwable {
driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://www.store.demoqa.com");
}
```

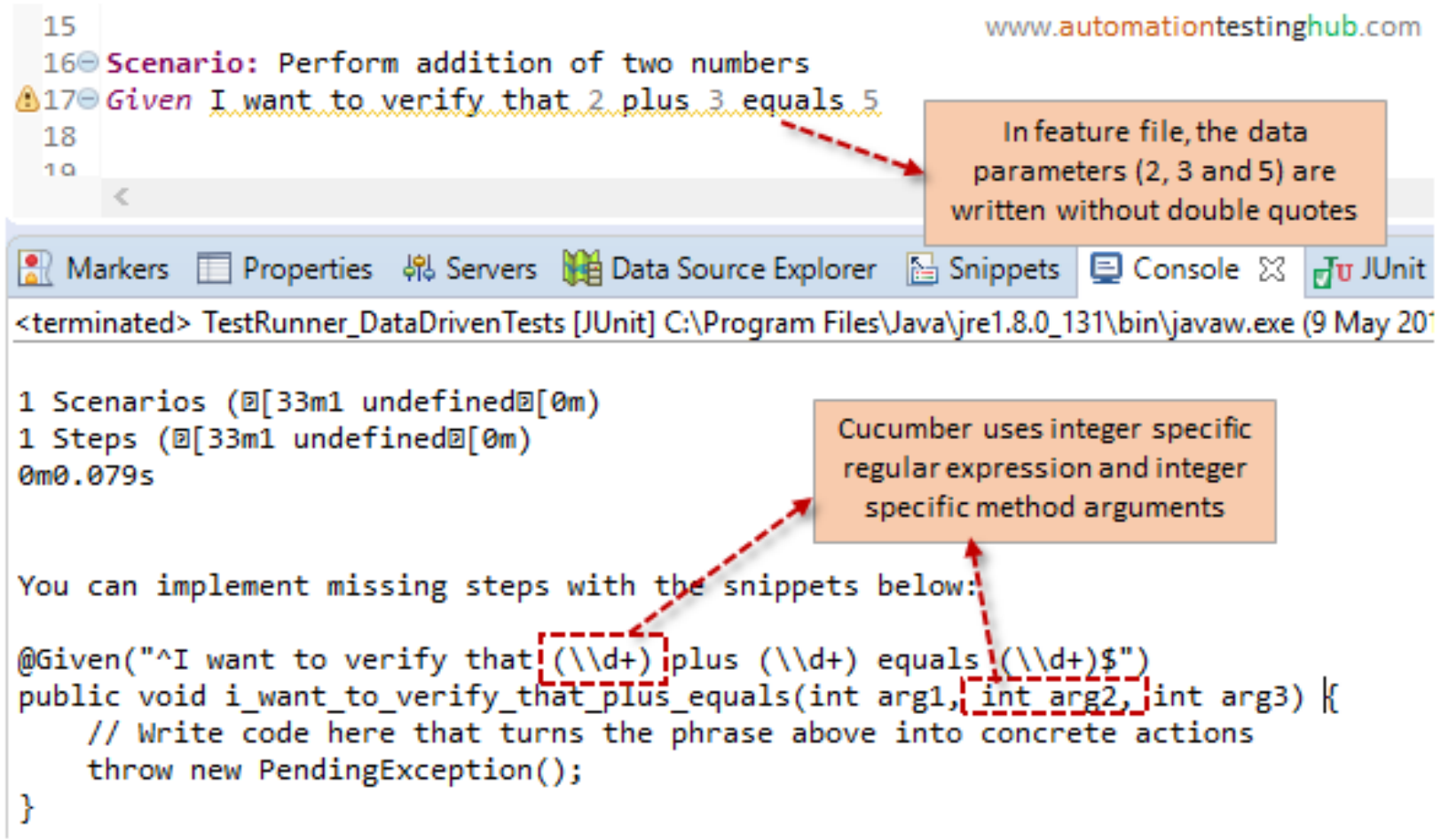
Parameterized steps - String

```
Scenario: Login to application
Given I open my application
And I login with credentials "admin" and "pass1234"
```

```
@Given("^I login with credentials \"([^\"]*)\" and \"([^\"]*)\"$")
public void i_login_with_credentials(String username, String password) {
    //Add code to enter username, password and click on Login button
    //Add code to verify that login is successful
}
```

```
@Given("^I login with credentials (.*) and (.*)$")
public void i_login_with(String username, String password) {
    //Add code to login to application
}
```

Parameterized steps - INT



Parameterized steps – choose from the options

```
1 Scenario: Application Login
2 When I open application in Chrome browser
```

```
1  @When("^I open application in (Chrome|Firefox|Safari) browser$")
2  public void i_open_application_in_browser(String browser) {
3      if(browser.equalsIgnoreCase("chrome")) {
4          //code to launch Chrome
5      } else if(browser.equalsIgnoreCase("firefox")) {
6          //code to launch Firefox
7      } else if(browser.equalsIgnoreCase("safari")) {
8          //code to launch Safari
9      }
10 }
```

Background

Don't repeat yourself

In some features, there might be one and the same Given steps before each scenario. In order to avoid copy/paste, it is better to define those steps as feature prerequisite with **Background** keyword.

```
1  Feature: search Wikipedia
2
3    Background:
4      Given Open http://en.wikipedia.org
5      And Do login
6
7    Scenario: direct search article
8      Given Enter search term 'Cucumber'
9      When Do search
10     Then Single result is shown for 'Cucumber'
```

Background

Tips for using Background

- Don't use `Background` to set up **complicated states**, unless that state is actually something the client needs to know.
  - For example, if the user and site names don't matter to the client, use a higher-level step such as `Given I am logged in as a site owner`.
- Keep your `Background` section **short**.
  - The client needs to actually remember this stuff when reading the scenarios. If the `Background` is more than 4 lines long, consider moving some of the irrelevant details into higher-level steps.
- Make your `Background` section **vivid**.
  - Use colourful names, and try to tell a story. The human brain keeps track of stories much better than it keeps track of names like `"User A"`, `"User B"`, `"Site 1"`, and so on.
- Keep your scenarios **short**, and don't have too many.
  - If the `Background` section has scrolled off the screen, the reader no longer has a full overview of whats happening. Think about using higher-level steps, or splitting the `*.feature` file.

Scenario Outline

The `Scenario Outline` keyword can be used to run the same `Scenario` multiple times, with different combinations of values.

The keyword `Scenario Template` is a synonym of the keyword `Scenario Outline`.

Copying and pasting scenarios to use different values quickly becomes tedious and repetitive:

```
Scenario: eat 5 out of 12
  Given there are 12 cucumbers
  When I eat 5 cucumbers
  Then I should have 7 cucumbers

Scenario: eat 5 out of 20
  Given there are 20 cucumbers
  When I eat 5 cucumbers
  Then I should have 15 cucumbers
```

Scenario Outline

We can collapse these two similar scenarios into a `Scenario Outline`.

Scenario outlines allow us to more concisely express these scenarios through the use of a template with `< >`-delimited parameters:

```
Scenario Outline: eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers

  Examples:
    | start | eat | left |
    |    12 |   5 |    7 |
    |    20 |   5 |   15 |
```

A `Scenario Outline` must contain an `Examples` (or `Scenarios`) section. Its steps are interpreted as a template which is never directly run. Instead, the `Scenario Outline` is run *once for each row* in the `Examples` section beneath it (not counting the first header row).

The steps can use `<>` delimited *parameters* that reference headers in the examples table. Cucumber will replace these parameters with values from the table *before* it tries to match the step against a step definition.

Step definition

```
1  package steps;
2
3  import cucumber.api.java.en.Given;
4  import cucumber.api.java.en.Then;
5  import cucumber.api.java.en.When;
6  import org.openqa.selenium.By;
7  import org.openqa.selenium.chrome.ChromeDriver;
8
9  public class stepsDefinition {
10
11     private ChromeDriver driver = null;
12
13     @Given("^I have open the browser$")
14     public void openBrowser() {
15         String driverpath = "C:\\Users\\Tarkon\\IdeaProjects\\ForTutorial\\src\\main\\resources\\";
16         System.setProperty("webdriver.chrome.driver", driverpath + "chromedriver.exe");
17         driver = new ChromeDriver();
18     }
19
20     @When("^I open Facebook website$")
21     public void goToFacebook() { driver.navigate().to( url: "https://www.facebook.com/"); }
24
25     @Then("^Login button should exits$")
26     public void loginButton() {
27         if (driver.findElement(By.id("u_0_v")).isEnabled()) {
28             System.out.println("Test 1 Pass");
29         } else {
30             System.out.println("Test 1 Fail");
31         }
32         driver.close();
33     }
```

Hooks

What are Hooks in Cucumber?

Cucumber supports ***hooks***, which are blocks of code that run ***before*** or ***after*** each scenario. You can define them anywhere in your project or step definition layers, using the methods ***@Before*** and ***@After***. ***Cucumber Hooks*** allows us to better manage the code workflow and helps us to reduce the code redundancy. We can say that it is an unseen step, which allows us to perform our scenarios or tests.

Hooks

Why Cucumber Hooks?

In the world of testing, you must have encountered the situations where you need to perform the prerequisite steps before testing any test scenario. This prerequisite can be anything from:

- *Starting a webdriver*
- *Setting up DB connections*
- *Setting up test data*
- *Setting up browser cookies*
- *Navigating to certain page*
- *or anything before the test*

In the same way there are always after steps as well of the tests like:

- *Killing the webdriver*
- *Closing DB connections*
- *Clearing the test data*
- *Clearing browser cookies*
- *Logging out from the application*
- *Printing reports or logs*
- *Taking screenshots on error*
- *or anything after the test*

Hooks

```
1  package utilities;
2  import cucumber.api.java.After;
3  import cucumber.api.java.Before;
4
5  public class Hooks {
6
7   @Before
8      public void beforeScenario(){
9          System.out.println("This will run before the Scenario");
10     }
11
12  @After
13     public void afterScenario(){
14         System.out.println("This will run after the Scenario");
15     }
16 }
```

Test runner

```
1  package cucumber;
2
3  import cucumber.api.CucumberOptions;
4  import cucumber.api.junit.Cucumber;
5  import org.junit.runner.RunWith;
6
7  @RunWith(Cucumber.class)
8  @CucumberOptions(plugin = {"pretty"},
9          features={"src/test/resources/features"},
10         glue = {"steps"},
11         tags={"@SmokeTest"}
12         )
13 public class runTest { }
```

Test runner contains

You will see that our sample test runner class has only two annotations. These are **@RunWith** and **@CucumberOptions**

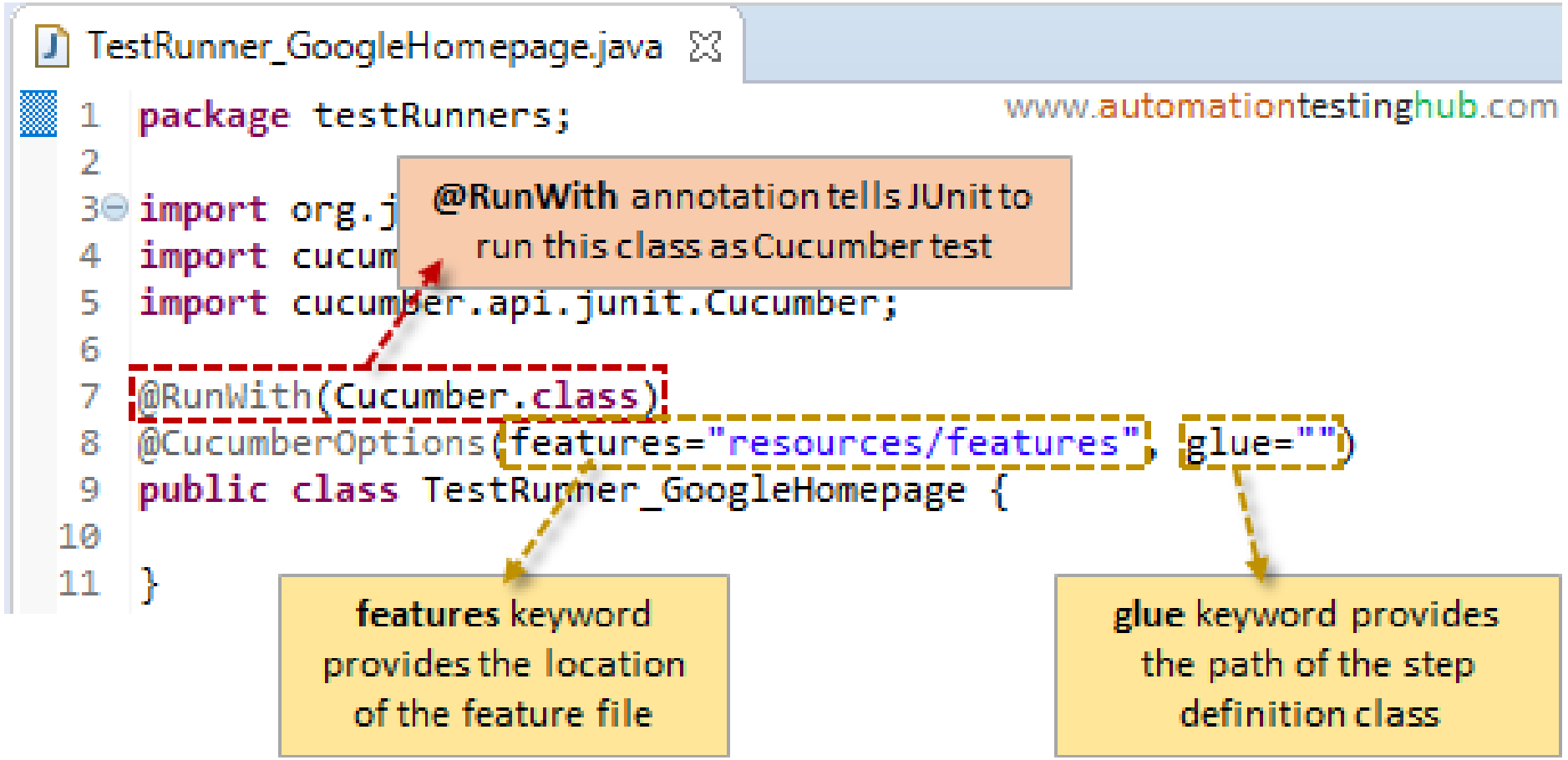
1. @RunWith annotation: This is a JUnit annotation that specifies which runner it has to use to execute this class. You can see that we have provided **Cucumber.class** as a parameter with this annotation. With this, JUnit will know that it has to execute this test case as a Cucumber test.

2. @CucumberOptions annotation: This annotation provides some important information which will be used to run your cucumber feature file. At the very least, Java should know the location of the feature file as well as the step definition class in your project. CucumberOptions annotation will do just that. As of now, we have provided two parameters in this annotation.

The first parameter, called **features**, provides the location of the feature file. Similarly, the second parameter, called **glue**, provides the path of the step definition class.

We have currently stored the feature file in **resources > features** folder in the project. So, we have added **resources/features** as the path for the feature file. Also, the glue parameter is empty, because we have not created any step definitions class till now.

Test runner contains



Test runner contains

Plugin: *plugin Option is used to specify different formatting options for the output reports. Various options that can be used as for-matters are:*

Note – *Format option is deprecated . Use Plugin in place of that.*

A- Pretty: Prints the *Gherkin* source with additional colors and stack traces for errors.

```
@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"}, strict = false)

public class RunYoursTest
{
 // This class will be empty
}
```

Agenda

- Cucumber
- Gherkin
- Architecture
- **How to configure**
- Practice

How to configure

1. Add cucumber libraries to project
2. Create feature file
3. Create step definitions
4. Create runner

1. Add cucumber libraries

- testCompile 'io.cucumber:cucumber-junit:4.4.0'
- testCompile 'io.cucumber:cucumber-jvm:4.4.0'
- testCompile 'io.cucumber:cucumber-java:4.4.0'

2. Create feature file

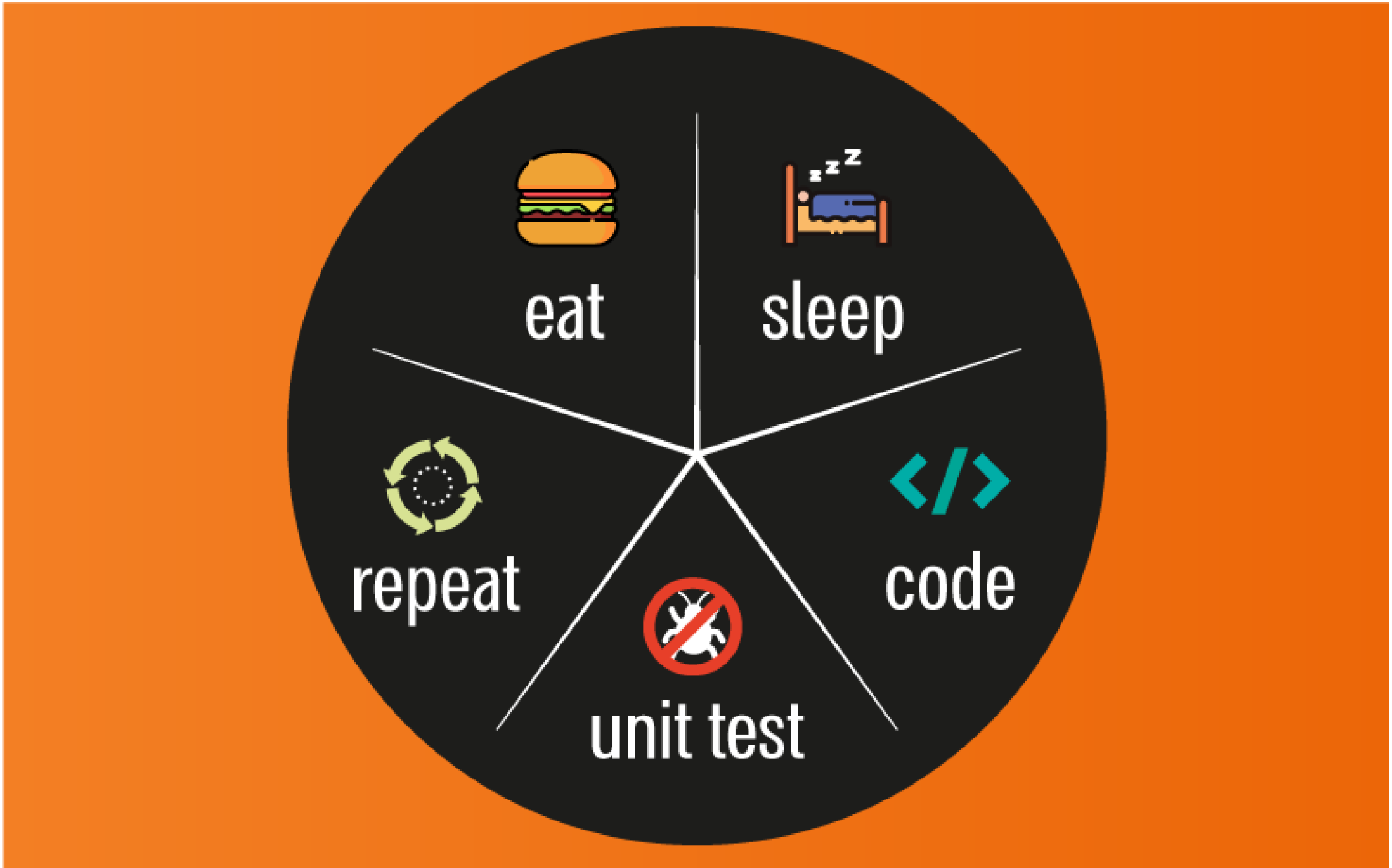
```
@SmokeTest
Feature: Cucumber run test

    I want to run a sample feature file.

Scenario: Cucumber setup
    Given I have open the browser
    When I open Facebook website
    Then Login button should exits
```

3. Create step definition

```java
package steps;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.openqa.selenium.By;
import org.openqa.selenium.chrome.ChromeDriver;

public class stepsDefinition {

    private ChromeDriver driver = null;

    @Given("^I have open the browser$")
    public void openBrowser() {
        String driverpath = "C:\\Users\\Tarkon\\IdeaProjects\\ForTutorial\\src\\main\\resources\\";
        System.setProperty("webdriver.chrome.driver", driverpath + "chromedriver.exe");
        driver = new ChromeDriver();
    }

    @When("^I open Facebook website$")
    public void goToFacebook() {
        driver.navigate().to("https://www.facebook.com/");
    }

    @Then("^Login button should exits$")
    public void loginButton() {
        if (driver.findElement(By.id("u_0_v")).isEnabled()) {
            System.out.println("Test 1 Pass");
        } else {
            System.out.println("Test 1 Fail");
        }
        driver.close();
    }
}
```

4. Create test runner

```
package cucumber;

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"},
        features={"src/test/resources/features"},
        glue = {"steps"},
        tags={"@SmokeTest"}
        )
public class runTest { }
```


eat
sleep
code
unit test
repeat

Agenda

- Cucumber
- Gherkin
- Architecture
- How to configure
- **Practice**

Task 1

- Create a separate gradle project
- Add cucumber libraries
- Create first feature file – usually create src/test/resources/features
- Create step definitions – java file – src/test/java/steps
- Create test runner – java file – src/test/java/runners
- Code sample is in the slides

Task 2

- Create feature, step def, test runner for ss.com
- Go to https://www.ss.com/lv/
- Open cars
- Enter price 6000 - 10000
- year from 2001
- engine max 3.0
- color – Balta
- Click submit
- Parameterize Steps

REMINDER - Background

Don't repeat yourself

In some features, there might be one and the same Given steps before each scenario. In order to avoid copy/paste, it is better to define those steps as feature prerequisite with **Background** keyword.

```
1  Feature: search Wikipedia
2
3    Background:
4      Given Open http://en.wikipedia.org
5      And Do login
6
7    Scenario: direct search article
8      Given Enter search term 'Cucumber'
9      When Do search
10     Then Single result is shown for 'Cucumber'
```

Task 2 – Part 2

- Using slide with Background – Move Go to https://www.ss.com/lv/ step to Background step

Task 3

- Create feature, step def, test runner for aliexpress
- Go to https://www.aliexpress.com/
- In search write: tattoo
- Press search
- Set min price 10
- Set max price 20
- Press ok
- Parameterize Steps

Homework – Group task - obligatory

- Create feature, step def, test runner for forumcinemas
- Go to https://www.forumcinemas.lv
- Login
- Go to profile page and check that name is AAAAAA
- And Surname BBBBBBBBB
- Go to movies and select random movie
- Parameterize Steps
- Now on above level of Jana roze build cucumber steps
- Add cucumber hooks Before and After

Reference

- Cucumber – tutorial Eng - https://www.tutorialspoint.com/cucumber/cucumber_java_testing.htm
- Cucumber – tutorial RU - https://habr.com/ru/post/332754/
- Cucumber - https://cucumber.io/
- Test runner - https://testingneeds.wordpress.com/2015/09/15/junit-runner-with-cucumberoptions/


THANK YOU FOR YOUR ATTENTION
YouTube