Testing in the BDD methodology

Paul Bodean

Summary

Recap – 1h

Theory – 1h

Practice – 2h

Install cucumber

Write your first scenario using Gherkin syntax

Write steps definition in Java

Run cucumber

Theory – 30 min

Basics of BDD

Practice – 2h

Let’s do some exercises

Scenarios definition

Steps implementation

Execution

Goals

- How to create a maven project
- Unit tests. What they are
- JSON understanding. Nice to have - Parser
- What is BDD and why we need it
- A basic end to end test execution with BDD
- Generate a report

Recap – Intro to programming

First program (a simple print to the console program)
Basic syntax (program, class, method)
Data types (int, bool, double, String)
Operator (+, -, ++, --, %, ==, *, /)
Decision making (**if**, switch)
Loop control (**for, while**, do while)

Recap – Intro to programming. Practice

Recap – Intro to programming. Practice

```java
// What value for register will be printed at the end of this block of Java code?
double register = 10.0;
register = register + 5; //Customer pays $5.
register = register - 2.5; //Customer receives $2.50 as change.
register = register + 10; //Customer pays $10.
register = register - 3; //Customer receives $3 as change.
System.out.println(register);

A. 19.0
B. 19.5
C. 22.5
D. 25.5
```

Recap – Intro to programming. Practice

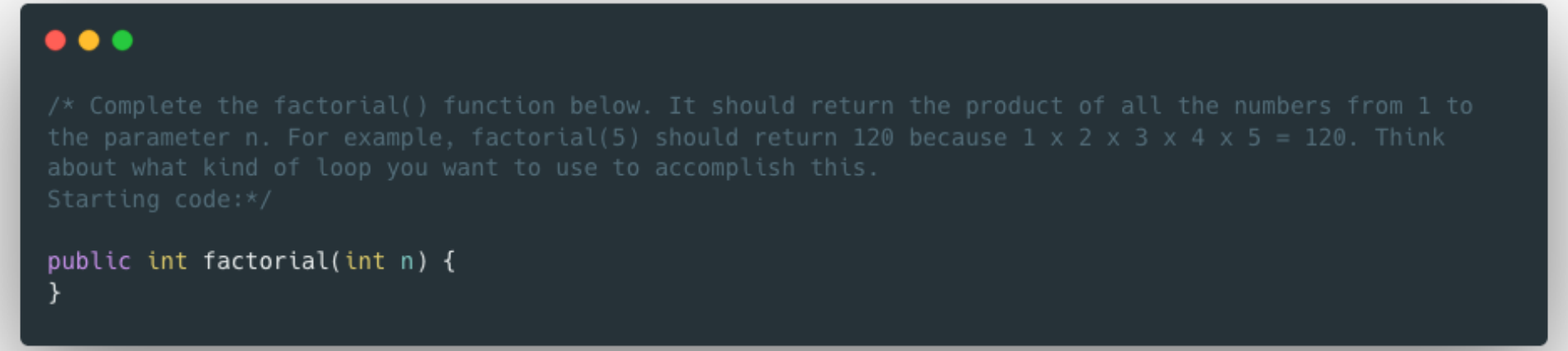
```java
// What will the following block of Java code print?

double balance = 0;
balance = balance + 20; //Add quarter 1 profits (thousands).
balance = balance - 25; //Subtract quarter 1 expenses (thousands).
balance = balance + 30; //Add quarter 2 profits (thousands).
balance = balance - 25; //Subtract quarter 1 expenses (thousands).

if (balance < 0) {
    System.out.println("We're in the red!");
} else if (balance > 0) {
    System.out.println("We made a profit!");
} else {
    System.out.println("We broke even.");
}

A. We're in the red!
B. We made a profit!
C. We broke even.
```

Recap – Intro to programming. Practice

```java
/* Find the error in this Java code. (Hint: think about scope!)
Imagine that the variables rewinding and playbackPosition could have different values depending on the
situation. */

double playbackPosition = 120;
boolean rewinding = true;
if (rewinding) {
    double rewindAmount = 0.1;
}
playbackPosition = playbackPosition - rewindAmount;
```

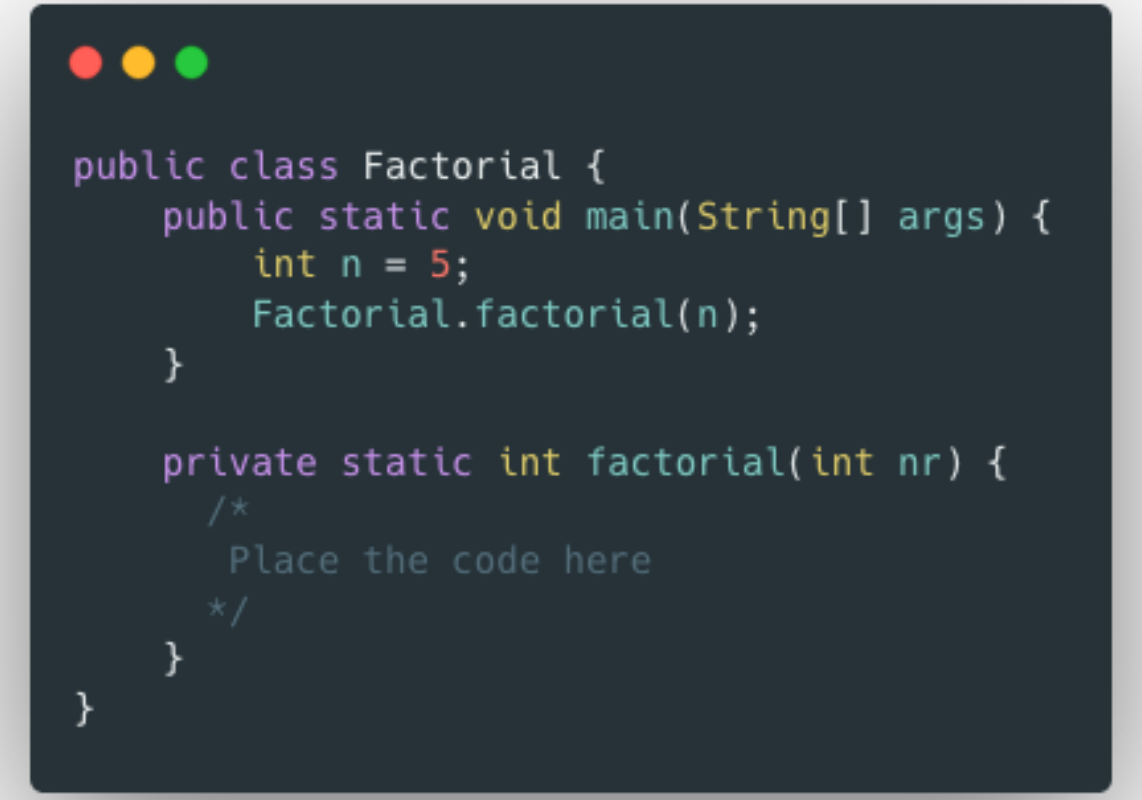
Recap – Intro to programming. Practice

Recap – Intro to programming. Practice



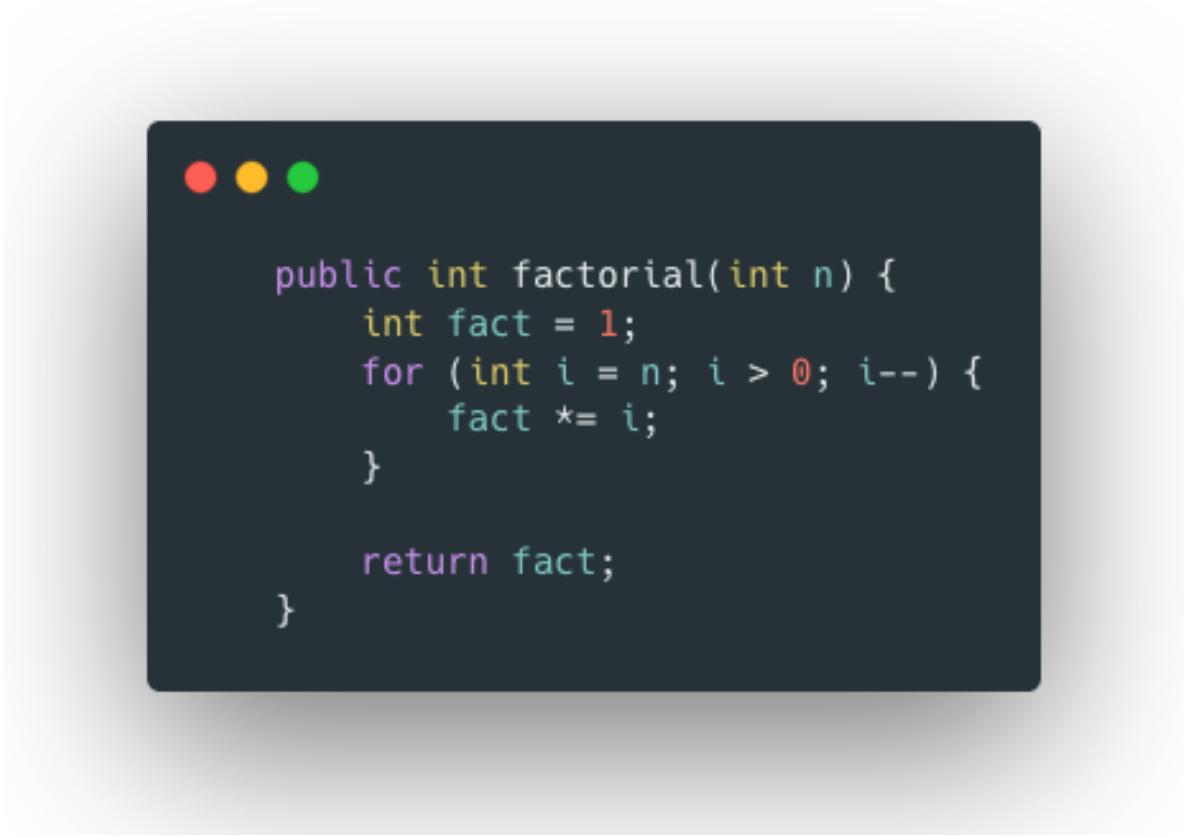
```
/* Complete the factorial() function below. It should return the product of all the numbers from 1 to
the parameter n. For example, factorial(5) should return 120 because 1 x 2 x 3 x 4 x 5 = 120. Think
about what kind of loop you want to use to accomplish this.
Starting code:*/

public int factorial(int n) {
}
```

Recap – Intro to programming. Practice



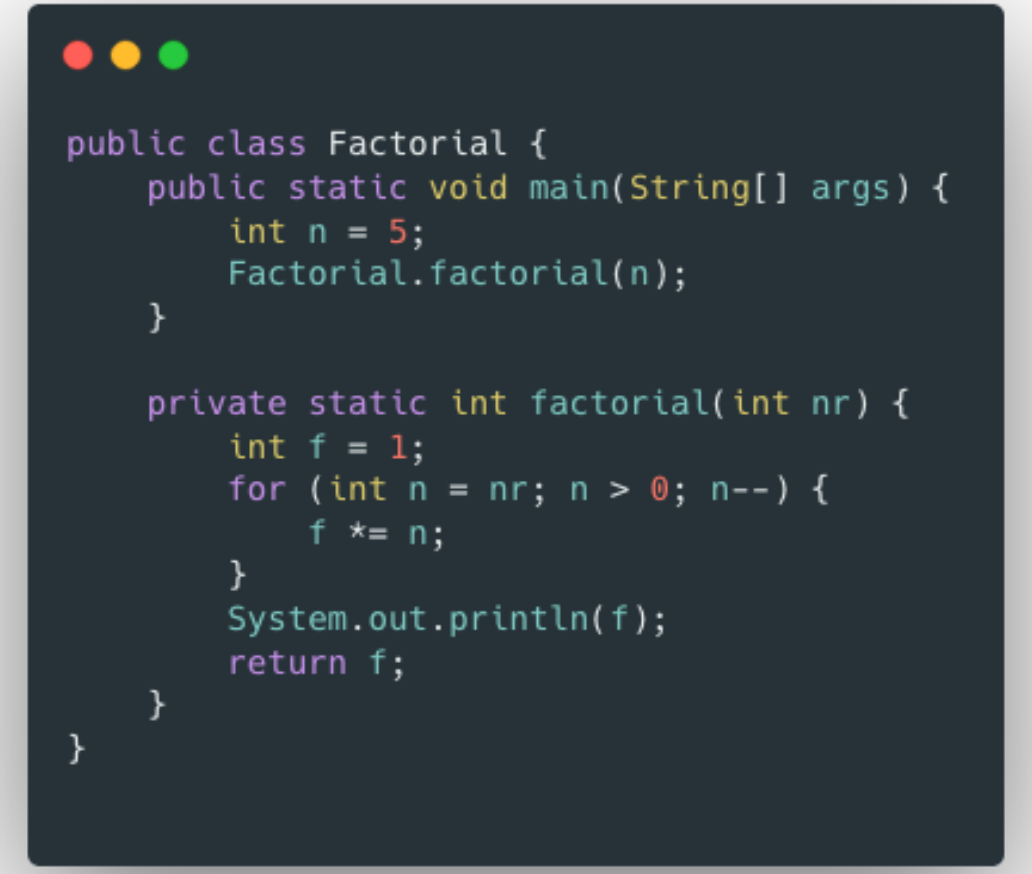
```java
public class Factorial {
    public static void main(String[] args) {
        int n = 5;
        Factorial.factorial(n);
    }

    private static int factorial(int nr) {
      /*
       Place the code here
      */
    }
}
```

Recap – Intro to programming. Practice



```java
public int factorial(int n) {
    int fact = 1;
    for (int i = n; i > 0; i--) {
        fact *= i;
    }

    return fact;
}
```

Recap – Intro to programming. Practice. Final solution



```java
public class Factorial {
    public static void main(String[] args) {
        int n = 5;
        Factorial.factorial(n);
    }

    private static int factorial(int nr) {
        int f = 1;
        for (int n = nr; n > 0; n--) {
            f *= n;
        }
        System.out.println(f);
        return f;
    }
}
```

Recap – Intro to programming. Practice

```
/*Complete the Java function below to print out all the Strings in the String array parameter in
reverse order. (The String at the highest index should be printed first, then the String at the next
highest index, and so on. The String at index 0 should be printed last.)
For example, if a String array called weekdays had value
{"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"} then this function call:
printInVerverse(weekdays);
would print:
Friday
Thursday
Wednesday
Tuesday
Monday

Starting code:
*/
public void printInReverse(String[] stringArray) {
}
```

Recap – Intro to programming. Practice

```
/*Complete the Java function below to print out all the Strings in the String array parameter in
reverse order. (The String at the highest index should be printed first, then the String at the next
highest index, and so on. The String at index 0 should be printed last.)
For example, if a String array called weekdays had value
{"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"} then this function call:
printInVerverse(weekdays);
would print:
Friday
Thursday
Wednesday
Tuesday
Monday

Starting code:
*/
public void printInReverse(String[] stringArray) {
}
```

Recap – Intro to programming. Practice

```java
// V1
public void printInReverse(String[] stringArray) {
    for (int i = 0; i < stringArray.length; i++) {
        //When i has its smallest possible value, 0, the expression
        //below will be the length of the string array minus one,
        //which is the highest index.  When i has its largest possible
        //value, stringArray.length - 1, this expression will be
        //0, which is the the lowest index.
        int indexToPrint = stringArray.length - 1 - i;
        System.out.println(stringArray[indexToPrint]);
    }
}

// V2
public void printInReverse(String[] stringArray) {
    for (int i = stringArray.length - 1; i >= 0; i--) {
        System.out.println(stringArray[i]);
    }
}
```

Recap – Intro to programming. Practice

```java
public class Rev {
    private static void getReverse(String[] myArray) {
        System.out.println(myArray.length);
        for (int i = 0; i < myArray.length; i++) {
            int indx = myArray.length - i - 1;
            System.out.println(indx);
            System.out.println(myArray[indx]);
        }
    }

    public static void main(String[] args) {
        String[] weekDays = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
        Rev.getReverse(weekDays);
    }
}
```

Recap – OOP

Basics

Encapsulation

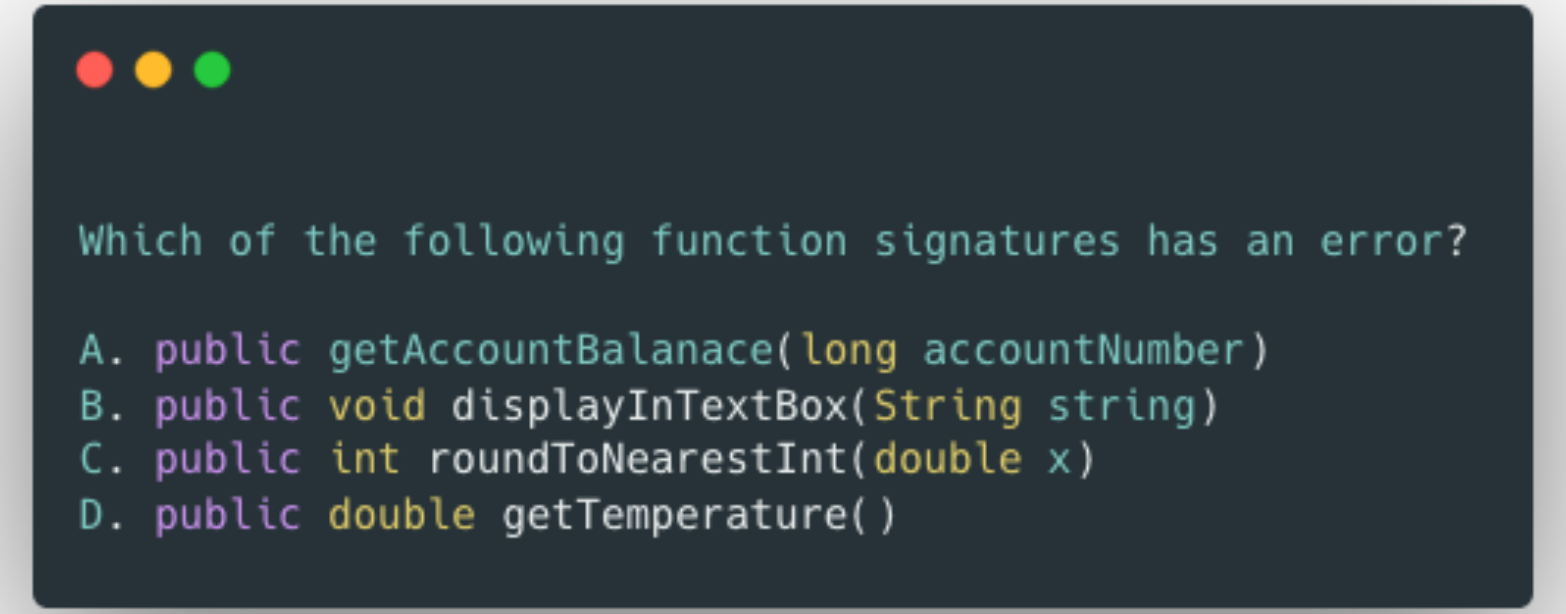
Inheritance

Abstraction

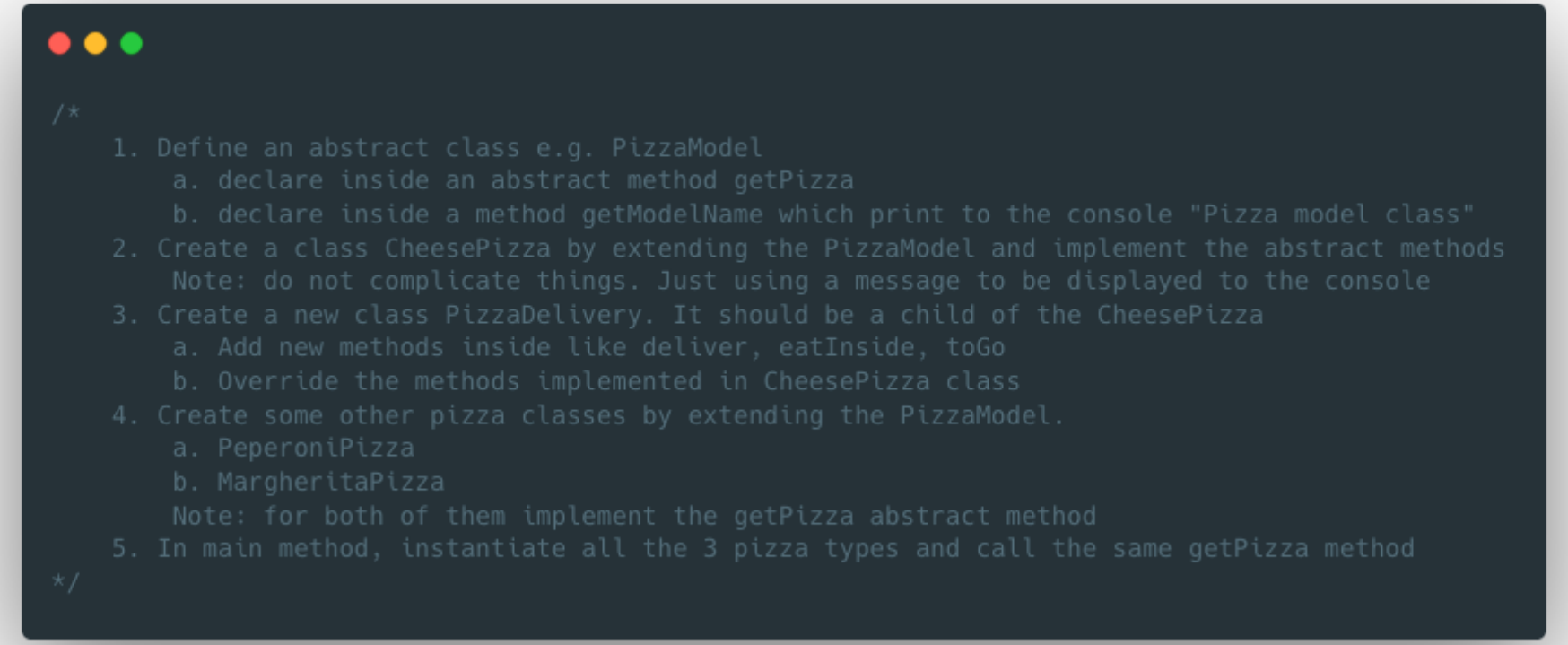
Polymorphism

Recap – Practice

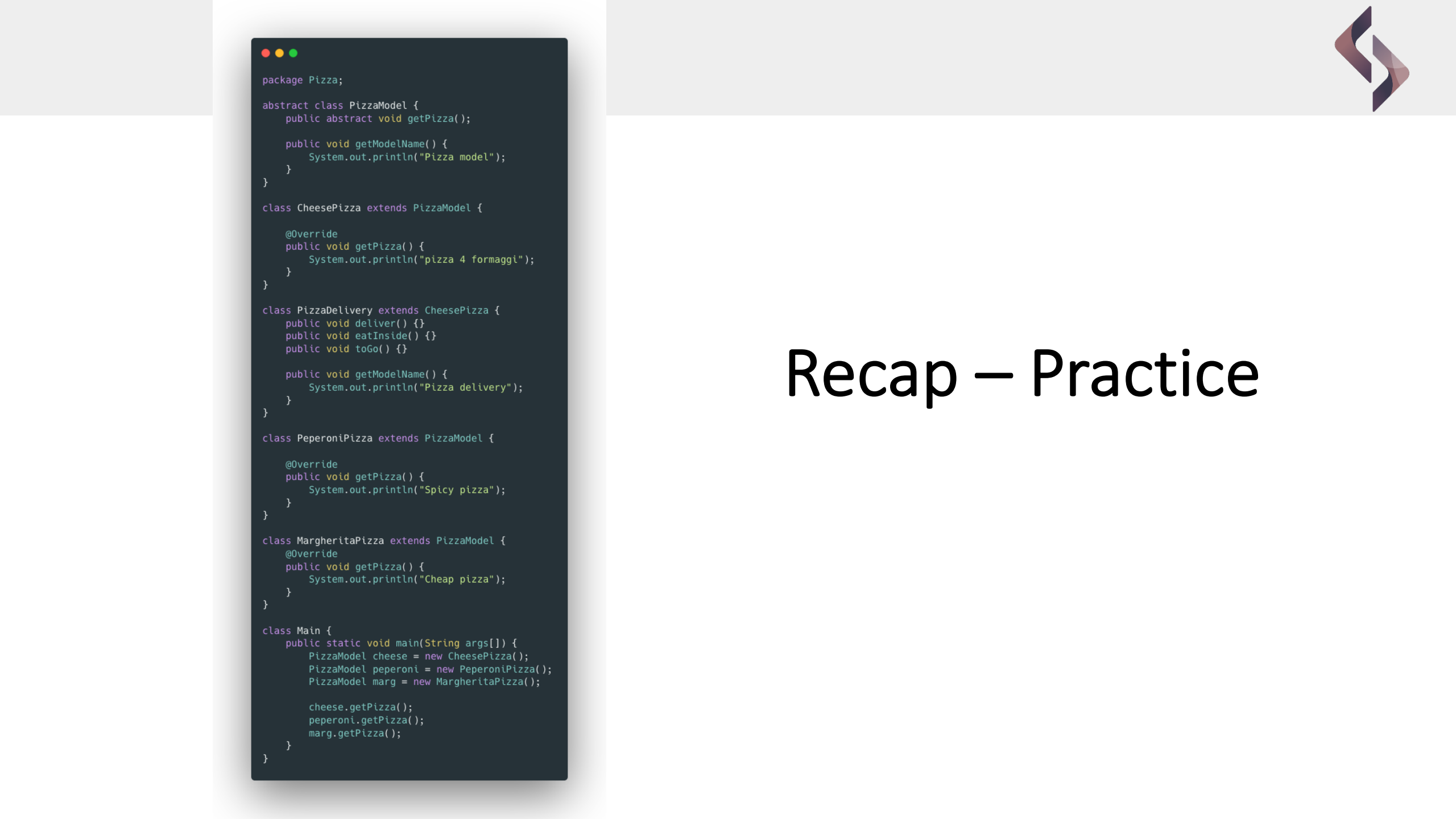
Recap – Practice



Recap – Practice



```
/*
    1. Define an abstract class e.g. PizzaModel
        a. declare inside an abstract method getPizza
        b. declare inside a method getModelName which print to the console "Pizza model class"
    2. Create a class CheesePizza by extending the PizzaModel and implement the abstract methods
        Note: do not complicate things. Just using a message to be displayed to the console
    3. Create a new class PizzaDelivery. It should be a child of the CheesePizza
        a. Add new methods inside like deliver, eatInside, toGo
        b. Override the methods implemented in CheesePizza class
    4. Create some other pizza classes by extending the PizzaModel.
        a. PeperoniPizza
        b. MargheritaPizza
        Note: for both of them implement the getPizza abstract method
    5. In main method, instantiate all the 3 pizza types and call the same getPizza method
*/
```



```
package Pizza;

abstract class PizzaModel {
    public abstract void getPizza();

    public void getModelName() {
        System.out.println("Pizza model");
    }
}

class CheesePizza extends PizzaModel {

    @Override
    public void getPizza() {
        System.out.println("pizza 4 formaggi");
    }
}

class PizzaDelivery extends CheesePizza {
    public void deliver() {}
    public void eatInside() {}
    public void toGo() {}

    public void getModelName() {
        System.out.println("Pizza delivery");
    }
}

class PeperoniPizza extends PizzaModel {

    @Override
    public void getPizza() {
        System.out.println("Spicy pizza");
    }
}

class MargheritaPizza extends PizzaModel {
    @Override
    public void getPizza() {
        System.out.println("Cheap pizza");
    }
}

class Main {
    public static void main(String args[]) {
        PizzaModel cheese = new CheesePizza();
        PizzaModel peperoni = new PeperoniPizza();
        PizzaModel marg = new MargheritaPizza();

        cheese.getPizza();
        peperoni.getPizza();
        marg.getPizza();
    }
}
```

Recap – Practice

Theory
How to deal in practice

Before BDD there is TDD

What is BDD

User stories

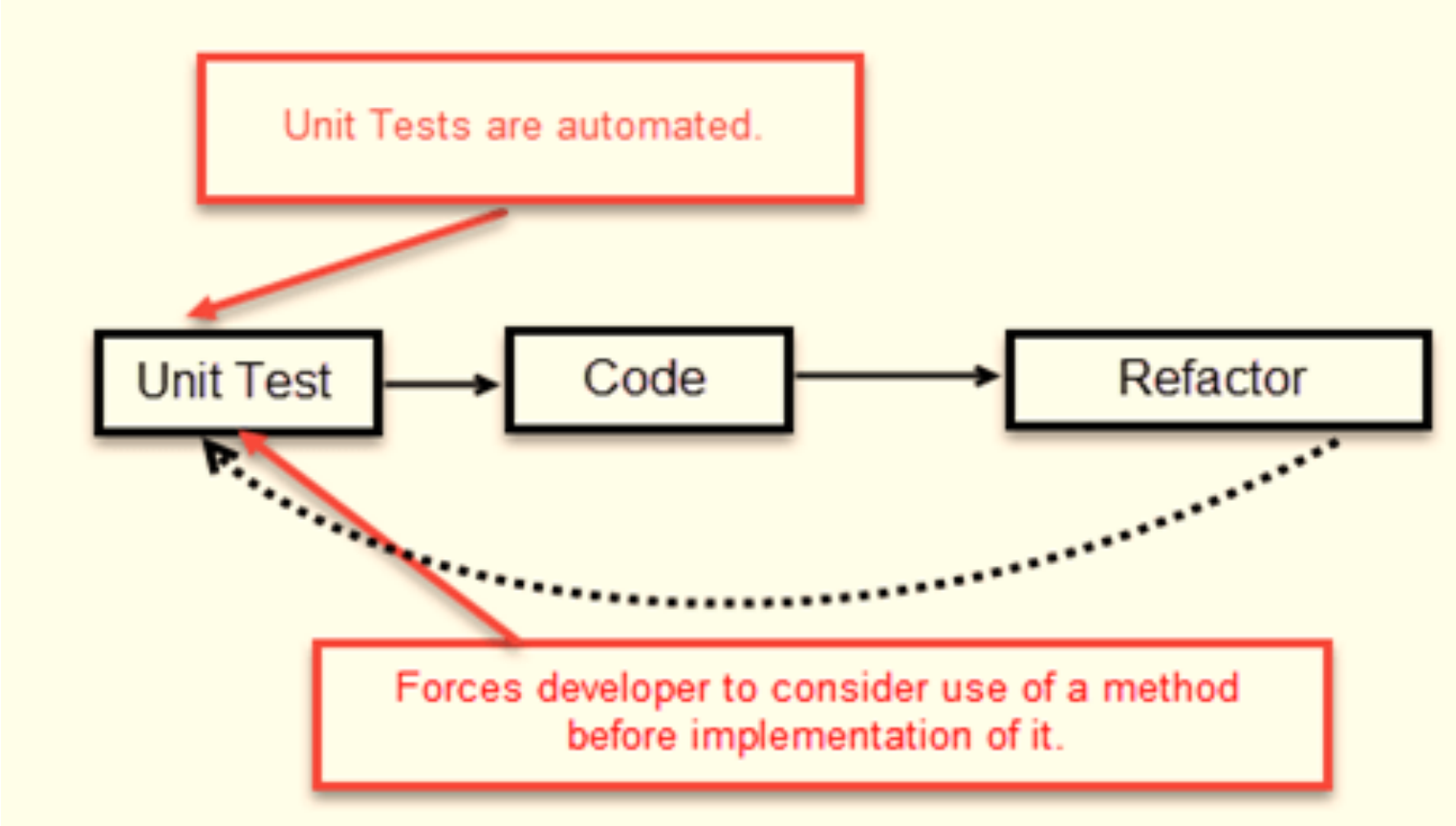
3 important things (feature, steps, execution)

Best tutorial ever for BDD

Let’s start!

Theory
Before BDD there is TDD

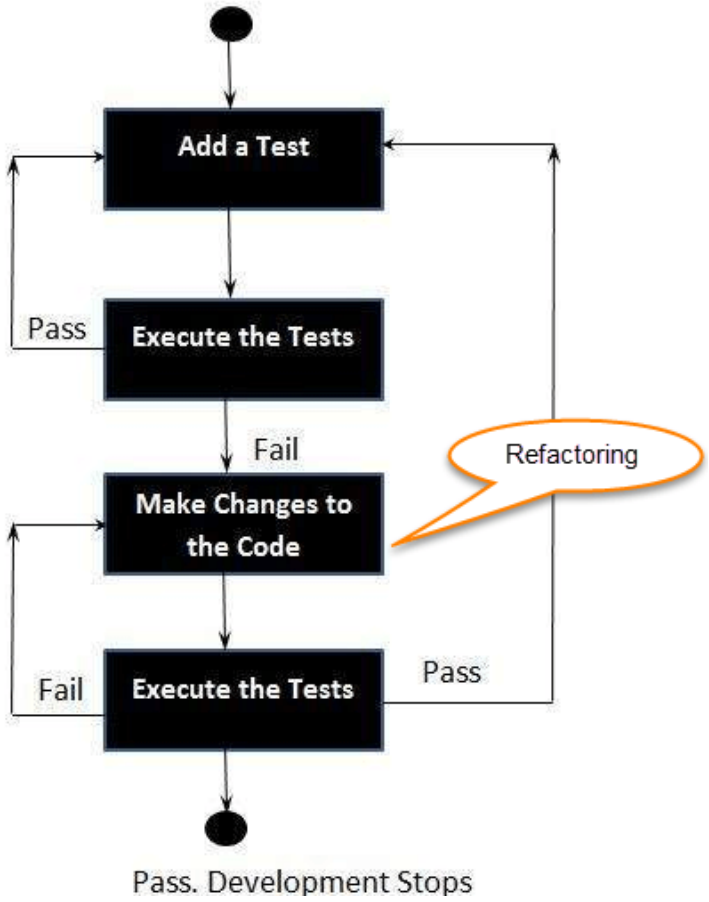
- instructs developers to write new code only if an automated test has failed.
- avoids duplication of code.
- means “Test Driven Development”.
- primary goal - make the code clearer, simple and bug-free.



Theory
Before BDD there is TDD

TDD cycle defines

- Write a test
- Make it run
- Change the code to make it right i.e. Refactor
- Repeat process



Theory
Before BDD there is TDD - Advantages

- Early bug notification.
- Better Designed, cleaner and more extensible code.
- Confidence to Refactor
- Good for teamwork

Theory
What is BDD?

Emerged from TDD

Illustrate the behavior of the system that are written in a readable and understandable language for everyone involved in the development

Providing a shared process and shared tools promoting communication to the software developers, business analysts and stakeholders to collaborate on software development, with the aim of delivering product with business value.

What a system should do and not on how it should be implemented.

Providing better readability and visibility.

Verifying not only the working of the software but also that it meets the customer’s expectations.

Theory
User Stories - What does the User Story consist of?

Title	•One line describing the story
Narration	•Identification of the stakeholder •Function description (Feature) •The reason why the stakeholder is interested in this function •The benefit that the stakeholder will achieve after adding this function •As a [role]I want [feature]So that [benefit]
Acceptance criteria	•They define when ‘enough is enough •On their basis test scenarios for automation are created •Presented as scenarios

Theory
User Stories - Example

```
Title (one line describing the story)
Narrative:
As a [role]
I want [feature]
So that [benefit]
Acceptance Criteria: (presented as Scenarios)

Scenario 1: Title
Given [context]
And [some more context]...
When [event]
Then [outcome]
And [another outcome]...

Scenario 2: ...
```

```
Story: Account Holder withdraws cash

As an Account Holder
I want to withdraw cash from an ATM
So that I can get money when the bank is closed

Scenario 1: Account has sufficient funds
Given the account balance is \$100
 And the card is valid
 And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should dispense \$20
 And the account balance should be \$80
 And the card should be returned
```

Theory
Given When Then - formula

The Given-When-Then formula is a template intended to guide the writing of acceptance tests for a User Story:

- (Given) some context
- (When) some action is carried out
- (Then) a particular set of observable consequences should obtain

An example:

- Given my bank account is in credit, and I made no withdrawals recently,
- When I attempt to withdraw an amount less than my card's limit,
- Then the withdrawal should complete without errors or warnings

Theory
Feature file

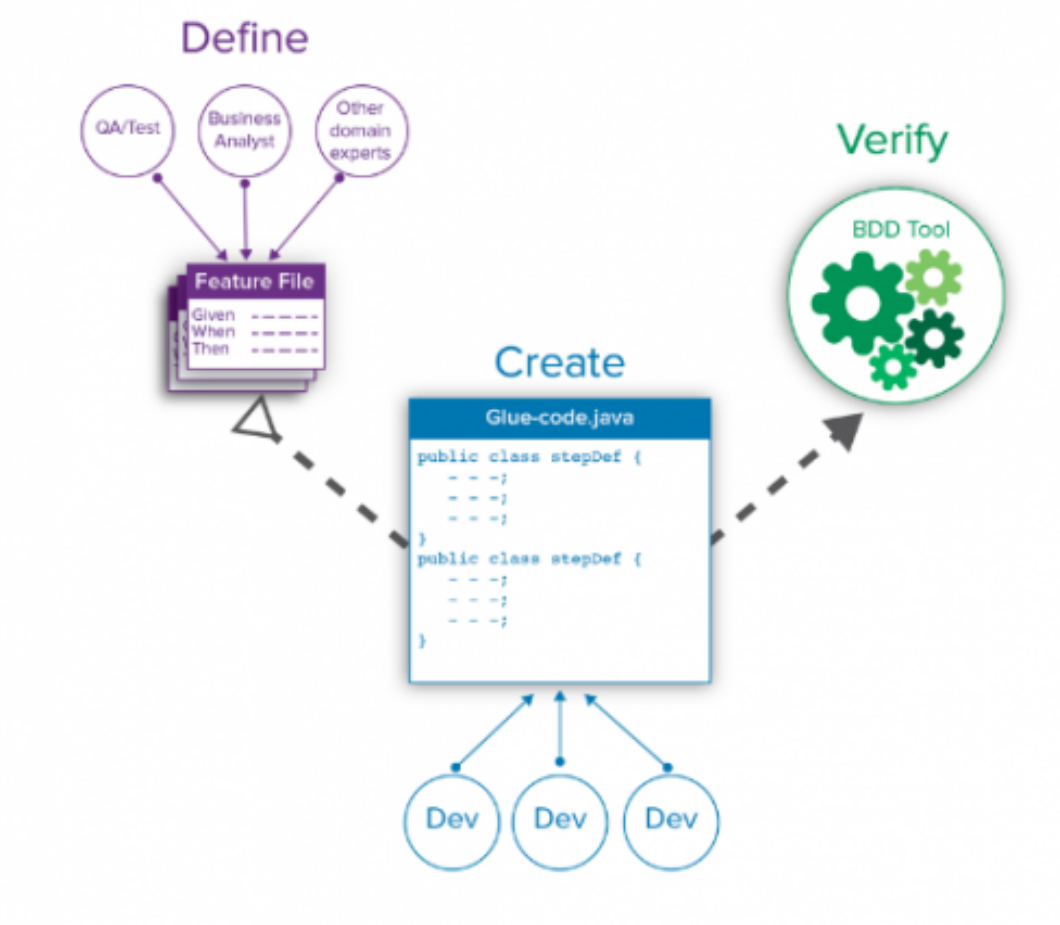
Features file contain high level description of the Test Scenario in simple language, known as Gherkin.

Gherkin is a plain English text language

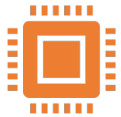
Feature File consist of following components –

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 (I I).	**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"

Theory
3 important things (feature, steps, execution)



Practice – Local setup
Use maven from IntelliJ IDE



If IntelliJ is not installed:

[Download IntelliJ]

else:

Open it

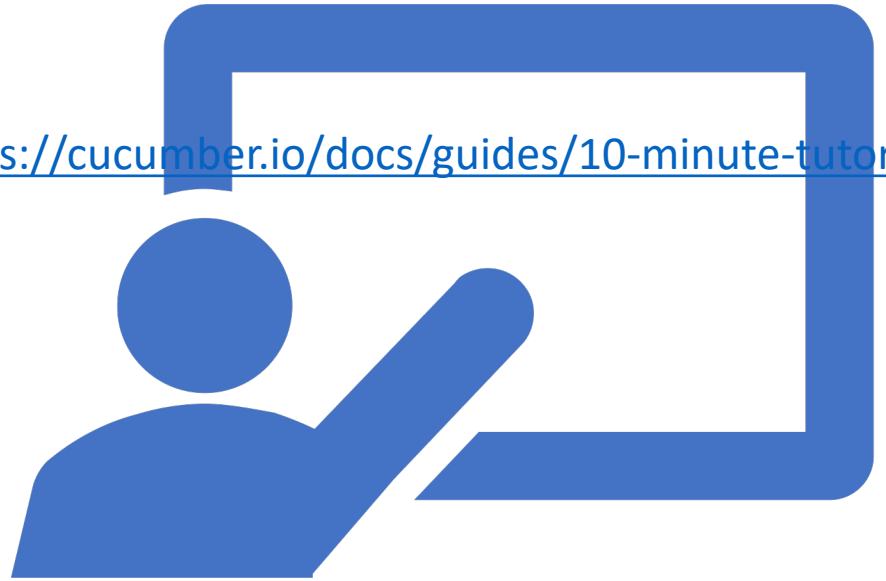
/* Create a new Maven project */

/* Enable auto-import */

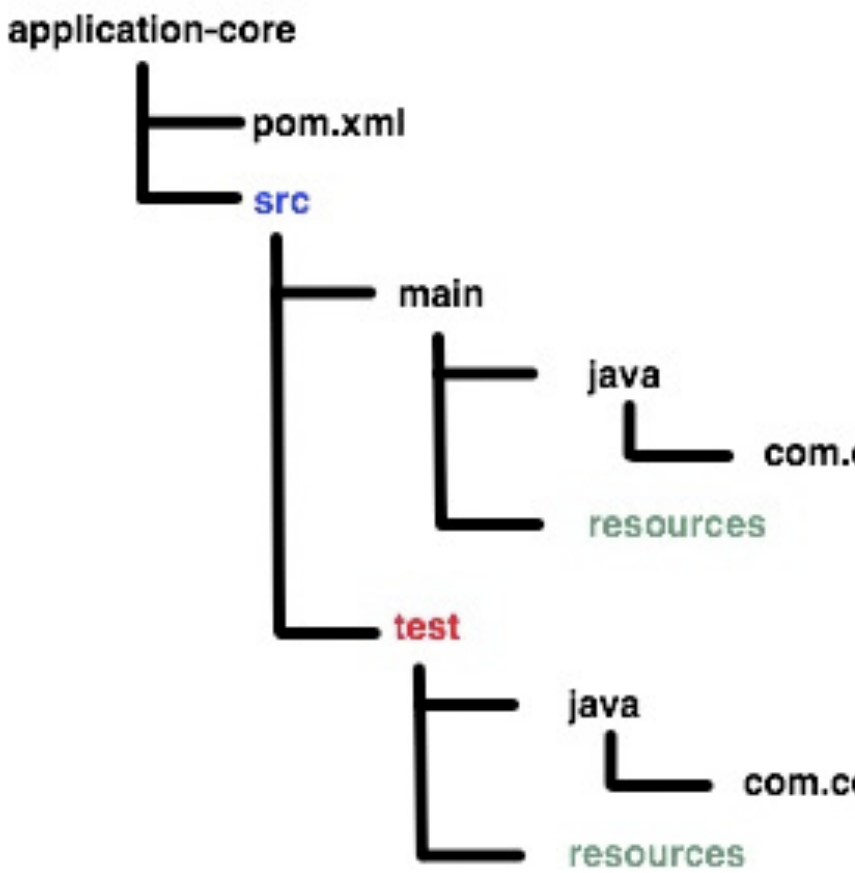
mvn test

Best tutorial ever

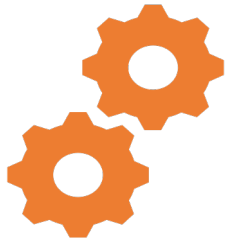
https://cucumber.io/docs/guides/10-minute-tutorial/



Practice – Maven project directory structure



Practice Maven



Maven

Accumulator of knowledge

Simplify the build process

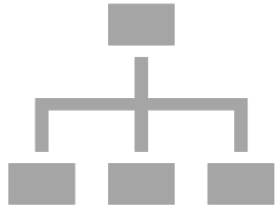
Making the build process easy

Providing a uniform build system

Providing quality project information

Providing guidelines for best practices development

Allowing transparent migration to new features



Archetype

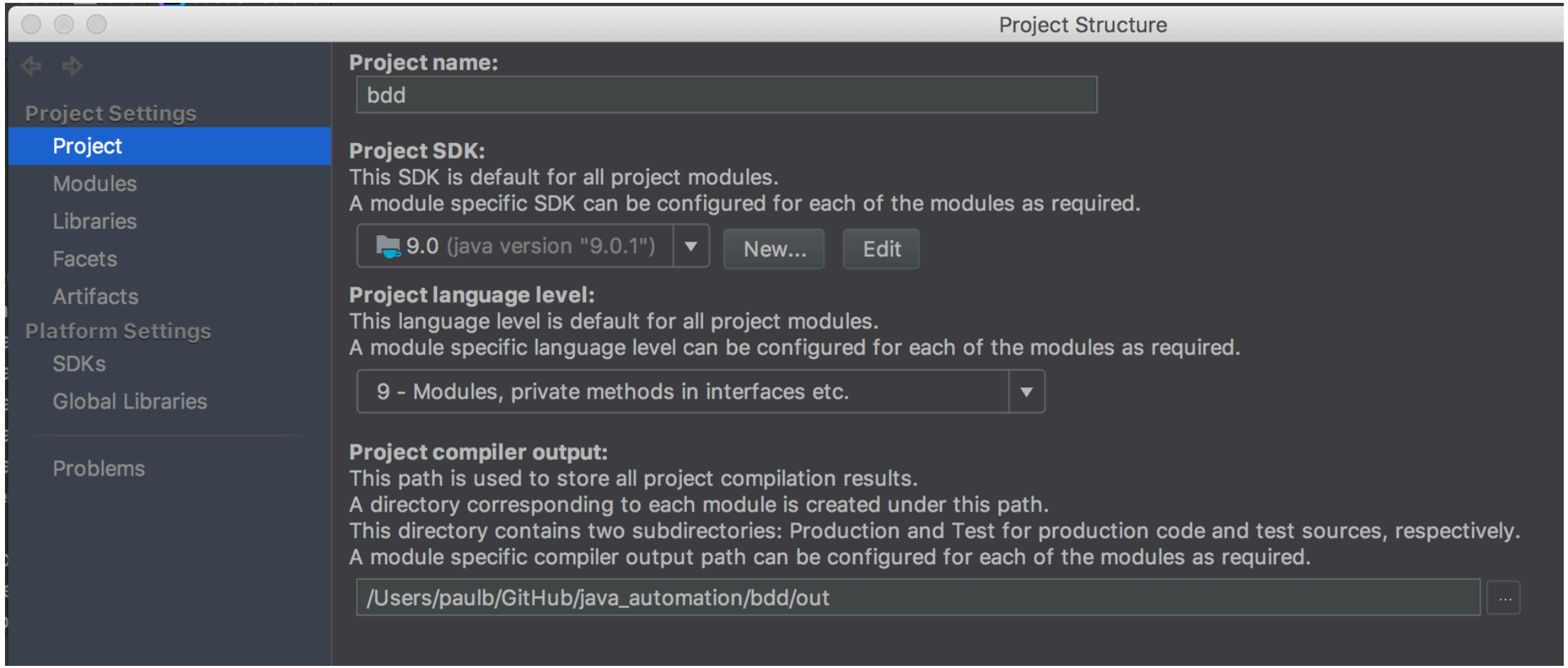
A Maven project templating toolkit

Provides a great way to enable developers quickly in a way consistent with best practices employed by your project or organization

Practice - Maven

- Create a new maven project: File -> New -> Project -> Maven
- Now we won’t use archetype -> Next -> Set GroupID e.g. mymaven -> Set ArtifactId -> mymaven.
- Project name could still be mymaven -> Finish
- Enable auto import
- Go to the Maven icon, click it. Select the M icon and choose mvn test
- Create a new class under the src/main/java, called Main
- Write the main method “it is supposed we all know how to define it”
- Inside the method print “it works”
- If not, there could be a problem with the SDK and try to go to: File -> Project structure -> Change the project SDK (1.8 works fine)

Practice – change SDK



Practice
JSON as a replacement for BDD

What is JSON?

- JSON is short for **JavaScript Object Notation**
- *a way to store information in an organized, easy-to-access manner.*
- human-readable collection of data that we can access in a really logical manner.

Why does JSON matter?

- Easy to define and understand
- Help you to avoid hardcoding content in variables
- Reusable across multiple programming languages

How do we use JSON in a project?

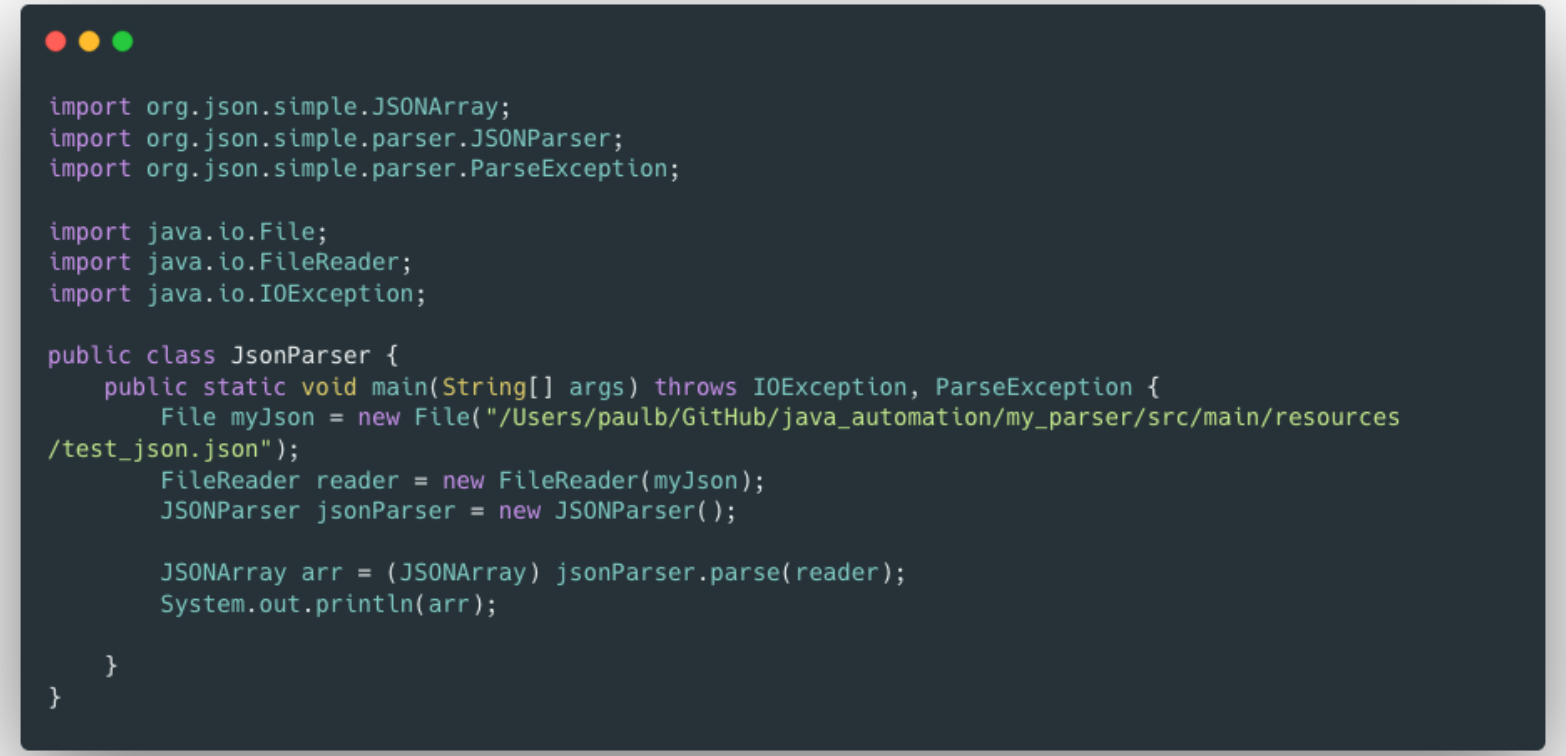
- Store it separated from the code. E.g. in resources
- Need a parsers

Practice
JSON – maven dependencies

```
[
  {
    "testCase":"test nr 1",
    "input":"Berlin",
    "expected":"Berlin"
  },
  {
    "testCase":"test nr 2",
    "input":"Cluj",
    "expected":"Cluj-Napoca"
  }
]
```

```
<dependencies>
    <dependency>
        <groupId>com.googlecode.json-simple</groupId>
        <artifactId>json-simple</artifactId>
        <version>1.1.1</version>
    </dependency>
</dependencies>
```

Practice – Parser. Simple version

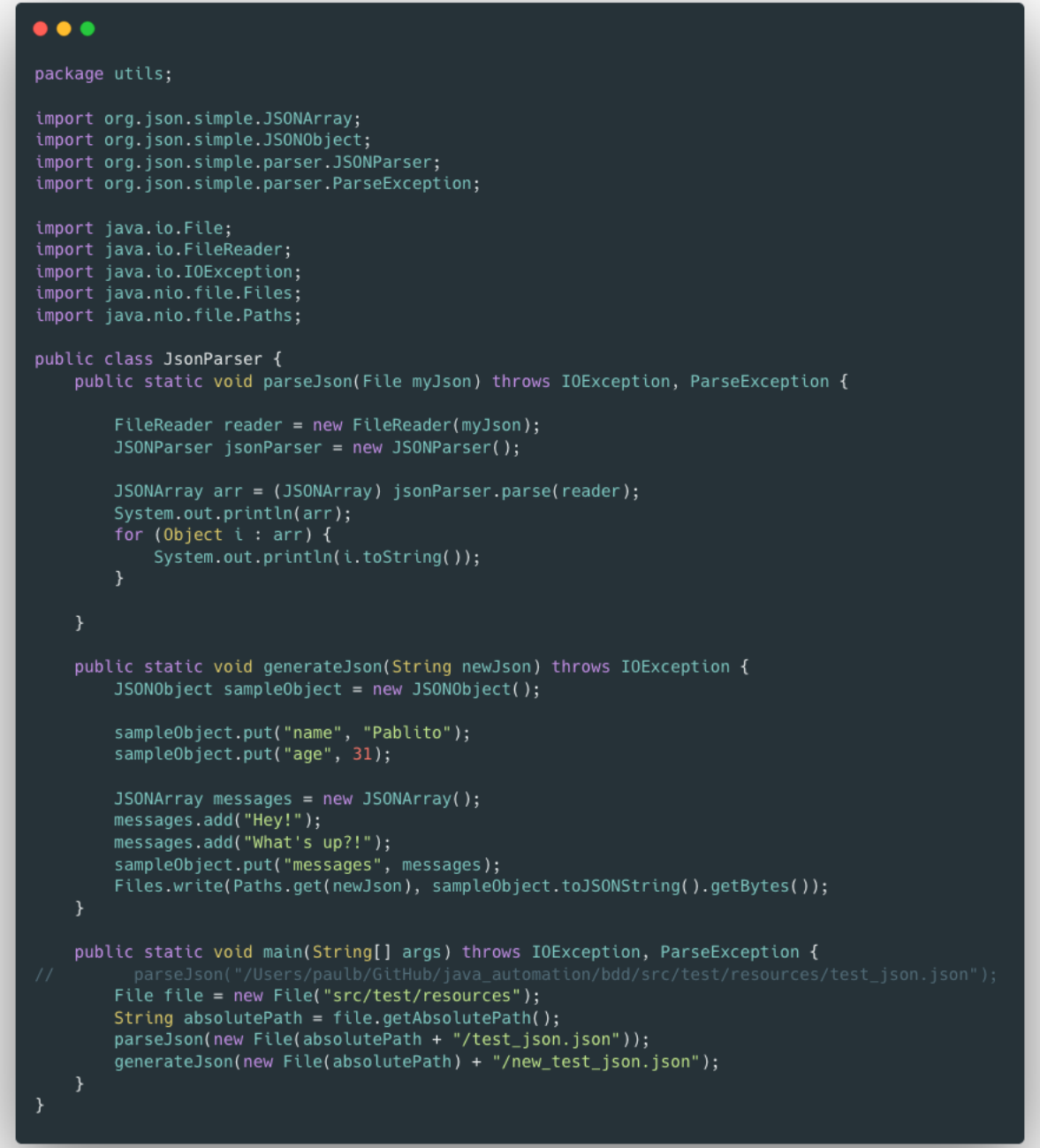


```java
import org.json.simple.JSONArray;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class JsonParser {
    public static void main(String[] args) throws IOException, ParseException {
        File myJson = new File("/Users/paulb/GitHub/java_automation/my_parser/src/main/resources
/test_json.json");
        FileReader reader = new FileReader(myJson);
        JSONParser jsonParser = new JSONParser();

        JSONArray arr = (JSONArray) jsonParser.parse(reader);
        System.out.println(arr);

    }
}
```

Practice – Parser. Simple version

```java
// Get the relative path to the json file
import org.json.simple.JSONArray;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class JsonParser {
    public static void main(String[] args) throws IOException, ParseException {
//        bad practice to use a hardcoded path
//        File myJson = new File("/Users/paulb/GitHub/java_automation/my_parser/src/main/resources
/test_json.json");

        File file = new File("src/main/resources");
        String absolutePath = file.getAbsolutePath();
        FileReader reader = new FileReader(absolutePath + "/test_json.json");
        JSONParser jsonParser = new JSONParser();

        JSONArray arr = (JSONArray) jsonParser.parse(reader);
        System.out.println(arr);

    }
}
```

Practice Parser – extra + generate



```java
package utils;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class JsonParser {
    public static void parseJson(File myJson) throws IOException, ParseException {

        FileReader reader = new FileReader(myJson);
        JSONParser jsonParser = new JSONParser();

        JSONArray arr = (JSONArray) jsonParser.parse(reader);
        System.out.println(arr);
        for (Object i : arr) {
            System.out.println(i.toString());
        }

    }

    public static void generateJson(String newJson) throws IOException {
        JSONObject sampleObject = new JSONObject();

        sampleObject.put("name", "Pablito");
        sampleObject.put("age", 31);

        JSONArray messages = new JSONArray();
        messages.add("Hey!");
        messages.add("What's up?!");
        sampleObject.put("messages", messages);
        Files.write(Paths.get(newJson), sampleObject.toJSONString().getBytes());
    }

    public static void main(String[] args) throws IOException, ParseException {
//        parseJson("/Users/paulb/GitHub/java_automation/bdd/src/test/resources/test_json.json");
        File file = new File("src/test/resources");
        String absolutePath = file.getAbsolutePath();
        parseJson(new File(absolutePath + "/test_json.json"));
        generateJson(new File(absolutePath) + "/new_test_json.json");
    }
}
```

Practice
Unit tests - TDD

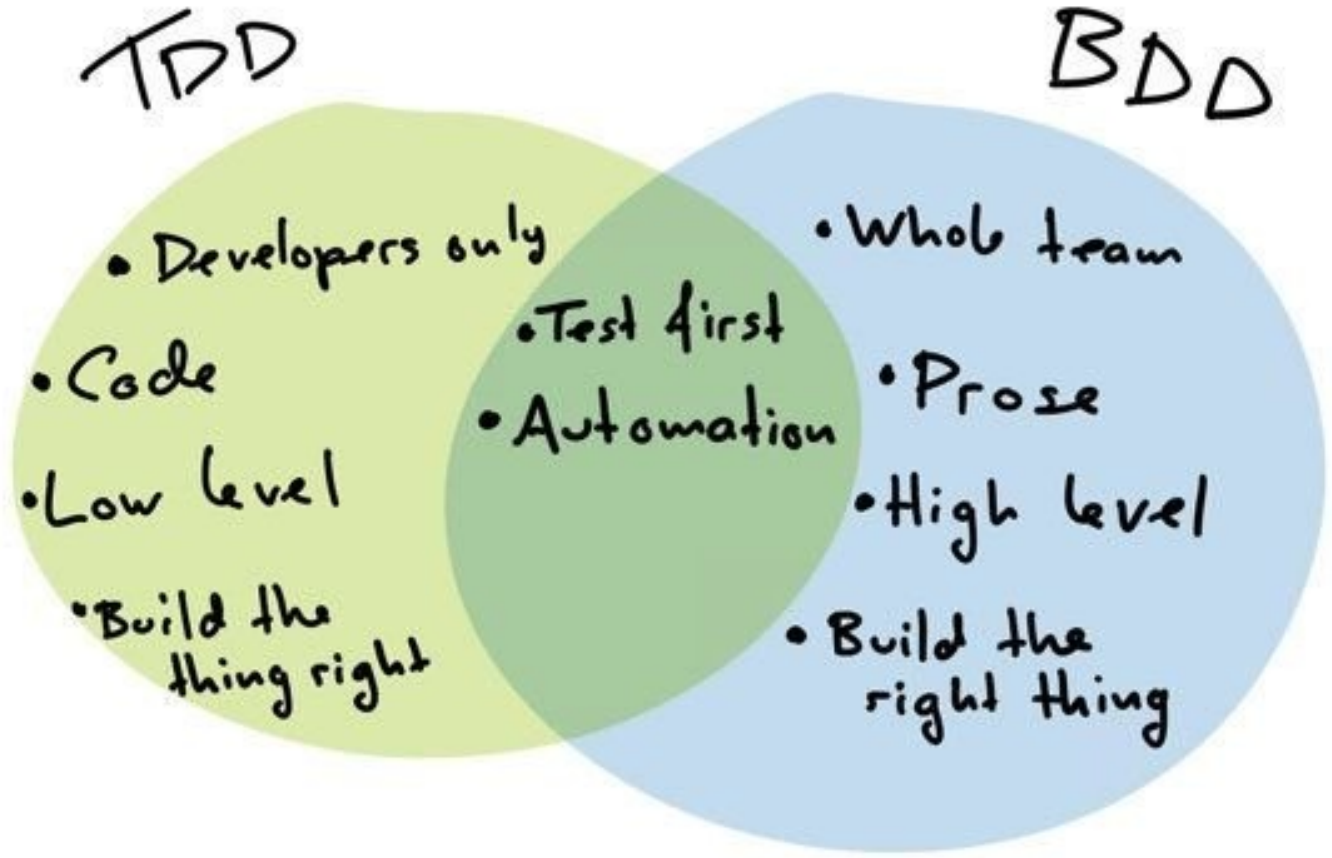
- Create a java package named “tdd”
- Create a new class FirstTest
- Add the maven dependencies to the POM.xml
  - <groupId>junit</groupId>
  - <artifactId>junit</artifactId>
  - <version>4.12</version>
  - <scope>test</scope>
- Write the following code
- Make the tests to pass

```java
package tdd;

import org.junit.Test;


import static junit.framework.TestCase.assertEquals;
import static junit.framework.TestCase.assertTrue;

public class FirstTest {
    @Test
    public void test() {
        String myStr = "my name is ";
        assertTrue("Checking the strings.", myStr.equals("My name is Nea Gigi"));

    }

    @Test
    public void testTwoObjects() {
        assertEquals(11.11, 11.12, 0);
    }
}
```

Practice
Unit tests - TDD

fail([message])	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)	Checks that the boolean condition is true.
assertFalse([message,] boolean condition)	Checks that the boolean condition is false.
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message,] object)	Checks that the object is null.
assertNotNull([message,] object)	Checks that the object is not null.
assertSame([message,] expected, actual)	Checks that both variables refer to the same object.
assertNotSame([message,] expected, actual)	Checks that both variables refer to different objects.

Theory
BDD vs TDD

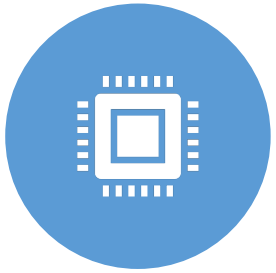


Theory Cucumber

TOOL SUPPORTING THE TESTING PROCESS

ALLOWS YOU TO RUN AUTOMATED TESTS WRITTEN IN BDD

USES A SIMPLE ENGLISH SYNTAX CALLED **'GHERKIN'** TO DESCRIBE TEST SCENARIOS



IT HAS API IMPLEMENTATION NOT ONLY IN JAVA, BUT ALSO OTHER PROGRAMMING LANGUAGES

Theory
Gherkin

Language for describing test scenarios used in BDD approaches (eg in Cucumber)

Created for accessibility for non-technical people

"Simple human collaboration" - focused on connecting people from different parts of the manufacturing process and business (client, developers, analysts, testers, etc.)

Defines the following keywords:

Given, When to specify the initial conditions

Then to determine the expected results

And for increasing the number of conditions / results

But for emphasizing conditions / results

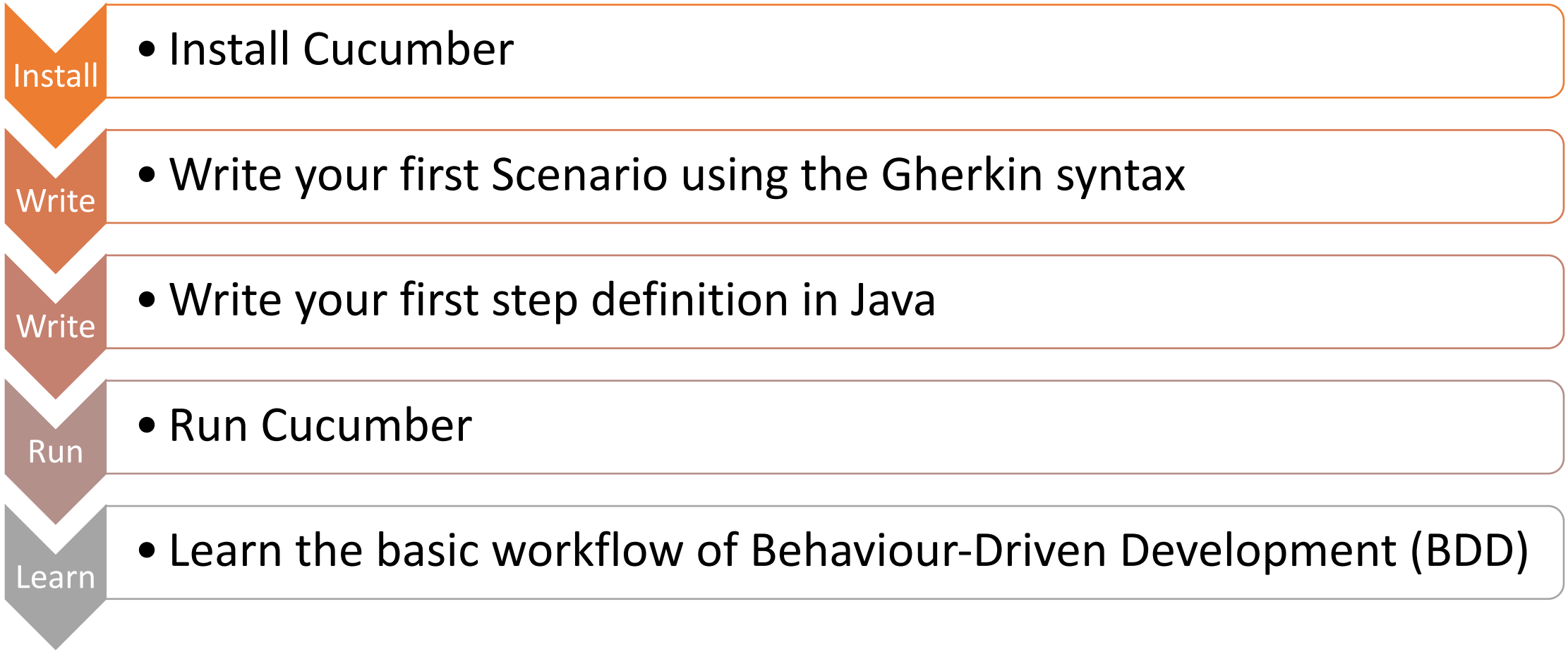
Practice
From JSON to BDD

- Step 1 Write a test case in JSON format
- Step 2 Write a feature file based on the content from JSON format

```
{
  "feature": "Deliver pizza 4 formaggi today?",
  "description": "Our colleagues what to order a pizza 4 formaggi",
  "scenarios": [
    {
      "scenario": "Can we order pizza with lots of cheese today?",
      "given": "",
      "when": "",
      "then": ""
    }
  ]
}
```

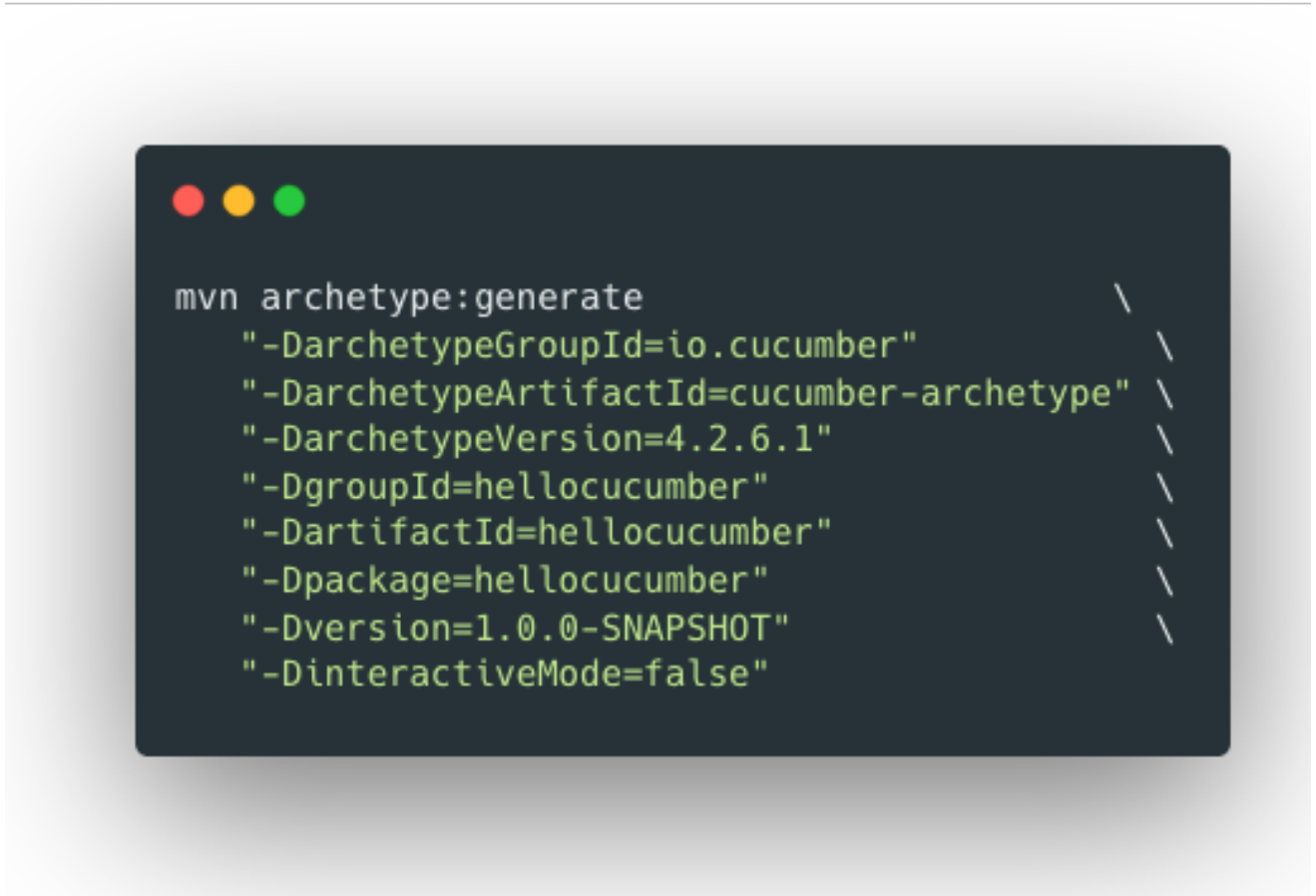
```
Feature: Deliver pizza 4 formaggi today?
  Our colleagues what to order a pizza 4 formaggi

  Scenario: Can we order pizza with lots of cheese today?
    Given we have pizza
    When I ask if you deliver pizza 4 formaggi
    Then You should reply yes
```

Practice
BDD complete tutorial

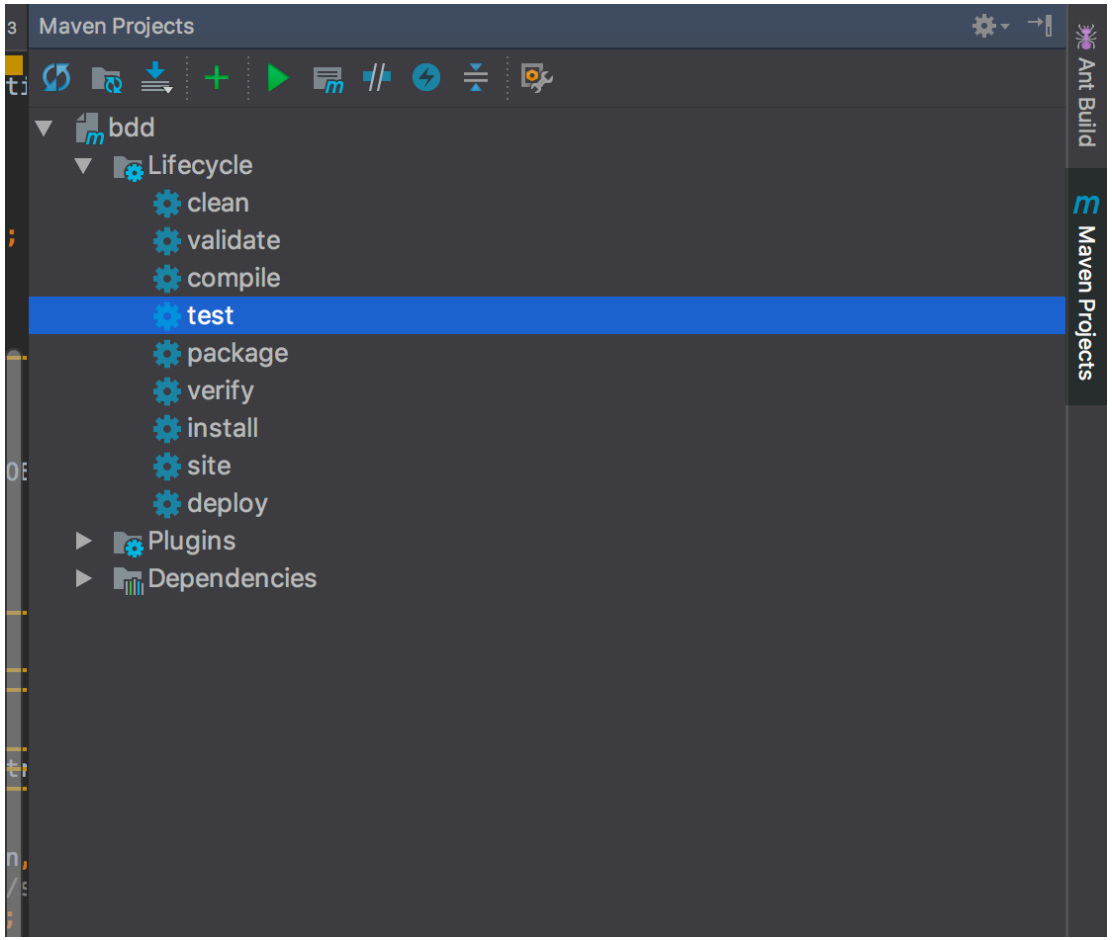


Practice
BDD complete tutorial

- Prerequisites:
  - Intellij
  - Java
  - Maven
- Create an empty Cucumber project
- Use cucumber-archetype
  - io.cucumber
- Verify Cucumber installation
  - mvn test



Practice
BDD complete tutorial – mvn test



Practice
BDD complete tutorial

Practice
BDD complete tutorial - Scenario

```
cucumber.runtime.junit.UndefinedThrowable: The step "step 1" is undefined

cucumber.runtime.junit.UndefinedThrowable: The step "step 2" is undefined

cucumber.runtime.junit.UndefinedThrowable: The step "step 3" is undefined
Feature: initial bdd
  my bdd

  Scenario: as a user I want to learn # simplecucumber/case1.feature:5
    Given step 1                      # null
    When step 2                       # null
    Then step 3                       # null

Undefined scenarios:
simplecucumber/case1.feature:5 # as a user I want to learn

1 Scenarios (1 undefined)
3 Steps (3 undefined)
0m0.187s


You can implement missing steps with the snippets below:

@Given("step {int}")
public void step(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

@When("step {int}")
public void step(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

@Then("step {int}")
public void step(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}
```

```
Feature: initial bdd
  my bdd


  Scenario: as a user I want to learn
    Given step 1
    When step 2
    Then step 3
```

Practice
BDD complete tutorial - Scenario

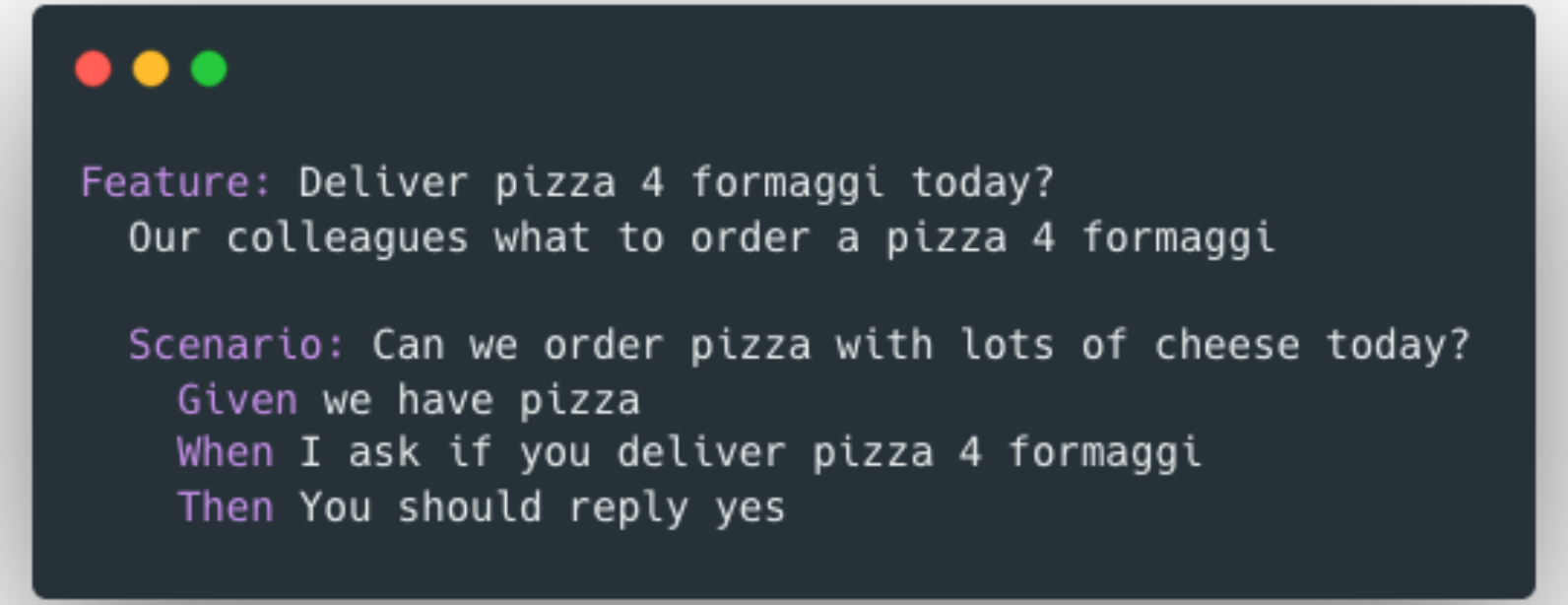
```java
package simplecucumber;

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

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"})
public class RunCucumberTest {
}
```

```java
package simplecucumber;

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

@RunWith(Cucumber.class)
@CucumberOptions(features = {"src/test/resources/simplecucumber/case1.feature"}, plugin = {"pretty"})
public class RunCucumberTest {
}
```

Practice BDD complete tutorial – Run and fix the failure

```java
package simplecucumber;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

import static org.junit.Assert.*;

public class Stepdefs {
    @Given("step 1")
    public void step1() {
        System.out.println("pizza");
    }

    @When("step 2")
    public void step2() {
        System.out.println("step2");
    }

    @Then("step 3")
    public void step3() {
        assertEquals("", "as");
    }

}
```

Practice
BDD complete tutorial - Scenario



```
Feature: Deliver pizza 4 formaggi today?
  Our colleagues what to order a pizza 4 formaggi

  Scenario: Can we order pizza with lots of cheese today?
    Given we have pizza
    When I ask if you deliver pizza 4 formaggi
    Then You should reply yes
```

Practice
BDD complete tutorial – run mvn test again
or use the RunCucumberClass

```
Undefined scenarios:
bdd/case1.feature:4 # Can we order pizza with lots of cheese today?

1 Scenarios (1 undefined)
3 Steps (3 undefined)
0m0.131s


You can implement missing steps with the snippets below:

@Given("we have pizza")
public void we_have_pizza() {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

@When("I ask if you deliver pizza {int} formaggi")
public void i_ask_if_you_deliver_pizza_formaggi(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

@Then("You should reply yes")
public void you_should_reply_yes() {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

Tests run: 3, Failures: 0, Errors: 0, Skipped: 3, Time elapsed: 0.363 sec
```

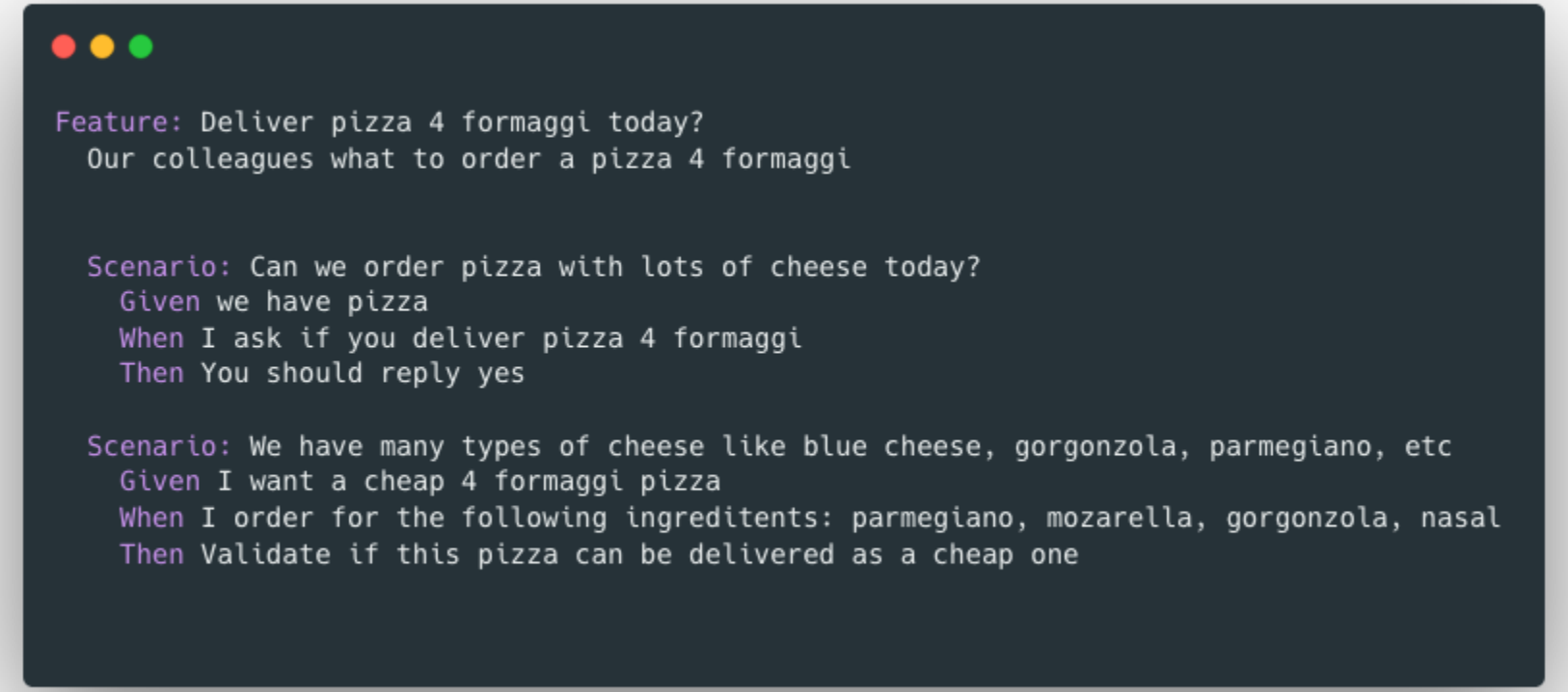
Practice - BDD complete tutorial

- Write code here that turns the phrase above into concrete actions
- Do not complicate yourself

```java
package bdd;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

import static org.junit.Assert.*;

public class Stepdefs {
    @Given("we have pizza")
    public void need_of_pizza() {
        System.out.println("pizza");
    }

    @When("I ask if you deliver pizza 4 formaggi")
    public void order() {
        System.out.println("order pizza");
    }

    @Then("You should reply yes")
    public void check_availability() {
        assertEquals("", "");
    }

}
```

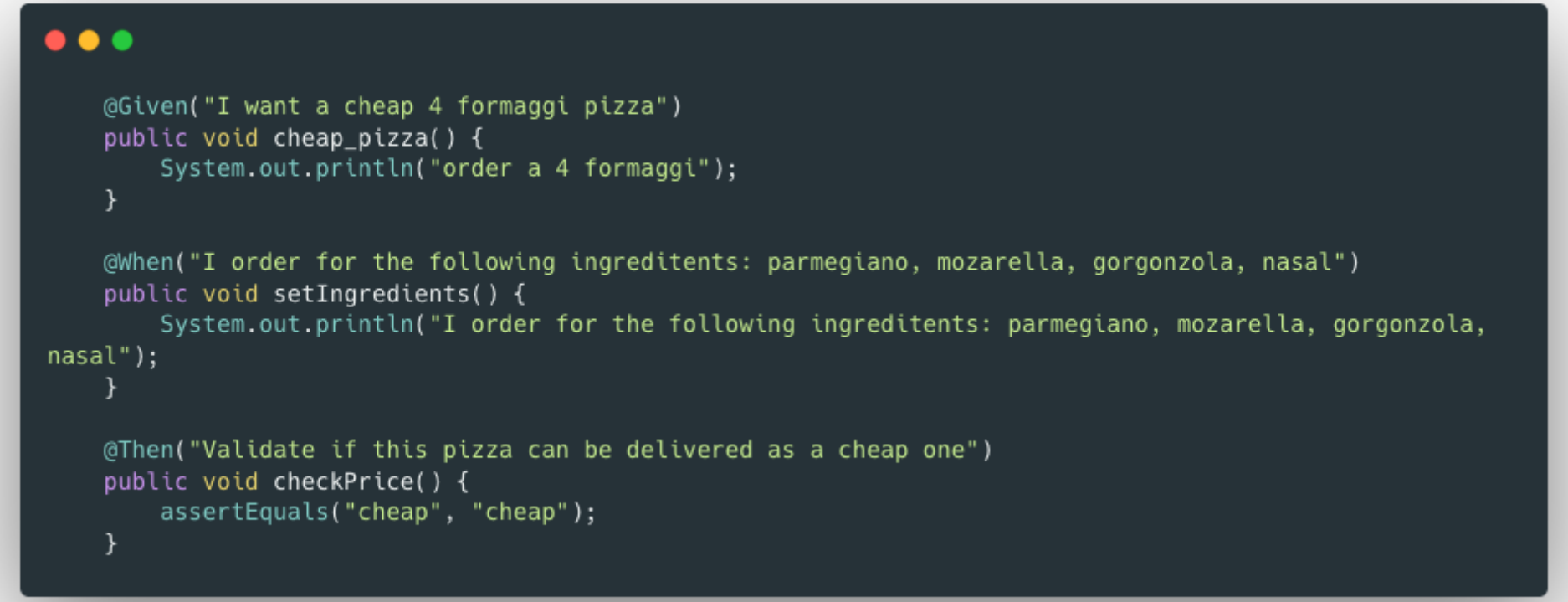
Practice - BDD complete tutorial
Run a separated test

Practice - BDD complete tutorial



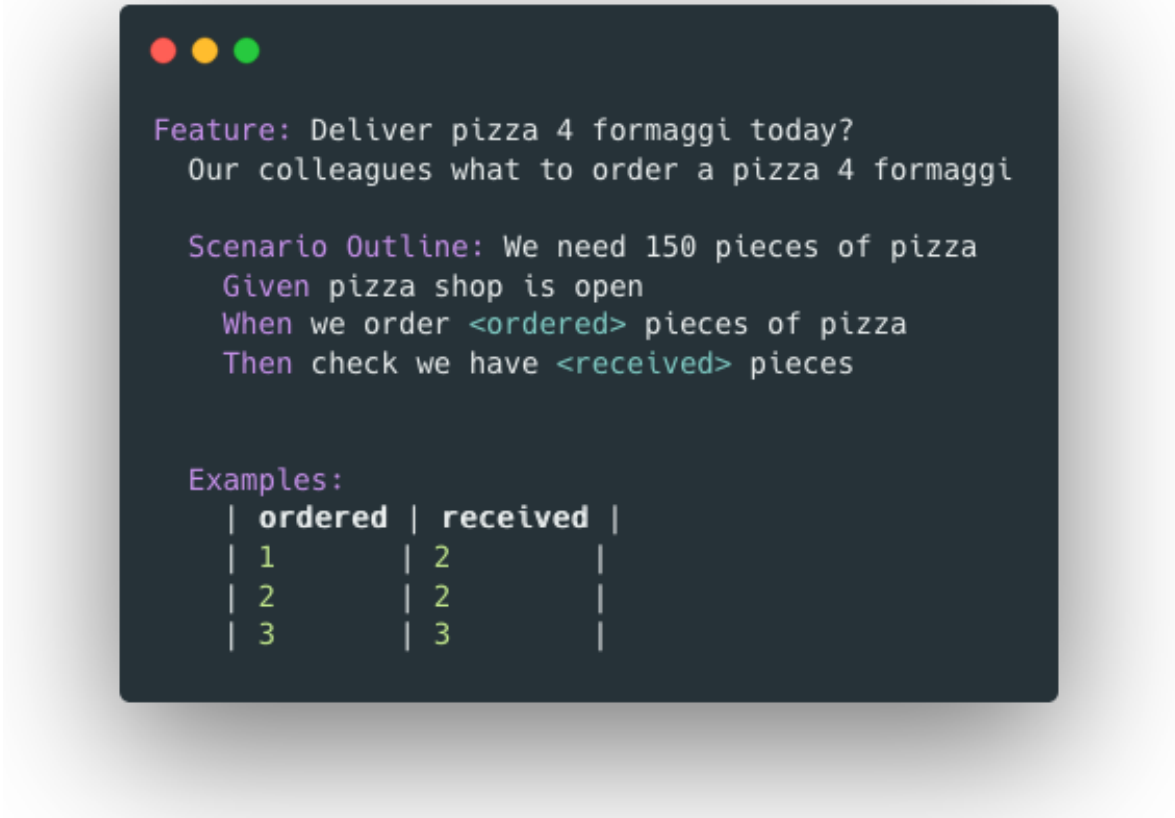
```
Feature: Deliver pizza 4 formaggi today?
  Our colleagues what to order a pizza 4 formaggi


  Scenario: Can we order pizza with lots of cheese today?
    Given we have pizza
    When I ask if you deliver pizza 4 formaggi
    Then You should reply yes

  Scenario: We have many types of cheese like blue cheese, gorgonzola, parmegiano, etc
    Given I want a cheap 4 formaggi pizza
    When I order for the following ingreditents: parmegiano, mozarella, gorgonzola, nasal
    Then Validate if this pizza can be delivered as a cheap one
```

Practice - BDD complete tutorial



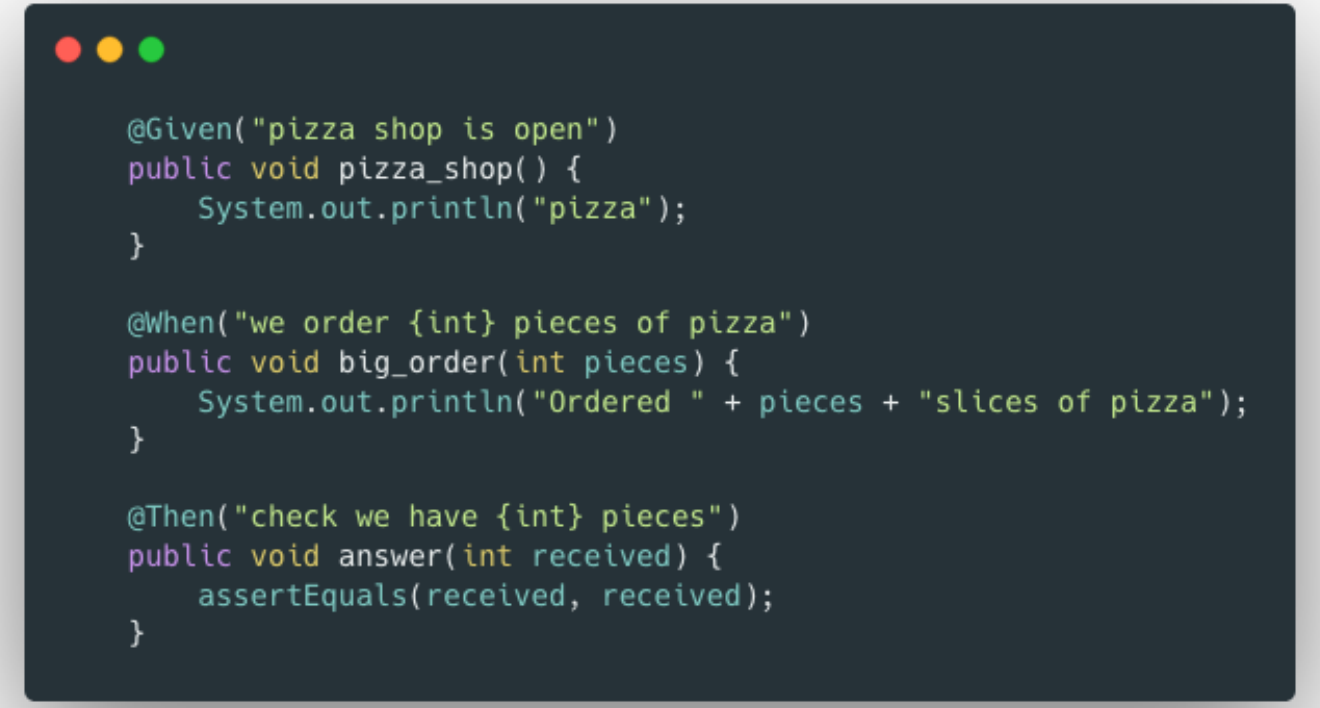
```
    @Given("I want a cheap 4 formaggi pizza")
    public void cheap_pizza() {
        System.out.println("order a 4 formaggi");
    }

    @When("I order for the following ingreditents: parmegiano, mozarella, gorgonzola, nasal")
    public void setIngredients() {
        System.out.println("I order for the following ingreditents: parmegiano, mozarella, gorgonzola,
nasal");
    }

    @Then("Validate if this pizza can be delivered as a cheap one")
    public void checkPrice() {
        assertEquals("cheap", "cheap");
    }
```

Practice
BDD complete tutorial



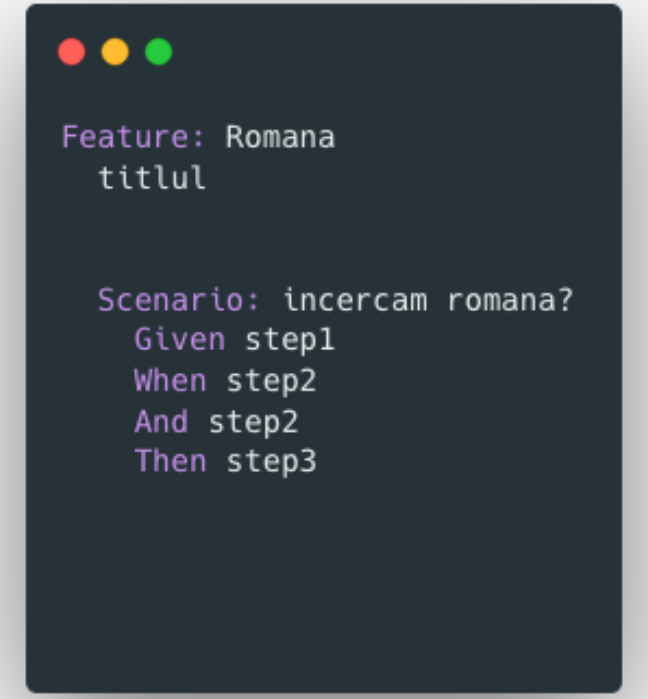
```
Feature: Deliver pizza 4 formaggi today?
  Our colleagues what to order a pizza 4 formaggi

  Scenario Outline: We need 150 pieces of pizza
    Given pizza shop is open
    When we order <ordered> pieces of pizza
    Then check we have <received> pieces


  Examples:
    | ordered | received |
    | 1       | 2        |
    | 2       | 2        |
    | 3       | 3        |
```

Practice
BDD complete tutorial



```java
@Given("pizza shop is open")
public void pizza_shop() {
    System.out.println("pizza");
}

@When("we order {int} pieces of pizza")
public void big_order(int pieces) {
    System.out.println("Ordered " + pieces + "slices of pizza");
}

@Then("check we have {int} pieces")
public void answer(int received) {
    assertEquals(received, received);
}
```

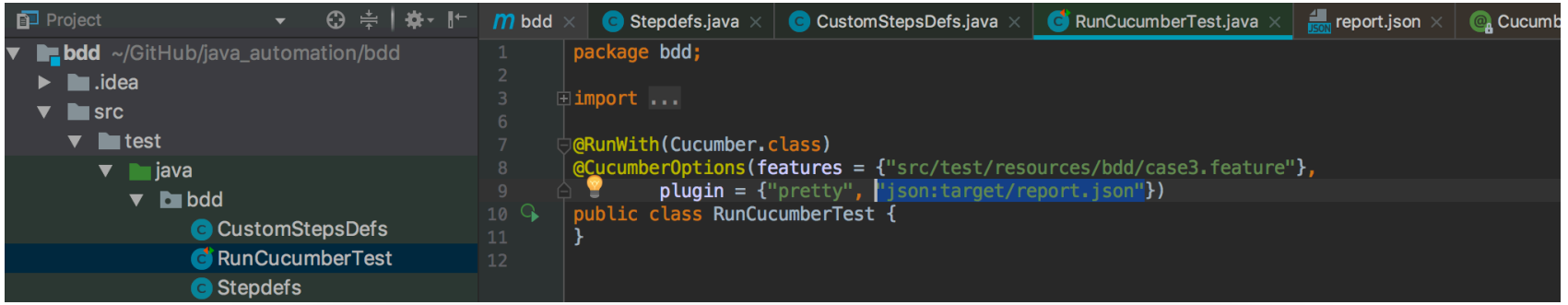
Practice
BDD complete tutorial



```
package bdd;
import cucumber.api.java.ro.*;
import static org.junit.Assert.*;

public class CustomStepsDefs {
    @Când("step1")
    public void step1() {
        System.out.println("step1");
    }

    @Atunci("step2")
    public void step2() {
        System.out.println("step2");
    }

    @Datefiind("step3")
    public void step3() {
        assertTrue(true);
    }
}
```

Practice BDD complete tutorial

Practice - Generate reports – JSON, HTML




```
plugin = {"pretty", "json:target/report.json", "html:target/cucumber-reports/"})
```

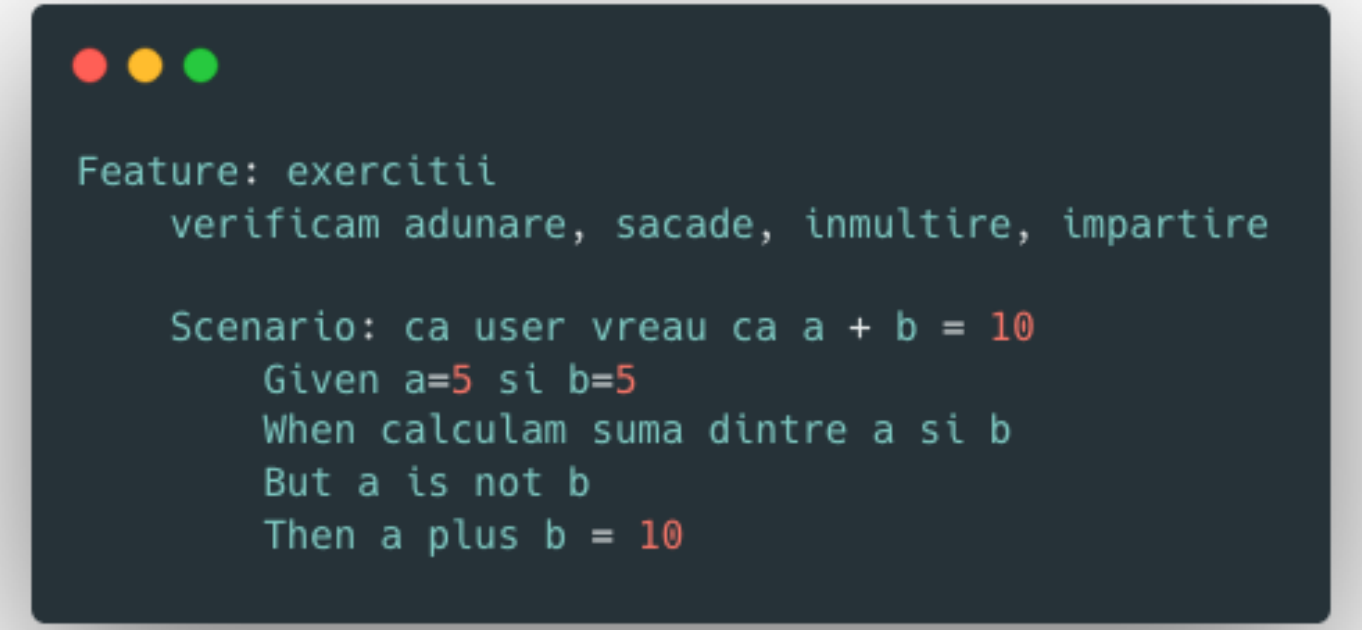
Practice - Generate reports - HTML

▼ **Feature**: Deliver pizza 4 formaggi today?
Our colleagues what to order a pizza 4 formaggi
▶ **Scenario**: Can we order pizza with lots of cheese today?
▶ **Scenario**: We have many types of cheese like blue cheese, gorgonzola, parmegiano, etc
▼ **Feature**: Deliver pizza 4 formaggi today?
Our colleagues what to order a pizza 4 formaggi
▼ **Scenario Outline**: We need 150 pieces of pizza
Given pizza shop is open
When we order <ordered> pieces of pizza
Then check we have <received> pieces
▼ **Examples**:

ordered	received
1	2
2	2
3	3

▶ **Scenario Outline**: We need 150 pieces of pizza
▶ **Scenario Outline**: We need 150 pieces of pizza
▶ **Scenario Outline**: We need 150 pieces of pizza
▼ **Feature**: Romana
titlul
▶ **Scenario**: incercam romana?
▼ **Feature**: Romana
titlul
▼ **Scenario**: incercam romana?
Datefiind step1
Când step2
Atunci step3

Practice - Calculator



```
Feature: exercitii
    verificam adunare, sacade, inmultire, impartire

    Scenario: ca user vreau ca a + b = 10
        Given a=5 si b=5
        When calculam suma dintre a si b
        But a is not b
        Then a plus b = 10
```

Practice - Calculator

```java
import cucumber.api.java.en.*;
import static org.junit.Assert.*;

public class Calculator {
    @Given("a=5 si b=5")
    public void define() {
        System.out.println("5, 5");
    }

    @When("calculam suma dintre a si b")
    public void sum() {
        System.out.println("a + b");
    }


    @Then("a plus b = 10")
    public void checkSum() {
        assertEquals(10, 10);
    }

    @But("a is not b")
    public void something() {
        assertEquals(5, 5);
    }

    @And("extra condition")
    public void extraStep() {
        assertEquals(5, 5);
    }
}
```

Practice - Calculator

```
Feature: exercitii
    verificam adunare, sacade, inmultire, impartire

    Scenario Outline: as a user I want to have the sum of two positive values
        Given <a> and <b>
        But <a> and <b> > 0
        When calculate sum between <a> and <b>
        And step "<name>"
        Then sum should be <sum>


      Examples:
        | a  | b | sum | name   |
        | 1  | 2 | 3   | tc1    |
        | 3  | 2 | 5   | tc2    |
        | 7  | 1 | 8   | tcsal  |
```

Practice - Calculator

```java
import cucumber.api.java.en.*;
import static org.junit.Assert.*;

public class Calculator {
    private int my_a, my_b;

    @Given("{int} and {int}")
    public void s1(int a, int b) {
        this.my_a = a;
        this.my_b = b;
    }

    @When("calculate sum between {int} and {int}")
    public void s2(int a, int b) {
        System.out.println("a + b");
    }

    @But("{int} and {int} > 0")
    public void s3(int a, int b) {
        if (a >= 0 && b >= 0) {
            assertTrue(true);
        } else {
            fail();
        }
    }

    @And("step {string}")
    public void extraStep(String name) {
        if (name.contains("tc")) {
            assertTrue(true);
        } else {
            fail("Bad naming convention for the test case");
        }
    }

    @Then("sum should be {int}")
    public void s4(int sum) {
        int newSum = this.my_a + this.my_b;
        assertEquals(newSum, sum);
    }
}
```

Practice - Calculator

Subtraction
Multiplication
Division

Homework – basic level

1. PARSE THE GENERATED JSON REPORT
2. WRITE A SCENARIO OF HOW WOULD YOU TEST A CAR AND IMPLEMENT THE STEPS INTO A SEPARATED CLASS
3. WRITE A SCENARIO OF HOW WOULD YOU TEST A TENNIS BALL AND IMPLEMENT THE STEPS
4. WRITE A SCENARIO OF HOW WOULD YOU TEST A SEARCH ENGINE USING 10 ADDRESSES INTO THE SAME FEATURE FILE USE “AIRBNB AS A STARTING POINT”

Homework – medium level

1. Write a Gherkin file in Romanian, and try to make it work
2. Convert JSON to HTML. As a QA engineer you always need to find workarounds or adapt to other languages e.g. Python, JavaScript
3. Define some new features based on the Scenario outlines. Define a table with 3 columns
   - First with integers
   - Second with strings as input
   - Last with expected strings
4. Implement the steps to support the execution of the previous feature
5. Integrate Selenium in Java project

Thank you