No API? No problem!

API mocking with WireMock

An open source workshop by …

Originally created by Bas Dijkstra - bas@ontestautomation.com - https://www.ontestautomation.com

What are we going to do?

_Stubbing, mocking and service virtualization

_WireMock

_Exercises, examples, …

Preparation

_Install JDK (Java 8 preferred)

_Install IntelliJ IDEA (or any other IDE)

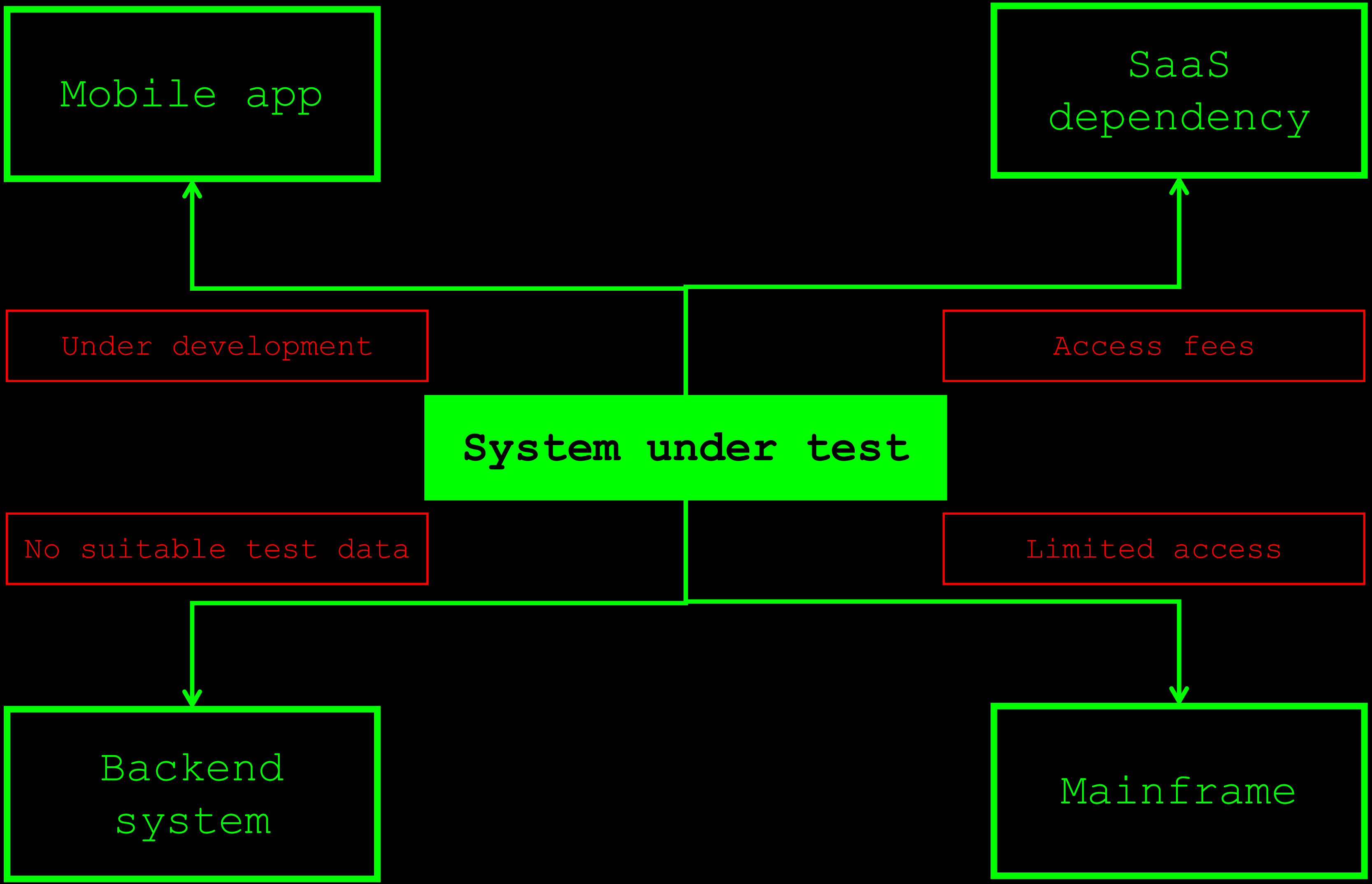
_Download or clone project

_Import Maven project in IDE

Problems in test environments

- Systems are constructed out of of many different components
- Not all of these components are always available for testing
  - Parallel development
  - No control over testdata
  - Fees required for using third party component
  - …

Problems in test environments



Simulation during test execution

_Simulate dependency **behaviour**

_Regain full control over test environment
  _Available on demand
  _Full control over test data (edge cases!)
  _No third party component usage fees
  _…

Stubbing

_Predefined responses

_No flexibility

_Status verification

Mocking

_Define mock behavior during test initialization

_(Somewhat) more flexible

_Behaviour verification

Service virtualization

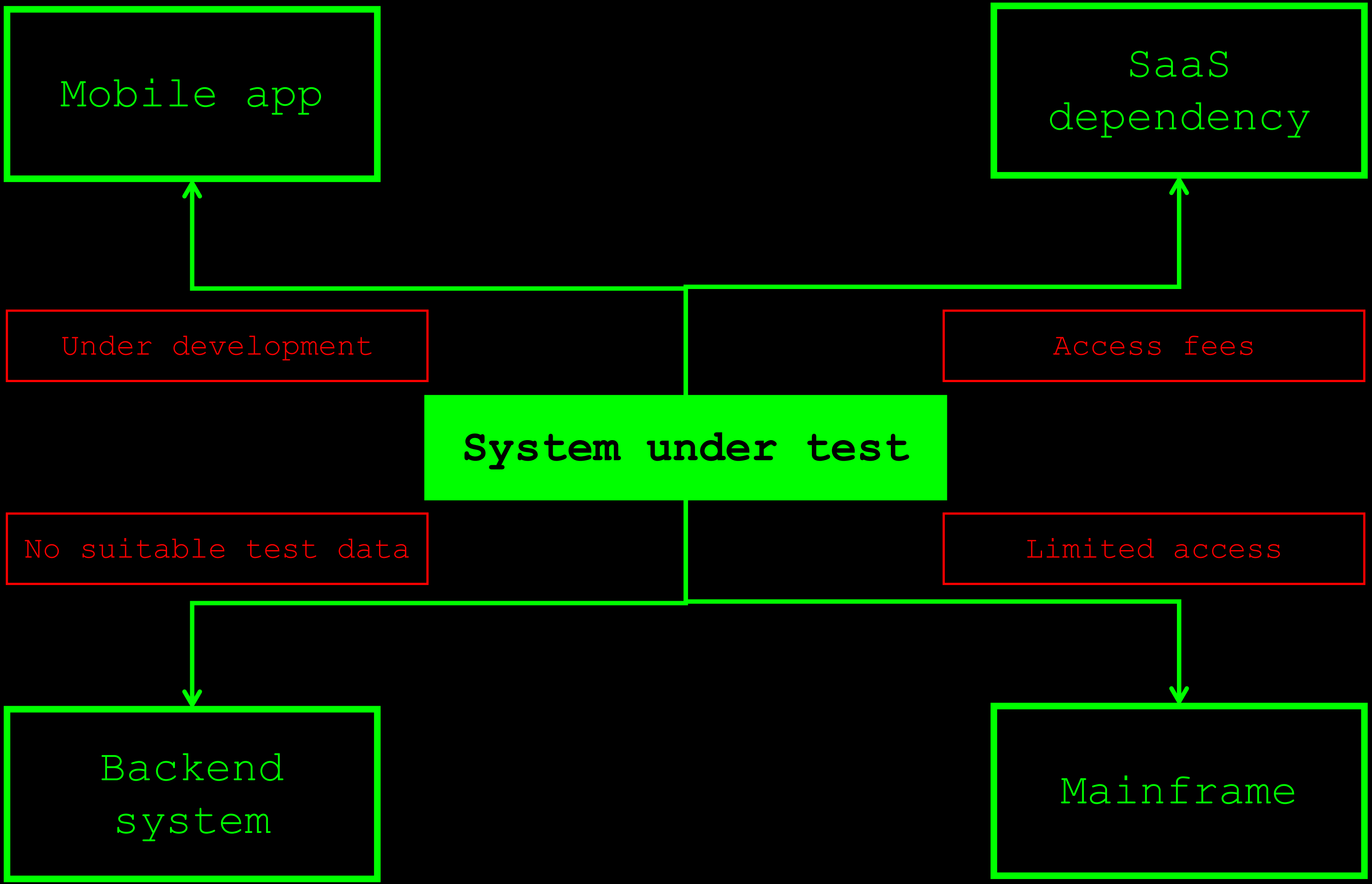
_Simulate complex dependency behaviour

_‘Enterprise level’ stubbing / mocking

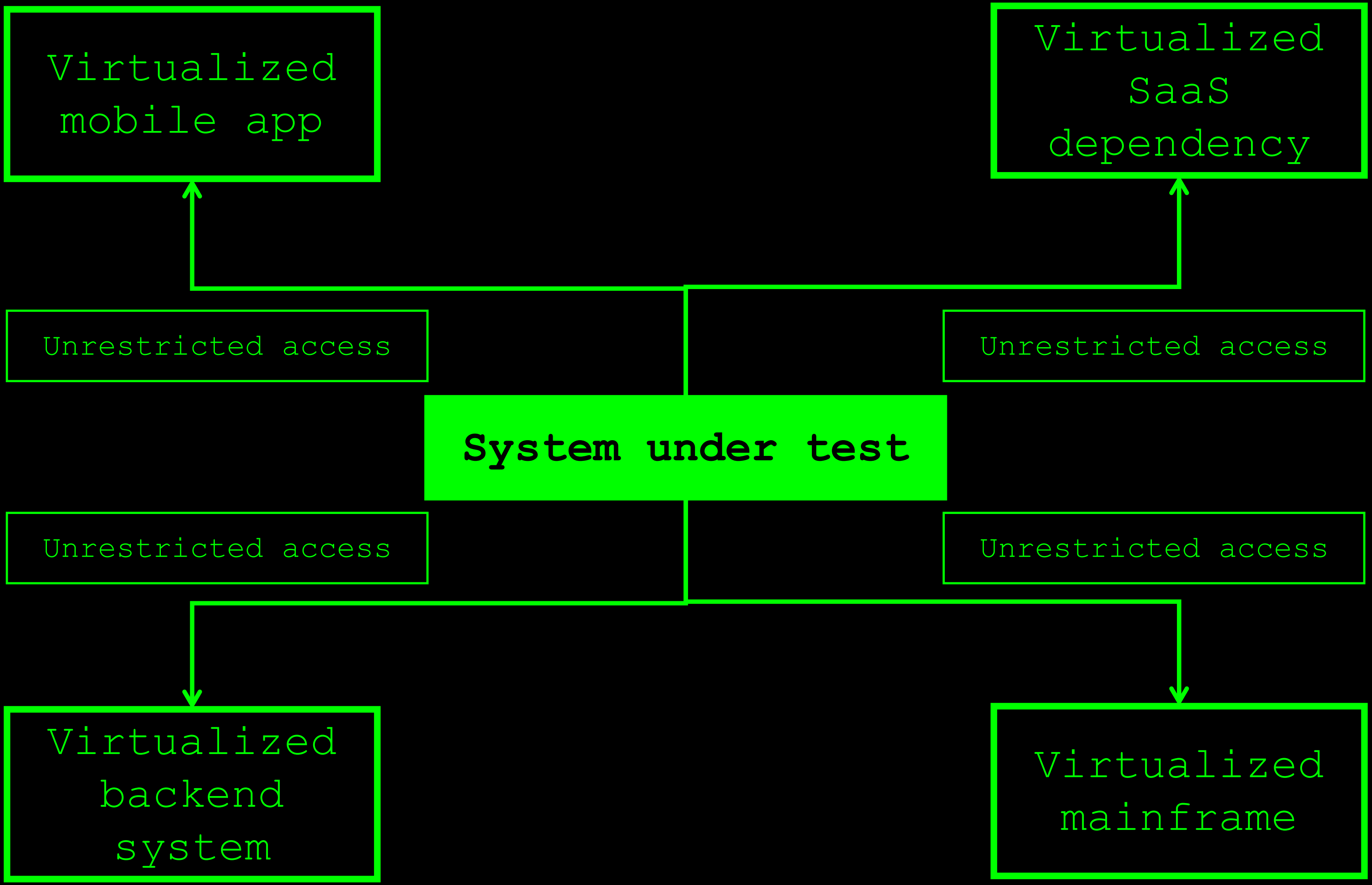
_Support for many different protocols and message formats

_Data driven

Problems in test environments



Simulation in test environments



Our API under test

_Zippopotam.us

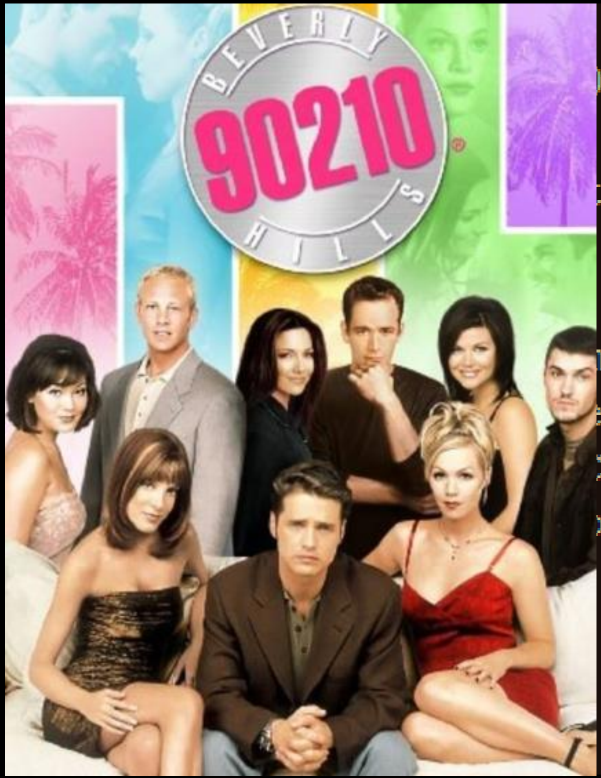
_Returns location data based on country and zip code

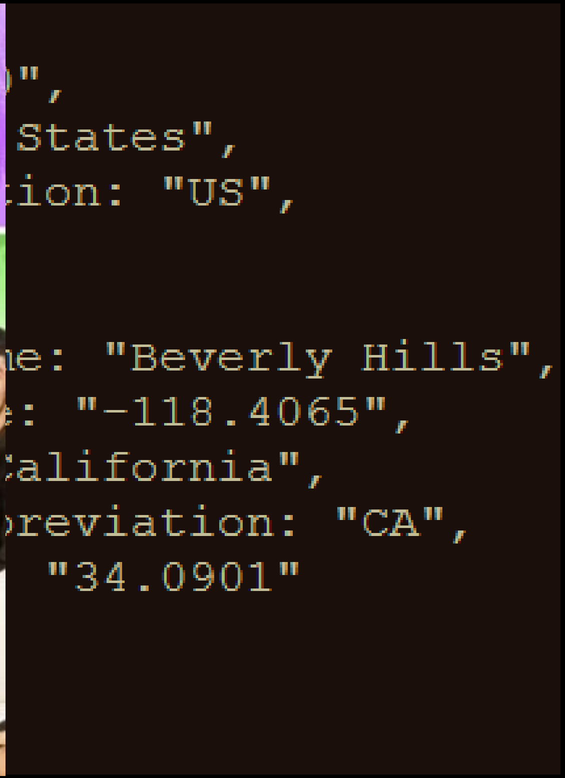
_http://api.zippopotam.us/

_RESTful API

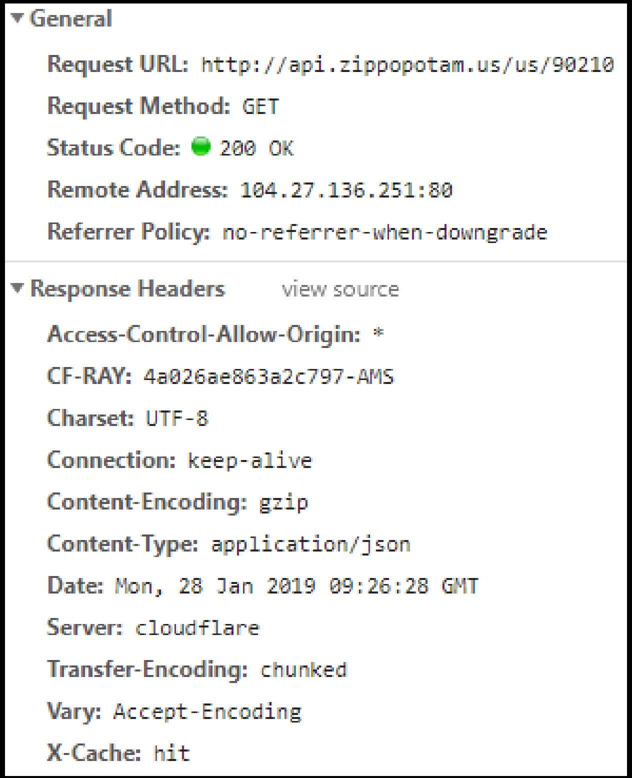
An example

_GET http://api.zippopotam.us/us/90210





```
)",
 States",
ion: "US",


ie: "Beverly Hills",
e: "-118.4065",
California",
oreviation: "CA",
 "34.0901"
```

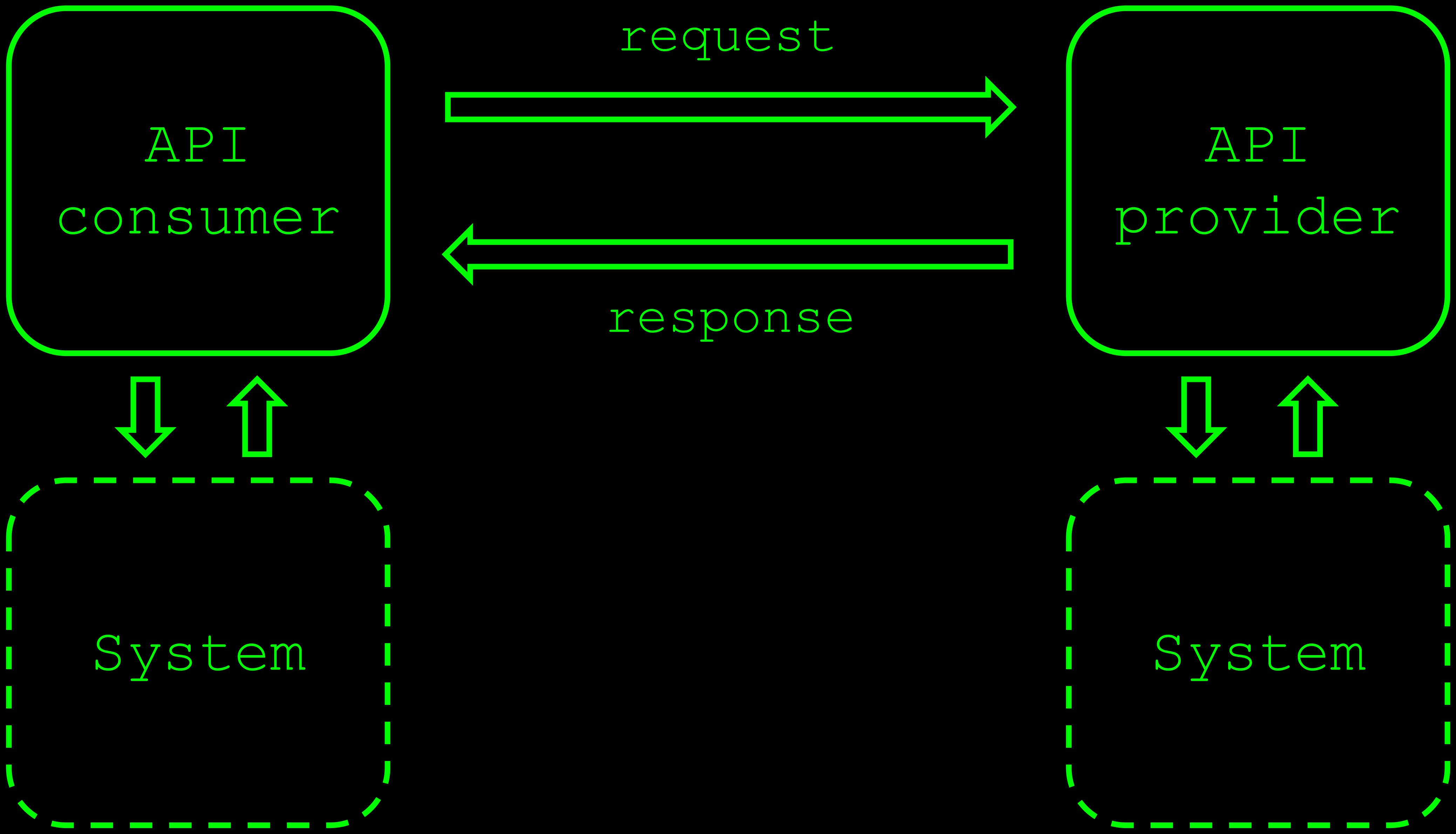


▼ General

Request URL: http://api.zippopotam.us/us/90210
Request Method: GET
Status Code: ● 200 OK
Remote Address: 104.27.136.251:80
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source

Access-Control-Allow-Origin: *
CF-RAY: 4a026ae863a2c797-AMS
Charset: UTF-8
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json
Date: Mon, 28 Jan 2019 09:26:28 GMT
Server: cloudflare
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Cache: hit


request
API consumer
API provider
response
System
System

Supporting operations other than GET

Creating specific responses for edge cases

What might we want to simulate?

Delays, fault status codes, malformatted responses, …

…

WireMock

_http://wiremock.org

_Java

_HTTP mock server

  _only supports HTTP(S)

_open source

  _developed and maintained by Tom Akehurst

Install WireMock

_Maven

```
<dependency>
     <groupId>com.github.tomakehurst</groupId>
     <artifactId>wiremock-jre8</artifactId>
     <version>2.26.3</version>
    <scope>test</test>
</dependency>
```

Starting WireMock

_In Java (via JUnit @Rule)

```java
@Rule
public WireMockRule wireMockRule = new WireMockRule( port: 9876);
```

_In Java (without using JUnit)

```java
WireMockServer wireMockServer =
        new WireMockServer(new WireMockConfiguration().port(9876));

wireMockServer.start();
```

_Standalone

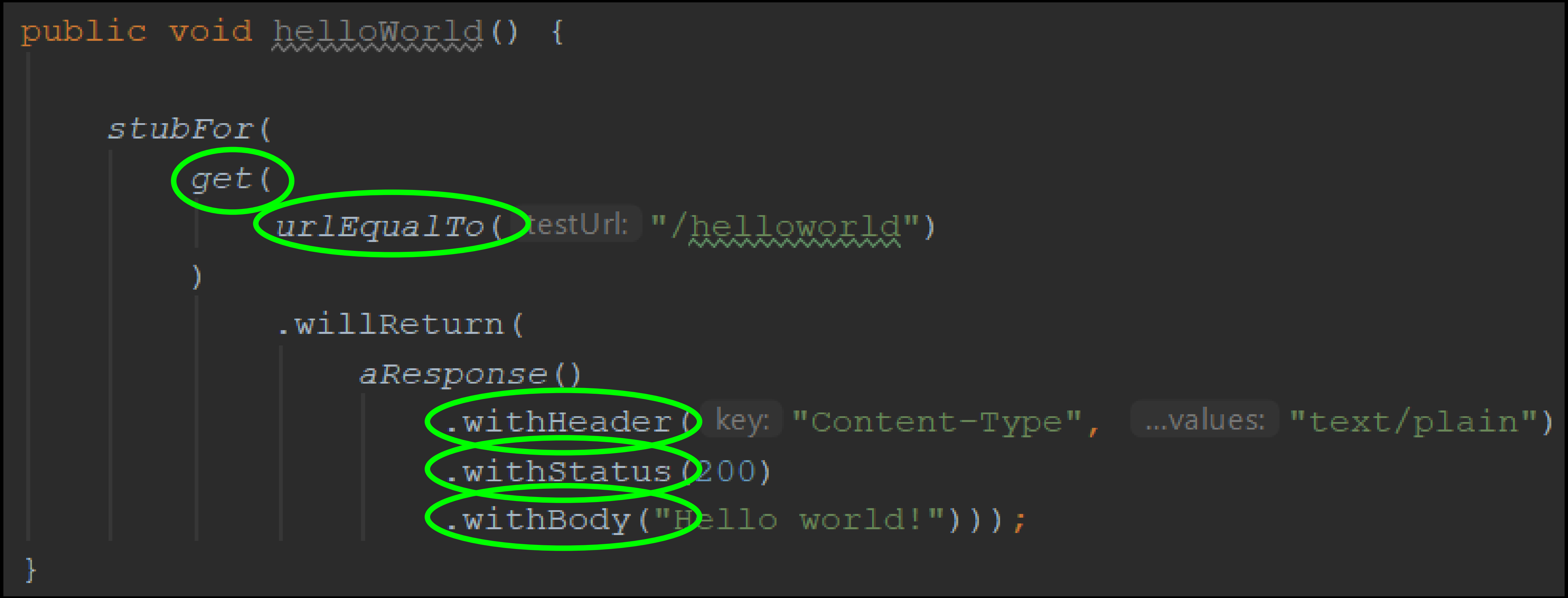
```
java -jar wiremock-standalone-2.26.3.jar --port 9876
```

Configure responses

_In (Java) code

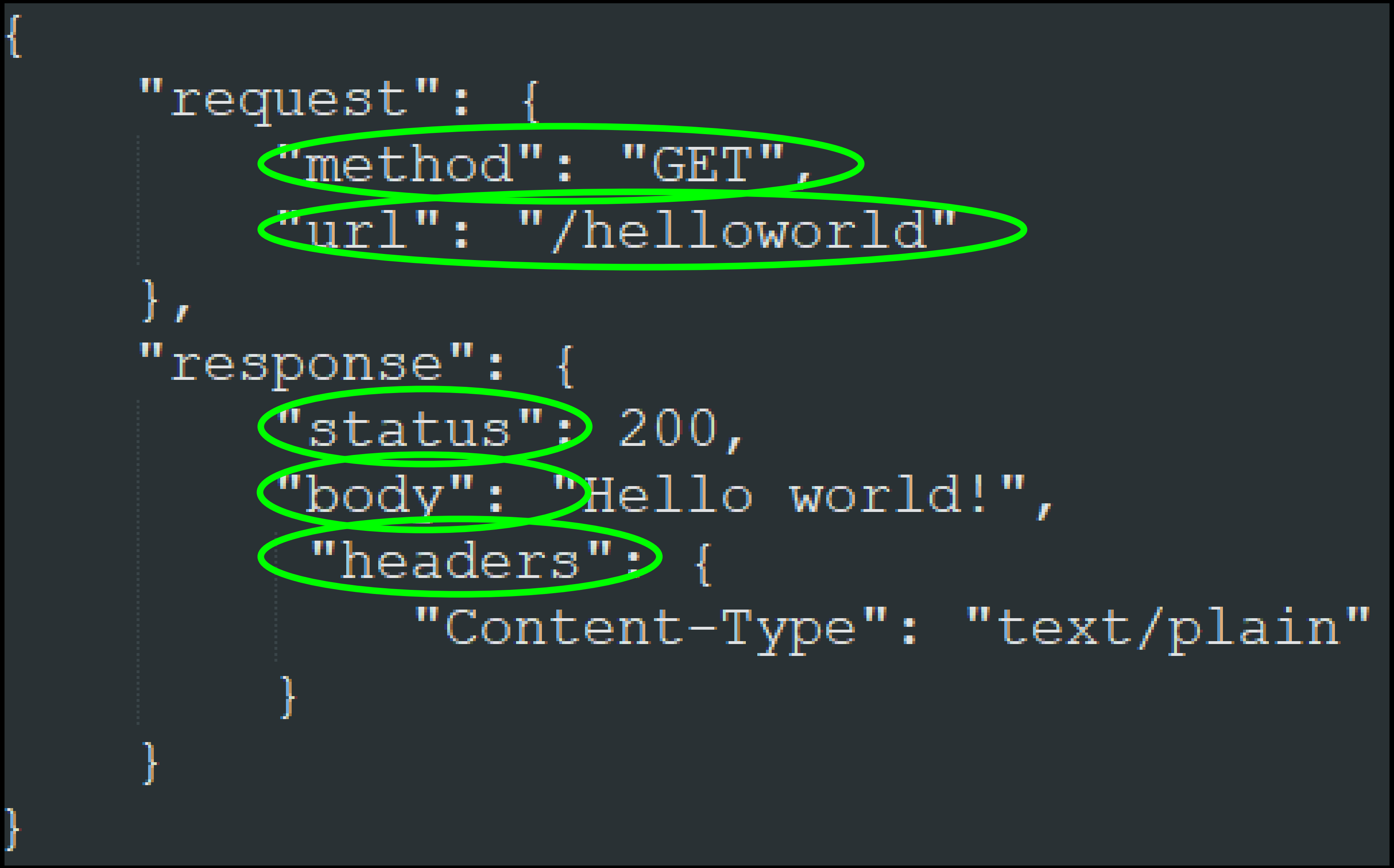
_Using JSON mapping files

An example mock defined in Java



```
public void helloWorld() {

    stubFor(
        get(
            urlEqualTo( testUrl: "/helloworld")
        )
            .willReturn(
                aResponse()
                    .withHeader( key: "Content-Type",  ...values: "text/plain")
                    .withStatus(200)
                    .withBody("Hello world!")));
}
```

The same mock, but now in JSON



```
{
    "request": {
        "method": "GET",
        "url": "/helloworld"
    },
    "response": {
        "status": 200,
        "body": "Hello world!",
        "headers": {
            "Content-Type": "text/plain"
        }
    }
}
```

Useful WireMock features

- _Verification
    - _Verify that certain requests are sent by application under test
- _Record and playback
    - _Generate mocks based on request-response pairs (traffic)
- _Fault simulation
- _…
- _Full documentation at http://wiremock.org/docs/

Now it's your turn!

_src/test/java/exercises/WireMockExercises1.java

_Create a couple of basic mocks
   _You can choose between Java, JSON or do both

_JSON mappings should be placed in
   _src/test/resources/mappings

_Verify your solution by running the tests in the same file

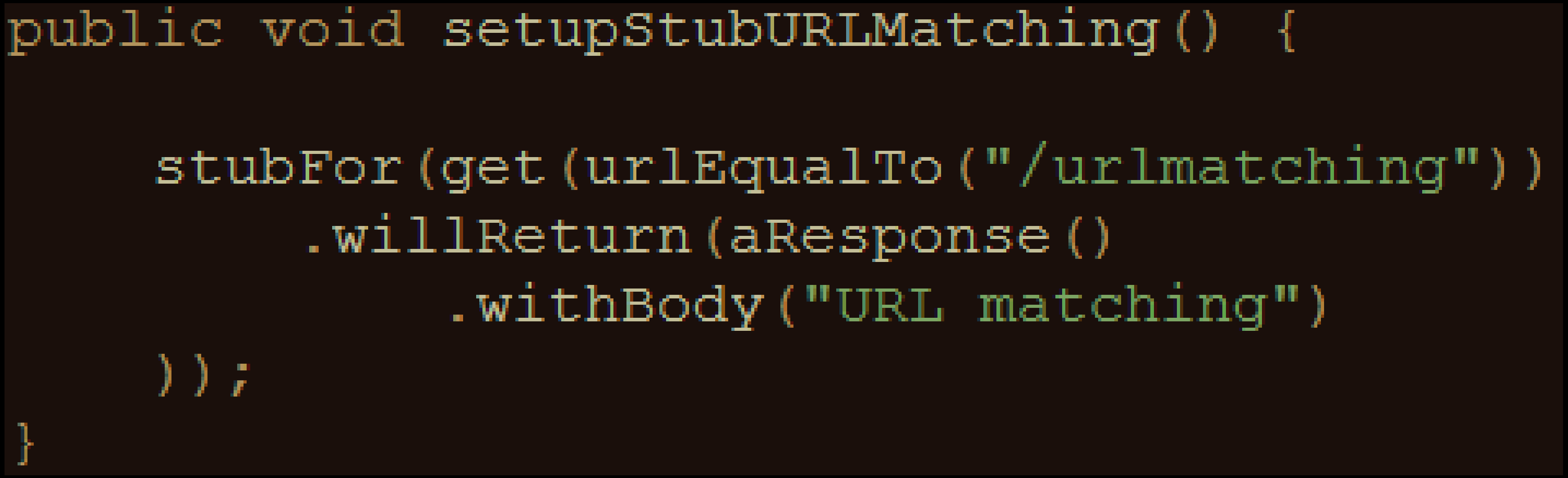
Request matching

_Send a response only when certain properties in the request are matched

_Options for request matching:

- _URL
- _HTTP method
- _Query parameters
- _Headers
- _Request body elements
- _…

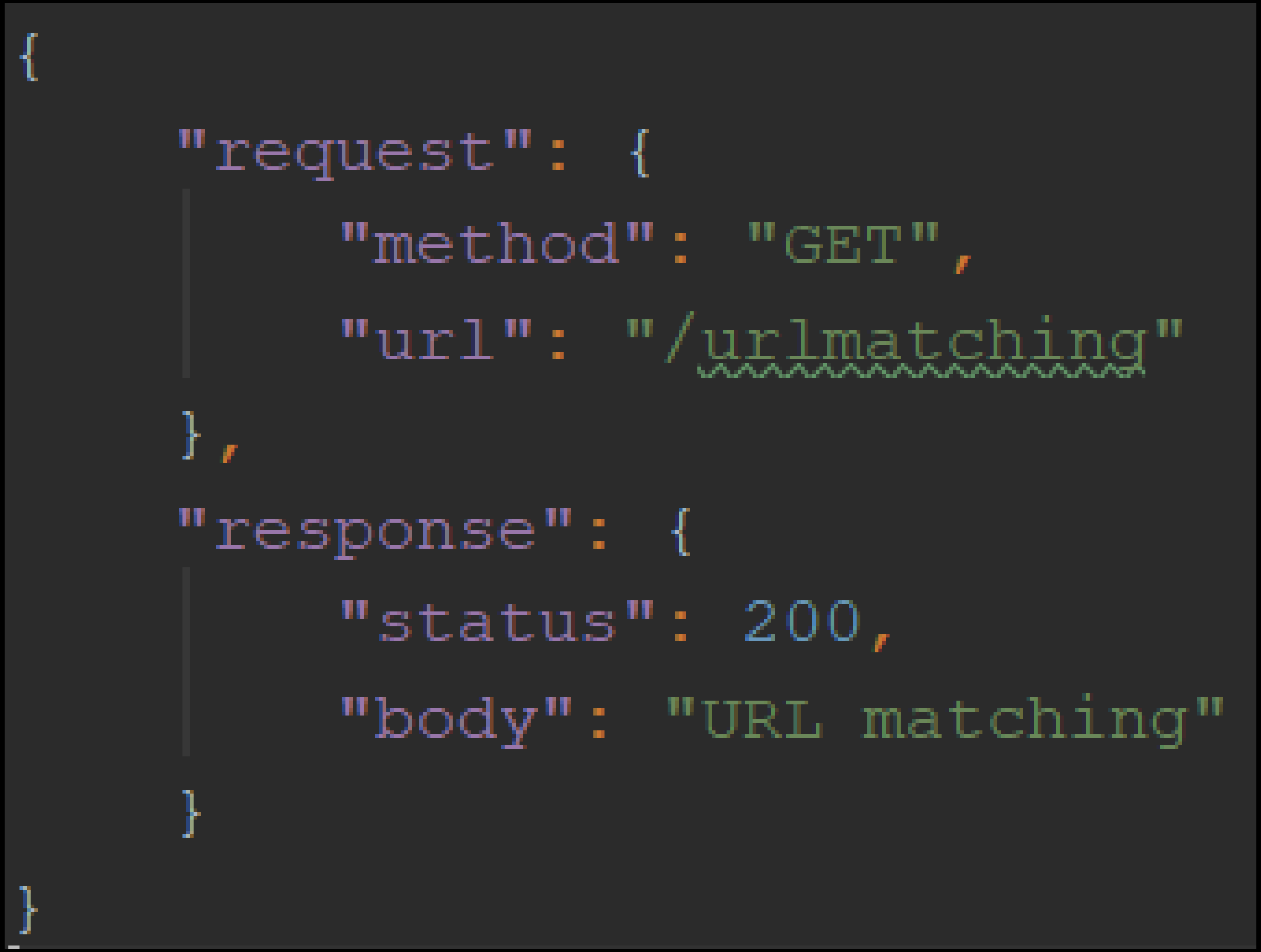
Example: URL matching (Java)



```java
public void setupStubURLMatching() {

    stubFor(get(urlEqualTo("/urlmatching"))
        .willReturn(aResponse()
            .withBody("URL matching")
    ));
}
```

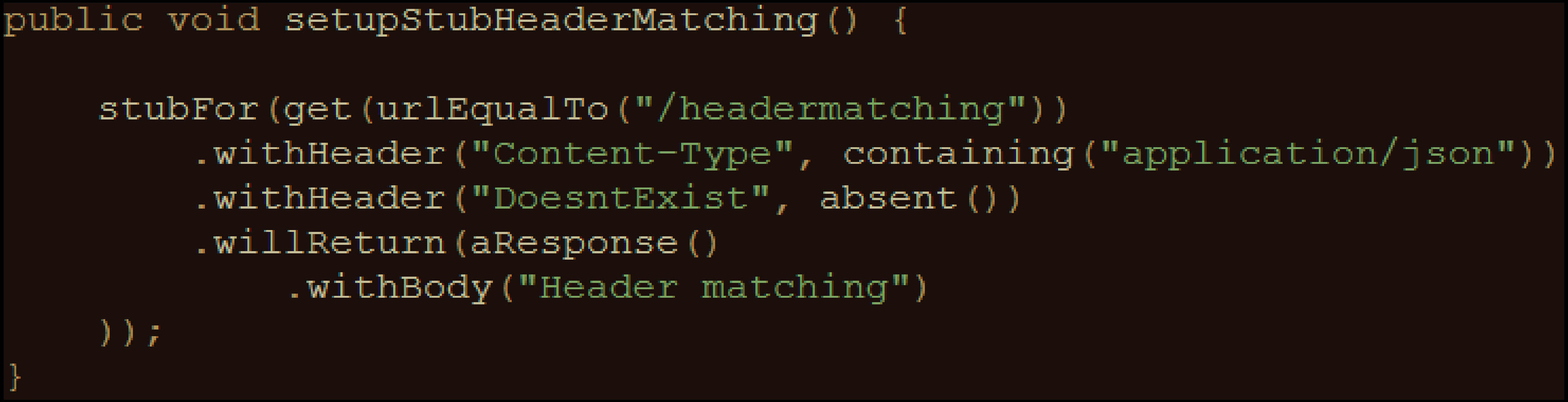
_Other URL options:
  _urlPathEqualTo (using exact values)
  _urlMatching (using regular expressions)
  _urlPathMatching (using regular expressions)

Example: URL matching (JSON)



```
{
    "request": {
        "method": "GET",
        "url": "/urlmatching"
    },
    "response": {
        "status": 200,
        "body": "URL matching"
    }
}
```

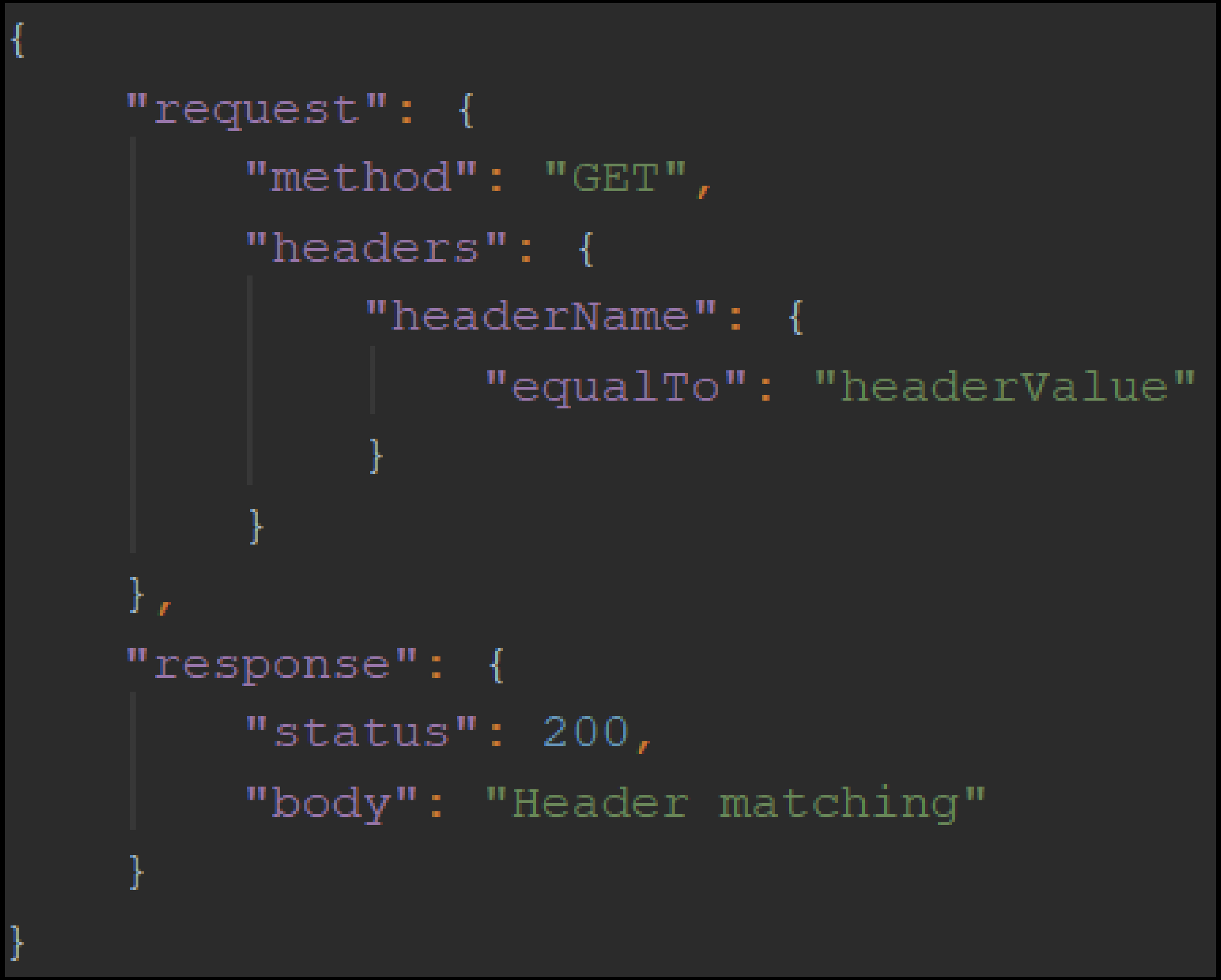
Example: header matching (Java)



```
public void setupStubHeaderMatching() {

    stubFor(get(urlEqualTo("/headermatching"))
        .withHeader("Content-Type", containing("application/json"))
        .withHeader("DoesntExist", absent())
        .willReturn(aResponse()
            .withBody("Header matching")
    ));
}
```

_absent(): check that parameter is not in request

Example: header matching (JSON)



```
{
    "request": {
        "method": "GET",
        "headers": {
            "headerName": {
                "equalTo": "headerValue"
            }
        }
    },
    "response": {
        "status": 200,
        "body": "Header matching"
    }
}
```

Other matching strategies

_Authentication (Basic, OAuth(2))

_Query parameters

_Request body

_Multipart/form-data

_You can write your own matching logic too

Fault simulation

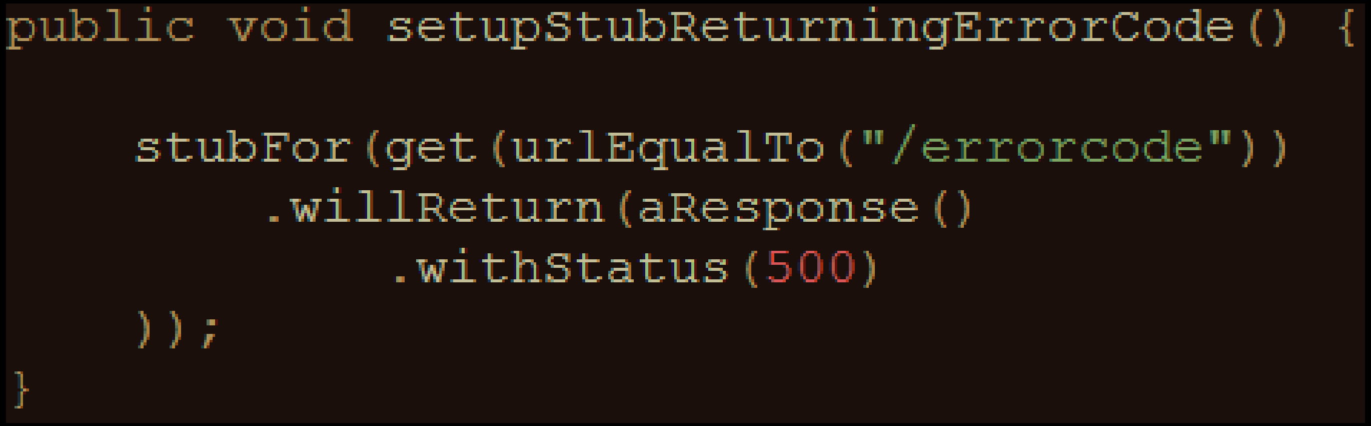
_Extend test coverage by simulating faults

_Often hard to do in real systems

_Easy to do using stubs or mocks

_Used to test the exception handling of your application under test

Example: HTTP status code (Java)

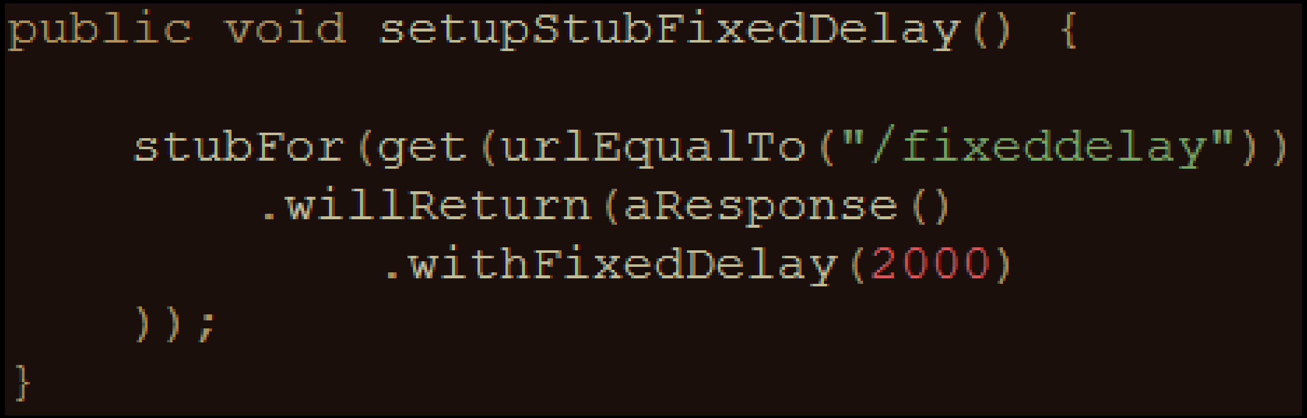


```
public void setupStubReturningErrorCode() {

    stubFor(get(urlEqualTo("/errorcode"))
        .willReturn(aResponse()
            .withStatus(500)
    ));
}
```

_Often used HTTP status codes:

Client error	**Server error**
403 (Forbidden)	500 (Internal server error)
404 (Not found)	503 (Service unavailable)

Example: timeout (Java)

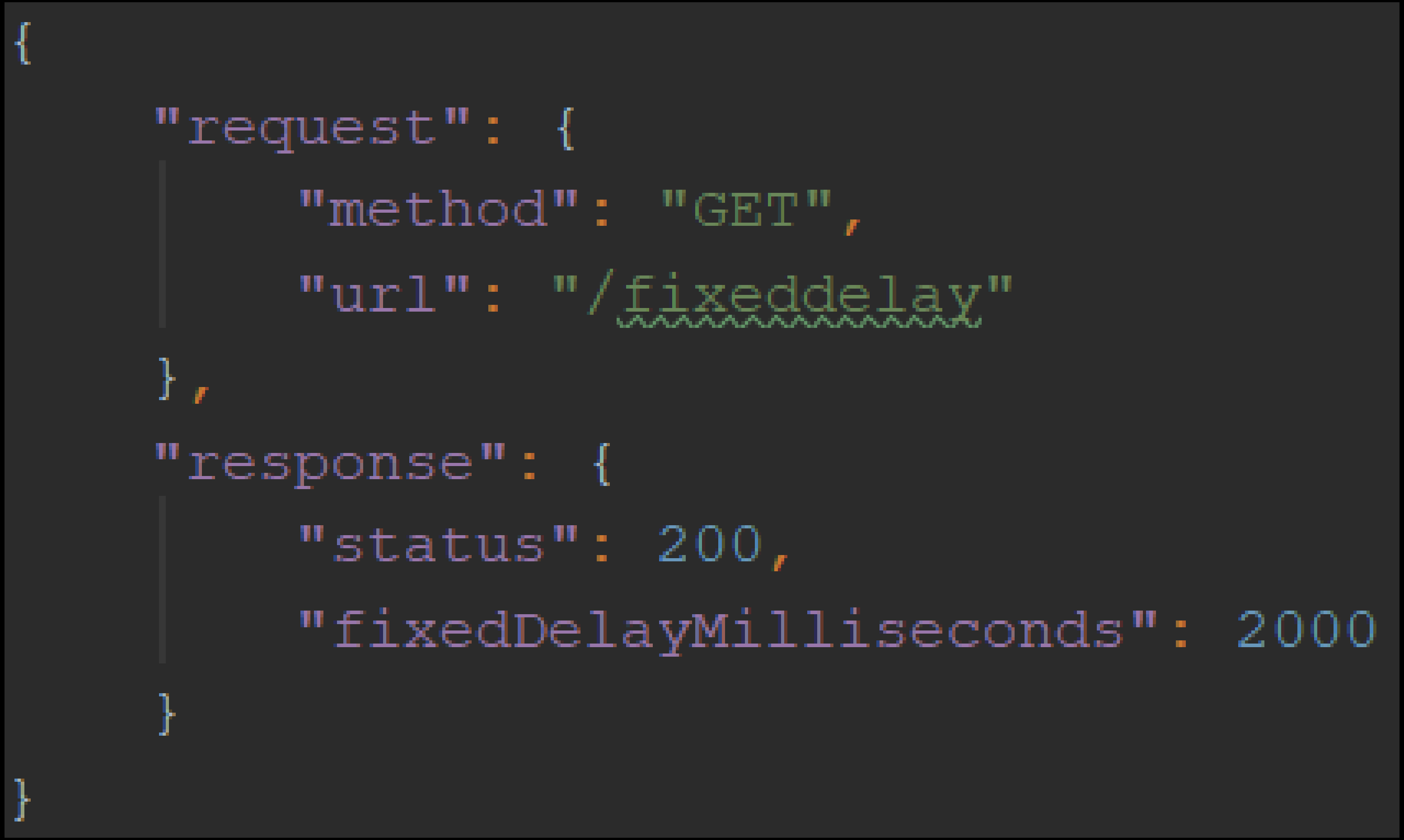


```
public void setupStubFixedDelay() {

    stubFor(get(urlEqualTo("/fixeddelay"))
        .willReturn(aResponse()
            .withFixedDelay(2000)
    ));
}
```

_Random delay can also be used
  _Uniform, lognormal, chunked dribble distribution options

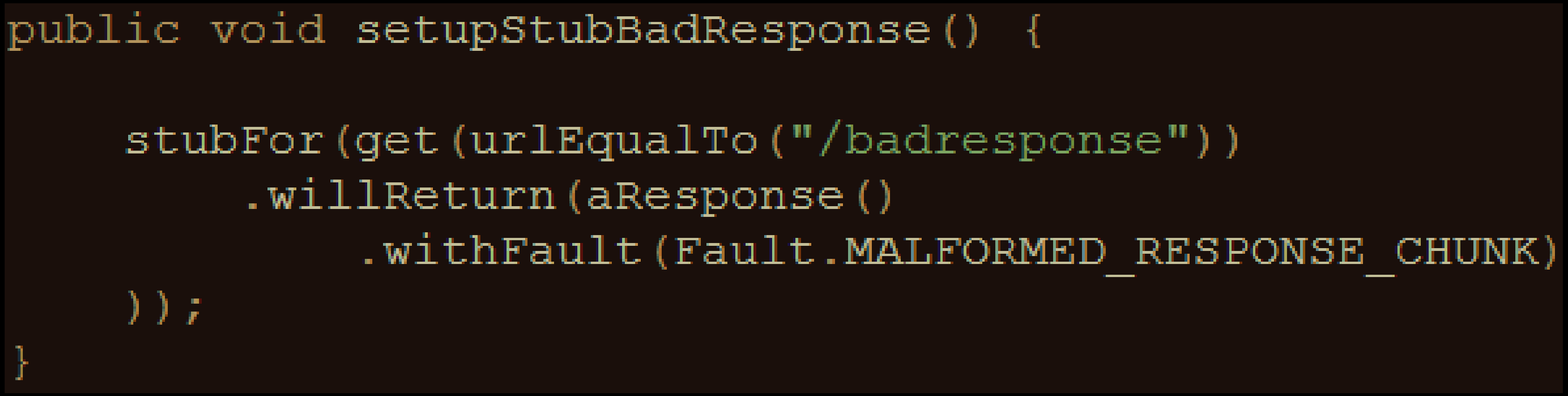
_Can be configured on a per-stub basis as well as globally

Example: timeout (JSON)



```json
{
    "request": {
        "method": "GET",
        "url": "/fixeddelay"
    },
    "response": {
        "status": 200,
        "fixedDelayMilliseconds": 2000
    }
}
```

Example: bad responses (Java)



```
public void setupStubBadResponse() {

    stubFor(get(urlEqualTo("/badresponse"))
        .willReturn(aResponse()
            .withFault(Fault.MALFORMED_RESPONSE_CHUNK)
    ));
}
```

_HTTP status code 200, but garbage in response body

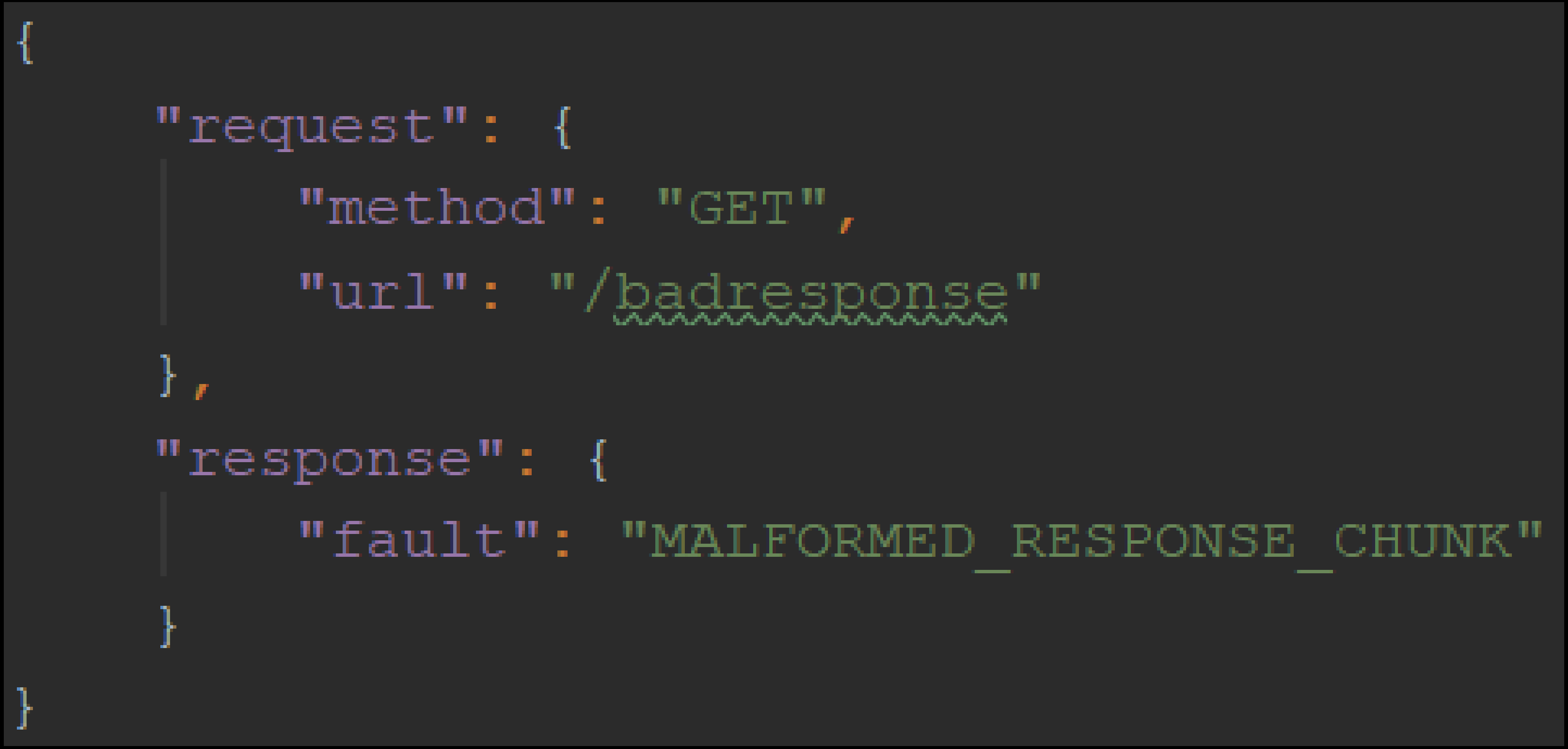
_Other options:

   _RANDOM_DATA_THEN_CLOSE (as above, without HTTP 200)

   _EMPTY_RESPONSE (does what it says on the tin)

   _CONNECTION_RESET_BY_PEER (close connection, no response)

Example: bad responses (JSON)



```
{
    "request": {
        "method": "GET",
        "url": "/badresponse"
    },
    "response": {
        "fault": "MALFORMED_RESPONSE_CHUNK"
    }
}
```

Now it's your turn!

_src/test/java/wiremockexercises/WireMockExercises2.java

_Create mocks that simulate edge / error cases
- _You can choose between Java, JSON or do both
- _Use the appropriate request matcher strategy

_Verify your solution by running the tests

Statefulness

_Sometimes, you want to simulate stateful behaviour

_Shopping cart (empty / full)

_Database (data present / not present)

_Order in which requests arrive is significant

Stateful mocks in WireMock

_Supported through the concept of a Scenario

_Essentially a finite state machine (FSM)

  _States and state transitions

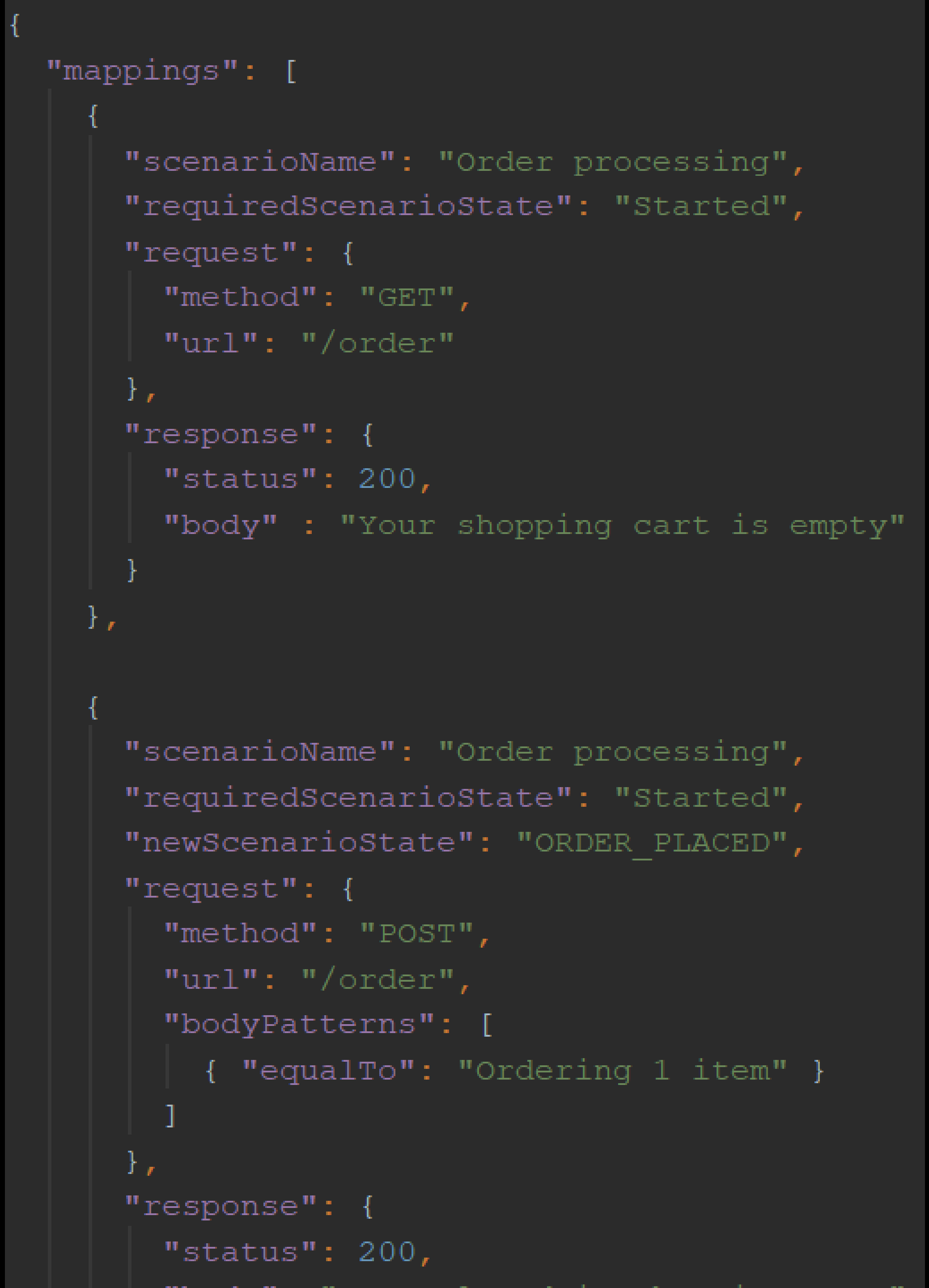
_Combination of current state and incoming request determines the response being sent

  _Before now, it was only the incoming request

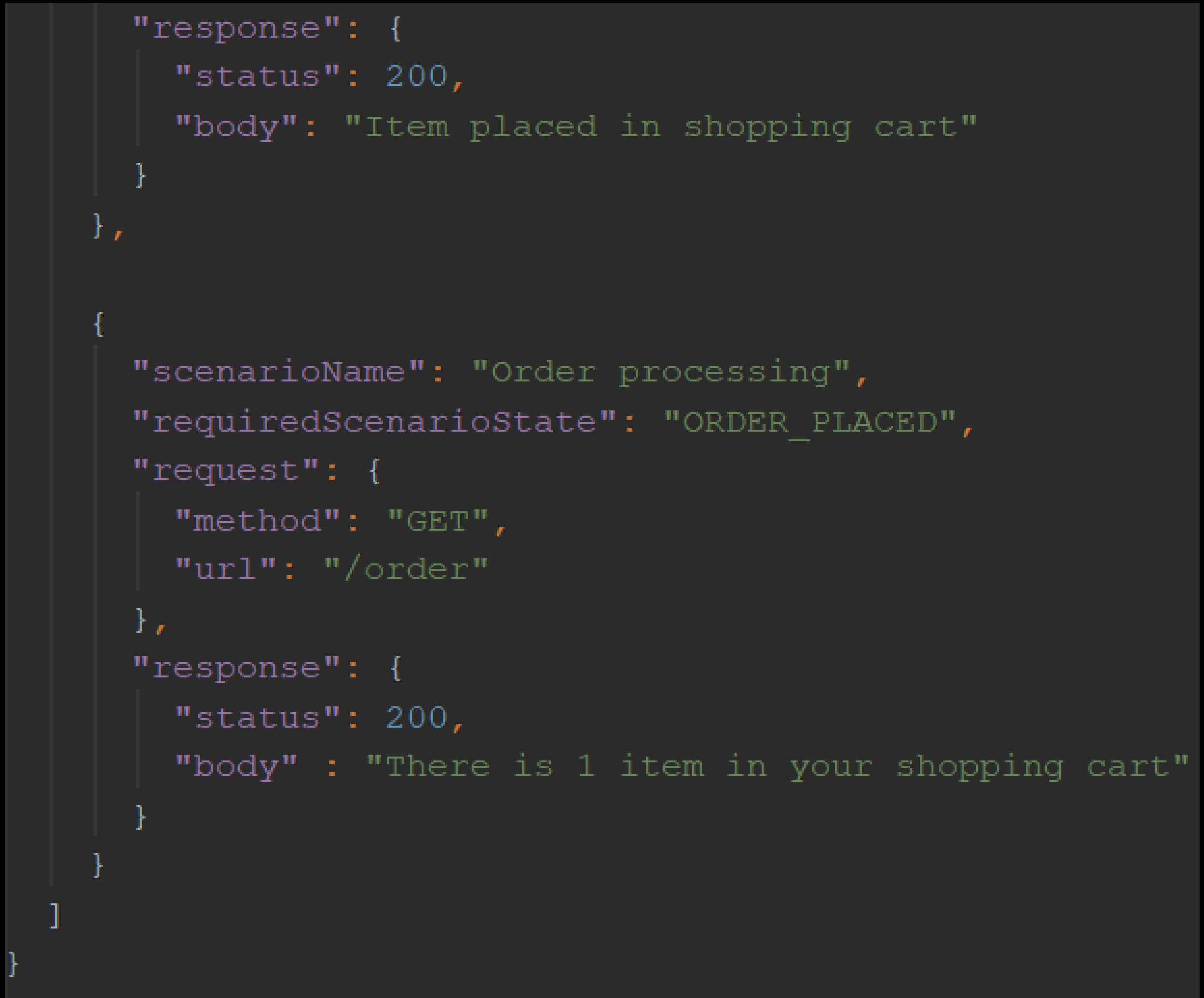
Stateful mocks: an example (Java)

```java
public void setupStubStateful() {

    stubFor(get(urlEqualTo("/order")).inScenario("Order processing")
        .whenScenarioStateIs(Scenario.STARTED)
        .willReturn(aResponse()
            .withBody("Your shopping cart is empty")
    ));

    stubFor(post(urlEqualTo("/order")).inScenario("Order processing")
        .whenScenarioStateIs(Scenario.STARTED)
        .withRequestBody(equalTo("Ordering 1 item"))
        .willReturn(aResponse()
            .withBody("Item placed in shopping cart"))
        .willSetStateTo("ORDER_PLACED")
    );

    stubFor(get(urlEqualTo("/order")).inScenario("Order processing")
        .whenScenarioStateIs("ORDER_PLACED")
        .willReturn(aResponse()
            .withBody("There is 1 item in your shopping cart")
    ));
}
```

Stateful mocks: an example (JSON)



```
{
  "mappings": [
    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "Started",
      "request": {
        "method": "GET",
        "url": "/order"
      },
      "response": {
        "status": 200,
        "body" : "Your shopping cart is empty"
      }
    },

    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "Started",
      "newScenarioState": "ORDER_PLACED",
      "request": {
        "method": "POST",
        "url": "/order",
        "bodyPatterns": [
          { "equalTo": "Ordering 1 item" }
        ]
      },
      "response": {
        "status": 200,
```



```
      "response": {
        "status": 200,
        "body": "Item placed in shopping cart"
      }
    },

    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "ORDER_PLACED",
      "request": {
        "method": "GET",
        "url": "/order"
      },
      "response": {
        "status": 200,
        "body" : "There is 1 item in your shopping cart"
      }
    }
  ]
}
```

Now it’s your turn!

_src/test/java/wiremockexercises/WireMockExercises3.java

_Create a stateful mock that exerts the described behaviour

  _You can choose between Java, JSON or do both

_Verify your solution by running the tests

Response templating

_Often, you want to reuse elements from the request in the response

  _Request ID header

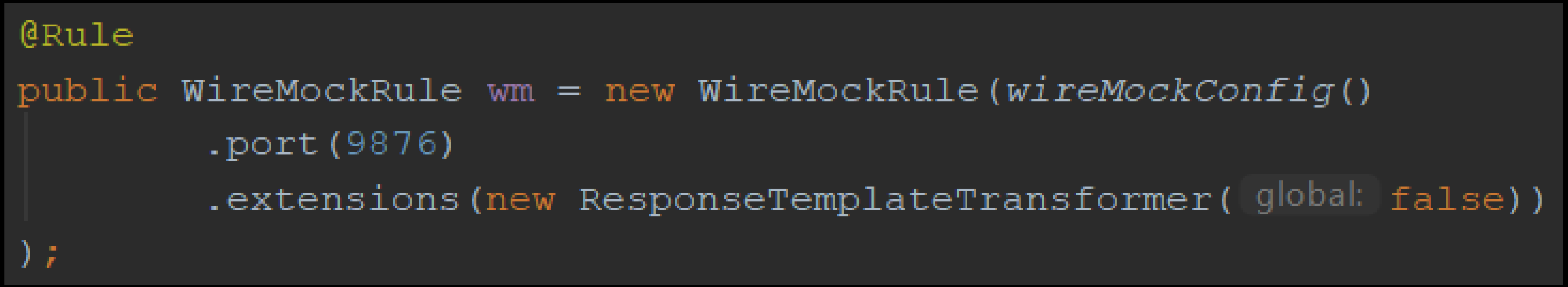
  _Unique body elements (client ID, etc.)

  _Cookie values

_WireMock supports this through response templating

Setup response templating

_In code: through the JUnit rule

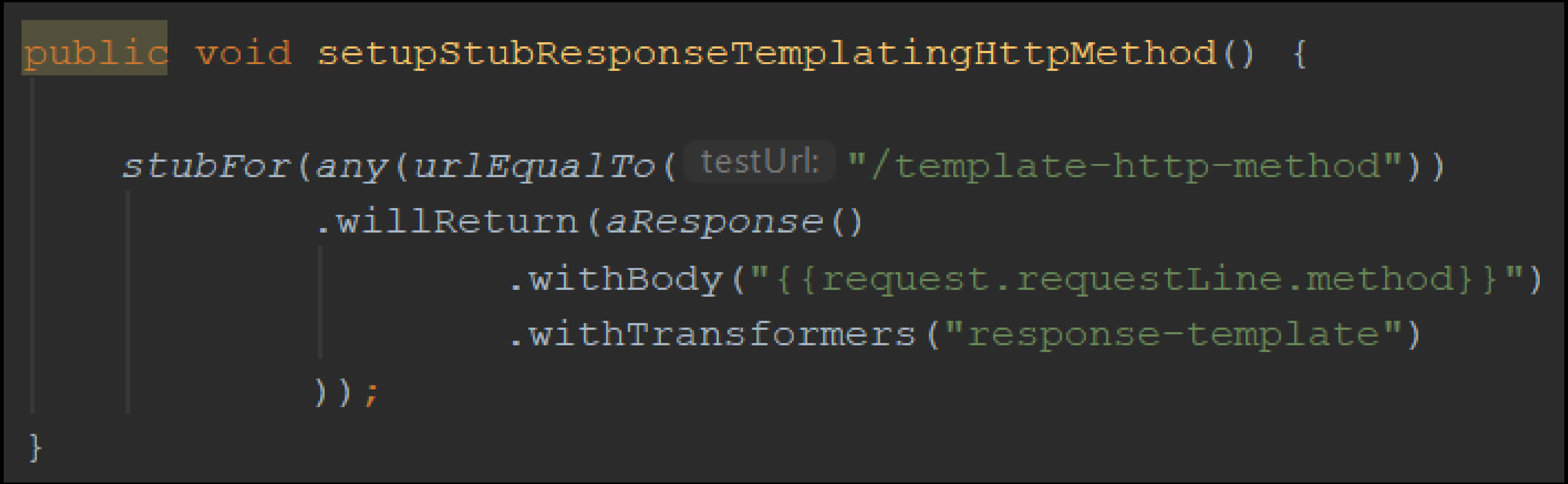


```
@Rule
public WireMockRule wm = new WireMockRule(wireMockConfig()
        .port(9876)
        .extensions(new ResponseTemplateTransformer( global: false))
);
```

_Global == false: response templating transformer has to be enabled for individual stubs

Enable/apply response templating

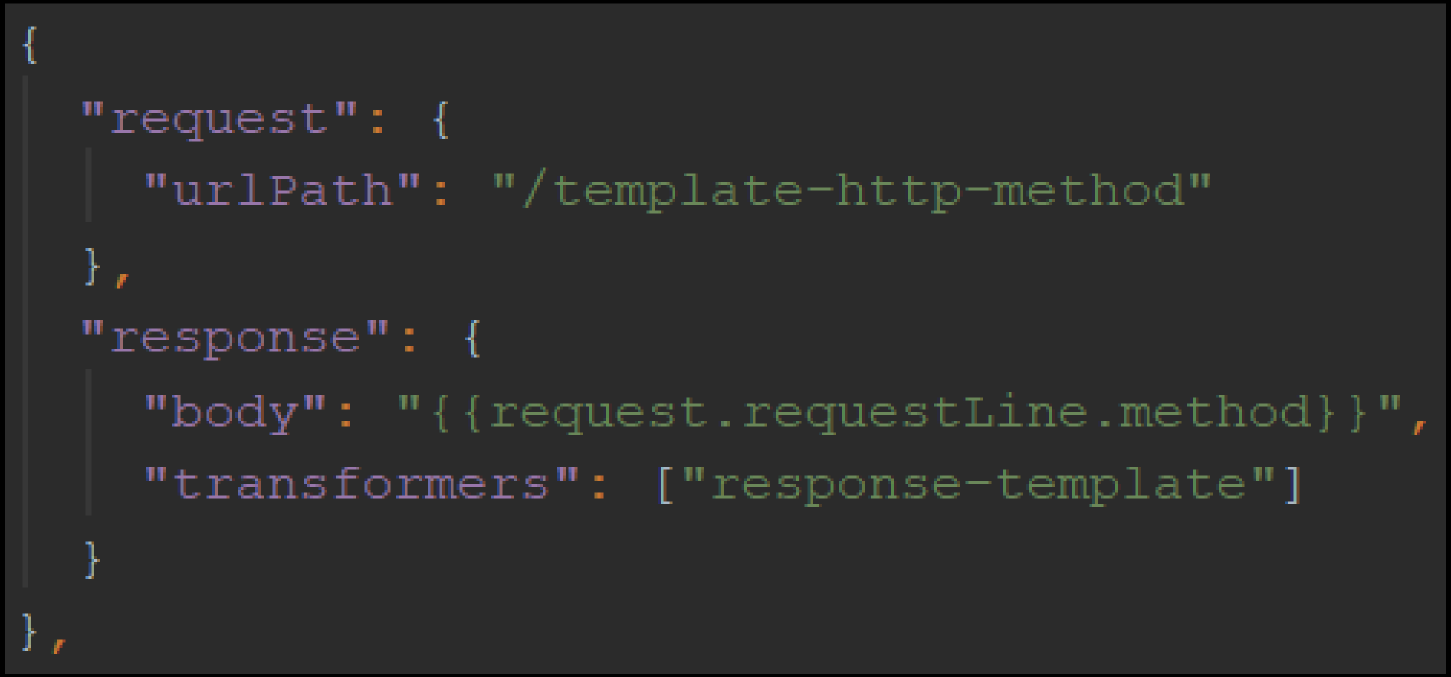
_This template reads the HTTP request method (GET/POST/PUT/…) and returns it as the response body



```
public void setupStubResponseTemplatingHttpMethod() {

    stubFor(any(urlEqualTo( testUrl: "/template-http-method"))
            .willReturn(aResponse()
                    .withBody("{{request.requestLine.method}}")
                    .withTransformers("response-template")
            ));
}
```

Enable/apply response templating

_This template reads the HTTP request method (GET/POST/PUT/…) and returns it as the response body



```
{
  "request": {
    "urlPath": "/template-http-method"
  },
  "response": {
    "body": "{{request.requestLine.method}}",
    "transformers": ["response-template"]
  }
},
```

Request attributes

- Many different request attributes available for use
  - request.requestLine.method : HTTP method (example)
  - request.requestLine.path.[<n>] : nth path segment
  - request.requestLine.scheme : protocol (e.g. HTTPS)
  - …

- All available attributes listed at

http://wiremock.org/docs/response-templating/

Request attributes (cont’d)

_Extracting and reusing body elements

_In case of a JSON request body:

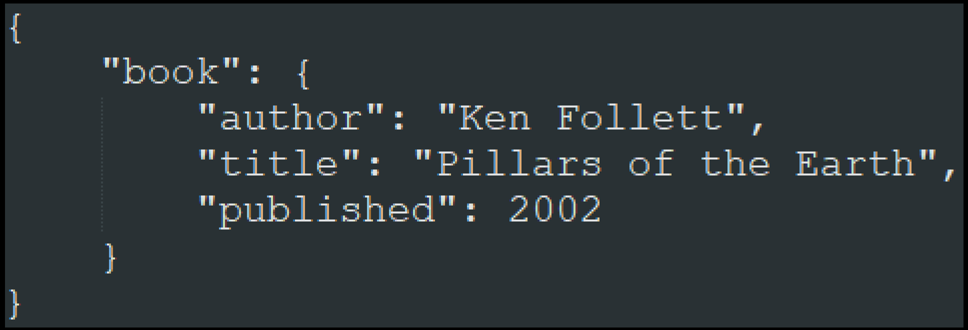
{{jsonPath request.body '$.path.to.element’}}

_In case of an XML request body:

{{xPath request.body ’/path/to/element/text()’}}

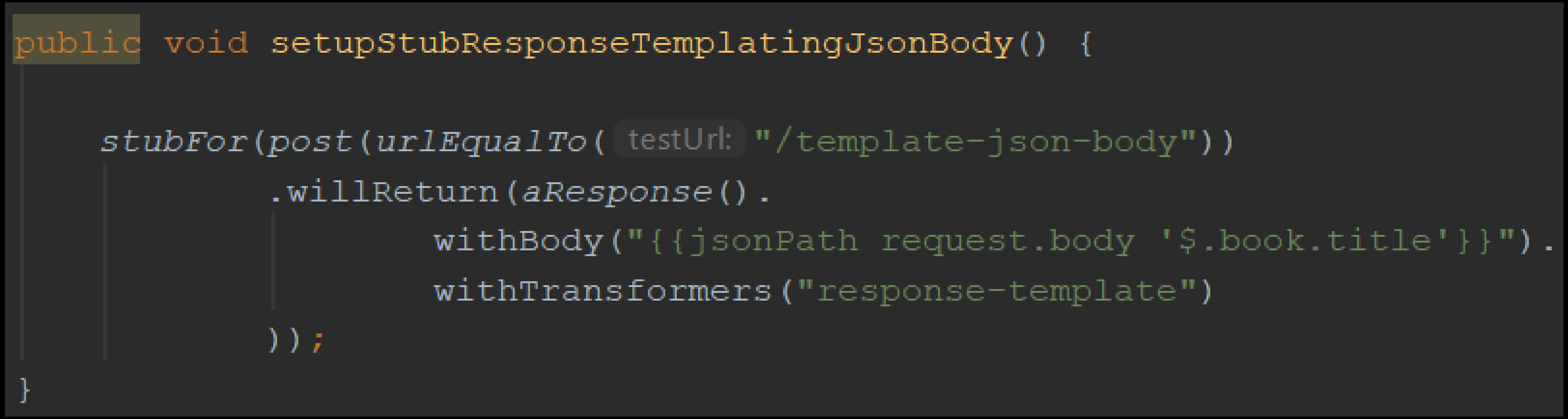
JSON extraction example

_When sent this JSON request body:



```
{
    "book": {
        "author": "Ken Follett",
        "title": "Pillars of the Earth",
        "published": 2002
    }
}
```

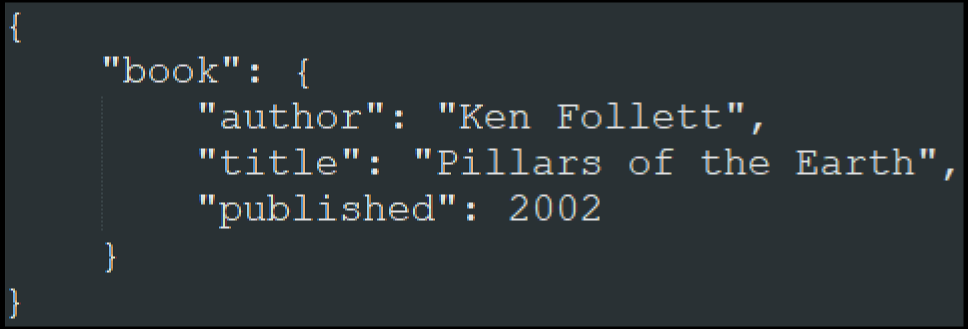
_This stub returns a response with body “Pillars of the Earth”:



```
public void setupStubResponseTemplatingJsonBody() {

    stubFor(post(urlEqualTo( testUrl: "/template-json-body"))
            .willReturn(aResponse().
                    withBody("{{jsonPath request.body '$.book.title'}}").
                    withTransformers("response-template")
            ));
}
```

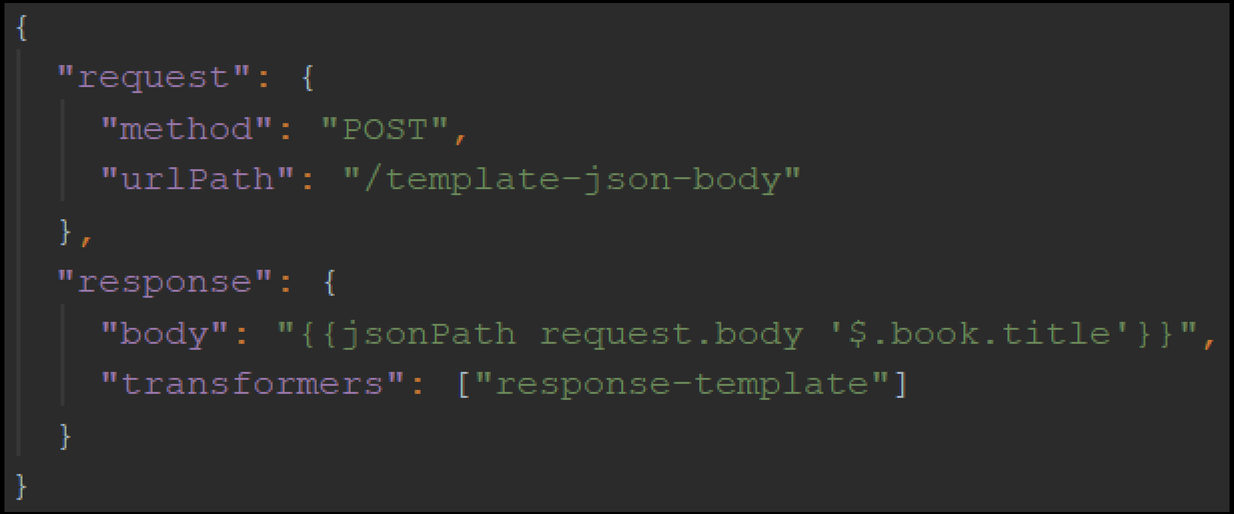
JSON extraction example

_When sent this JSON request body:



```
{
    "book": {
        "author": "Ken Follett",
        "title": "Pillars of the Earth",
        "published": 2002
    }
}
```

_This stub returns a response with body “Pillars of the Earth”:



```
{
  "request": {
    "method": "POST",
    "urlPath": "/template-json-body"
  },
  "response": {
    "body": "{{jsonPath request.body '$.book.title'}}",
    "transformers": ["response-template"]
  }
}
```

Now it’s your turn!

_src/test/java/wiremockexercises/
WireMockExercises4.java

_Create mocks that use response templating
 _You can choose between Java, JSON or do both

_Verify your solution by running the tests

?