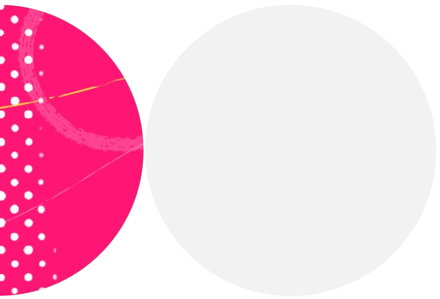
REST Assured Fundamentals

Introducing REST API Testing



REST Assured Fundamentals

Version Check

Version Check

This course was created by using:

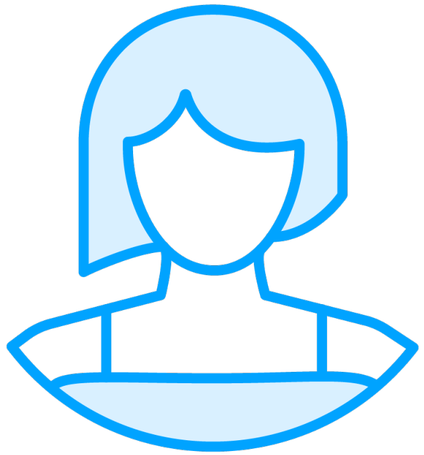
- REST Assured 5.3
- Java 17

Version Check

This course is 100% applicable to:

- REST Assured 5.x
- Java 17

Why learn REST Assured?



Why should I learn it?

To gain API testing skills

Is there demand?

Yes!

Speed and efficiency matter!

Which tool should we use?

GUI-based

Code-based

Advantages of GUI Tools

Lower learning curve

Works out-of-the-box

Suitable for both simple and complex APIs

But I also need it do X!

GUI Tool:

has 100+ features,
but not what you need

Code-based Solutions

REST Assured

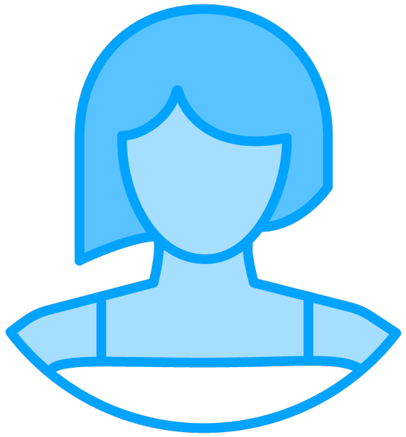
- Has a Fluent Interface

Plain HTTP Client (library or native)

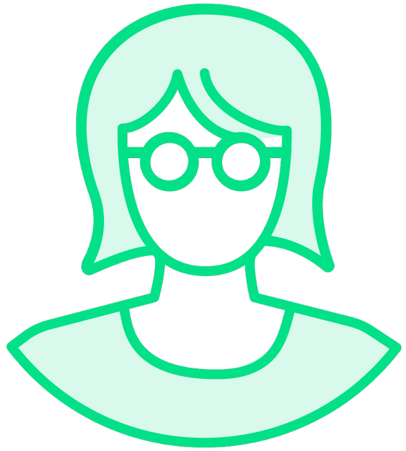
REST Assured Fluent Interface

```
RestAssured.get(“api/url”)
    .then()
    .assertThat()
        .statusCode(200)
    .and()
    .contentType(ContentType.JSON);
```

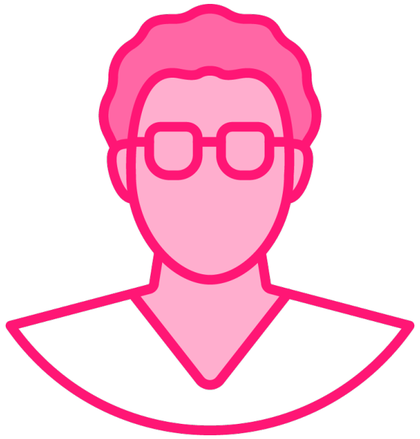
Who This Course Is For



Technical Testers



Automation Engineers



Developers

Prerequisites

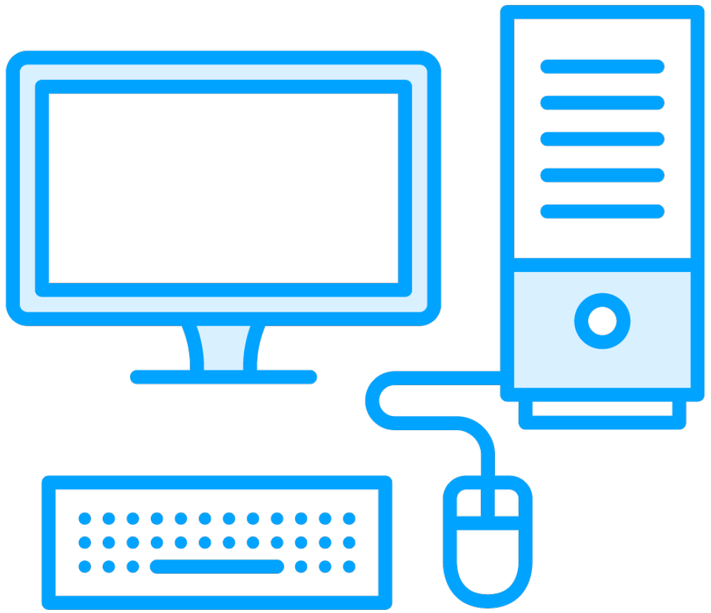
6+ months of working with Java

IDE: e.g. IntelliJ or Eclipse

Test Runner: JUnit or TestNG

Basics of the HTTP protocol

Tech Used



IntelliJ

TestNG

Maven

HTTP Refresher

Client:

- Browser
- Postman
- Java Program

Endpoint

https://api.github.com
https://api.github.com/users/{user}
https://api.github.com/users/{user}/repos

Response

Header (metadata)

e.g.
Status 200 OK
Status 404 Not Found

Body with data (payload)

Typically JSON

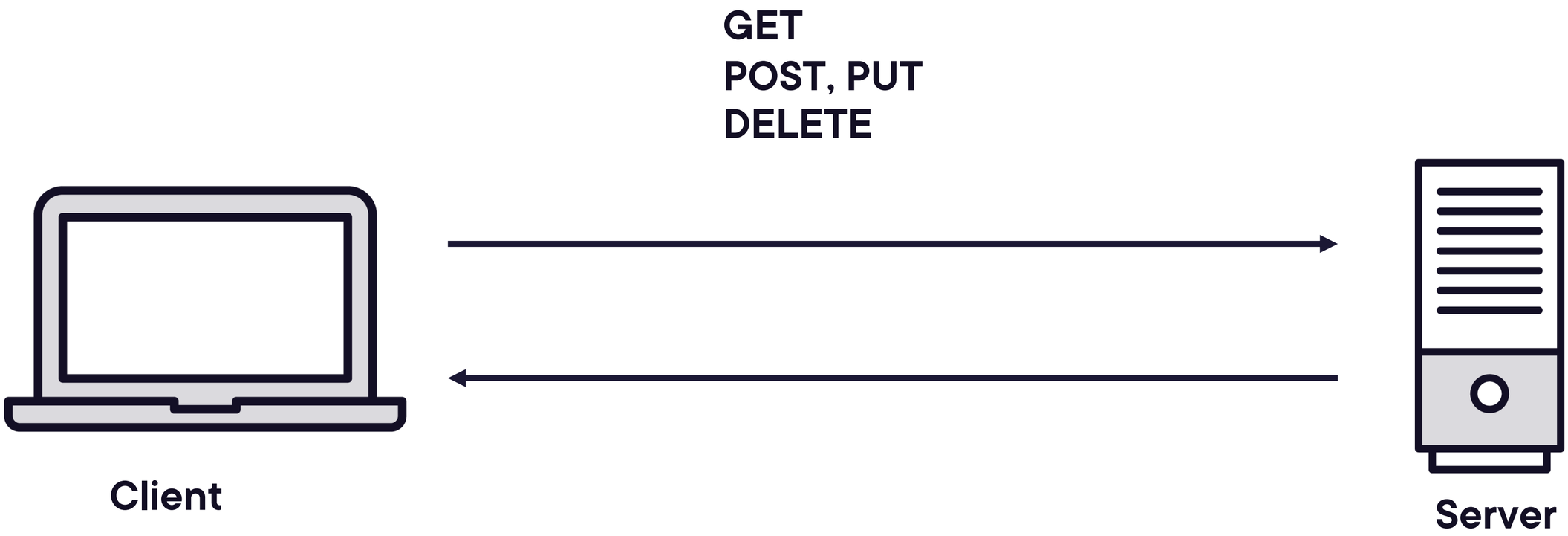
XML

```
<root>
  <login>somename</login>
  <id>12345</id>
  <followers>14</followers>
</root>
```

JSON

```
{
  login: "somename",
  id: 12345,
  followers: 14,
}
```

Other widespread HTTP methods:



Web API vs. REST API

Web API

A more general term

An API that understands HTTP

VS

REST API

A Web API...

Designed according to RESTful principles

RESTful principles

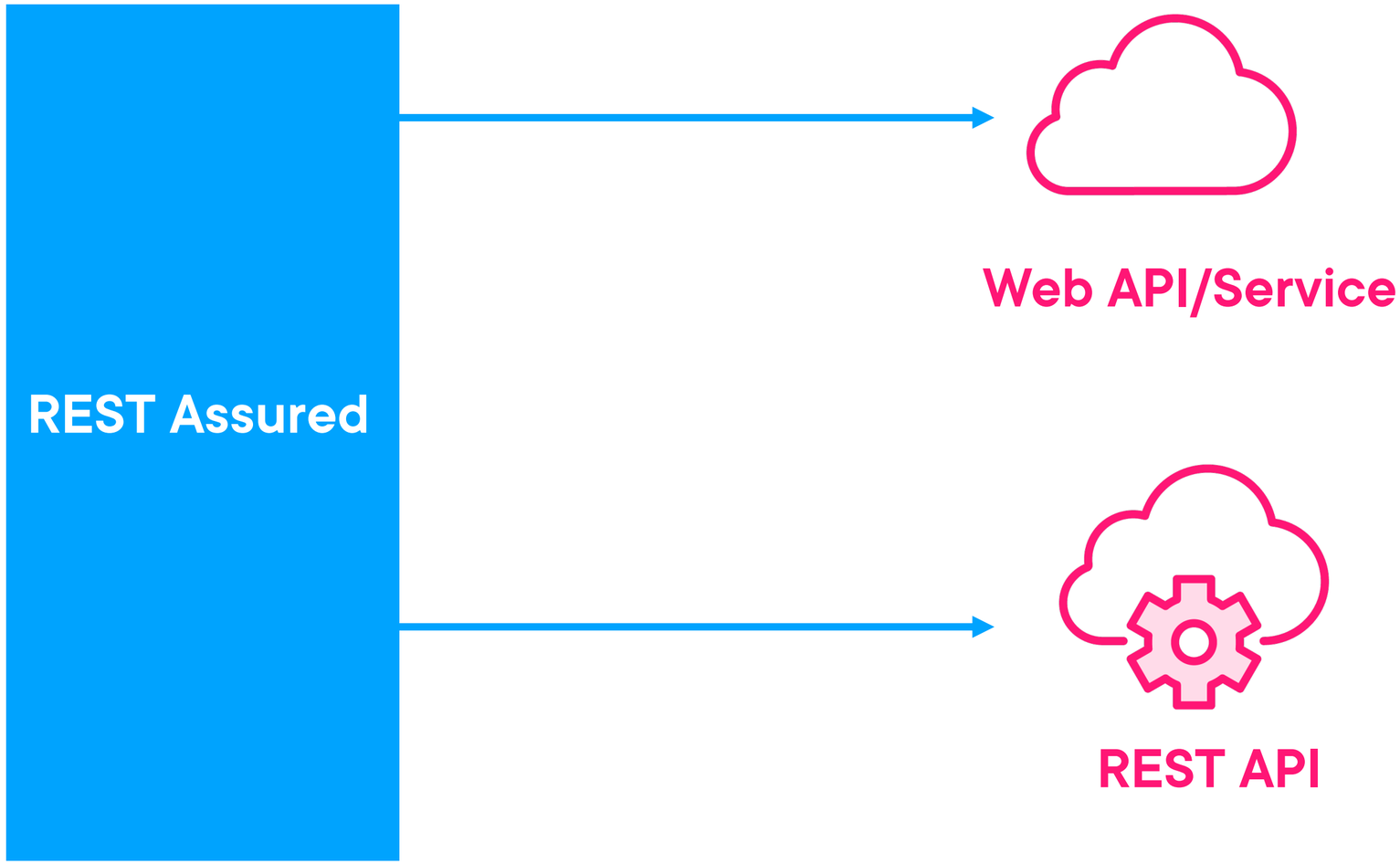
Not a protocol or a standard

Doesn’t mean we must use JSON

RESTful principles

Architectural constraints and principles:

- Uniform Interface
- Statelessness
- Layered System
- Cacheability


REST Assured
Web API/Service
REST API

REST API

A Web API...

Designed according to RESTful principles

VS

SOAP

Simple Object Access Protocol

XML-based

Imposes rules

ACID

Older

XML format

SOAP

guidelines

rules

enterprise

newer

RESTful

lightweight

usually JSON

Demo APIs

https://api.github.com/

https://jsonplaceholder.typicode.com/

https://reqres.in/

Summary

Setup

GET: Validating Response headers

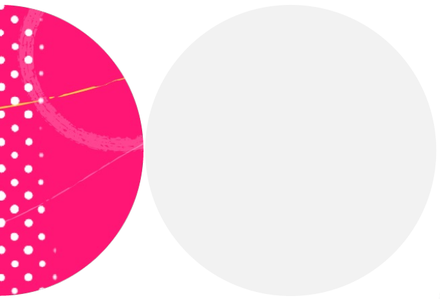
GET: Validating Response body

Other HTTP methods

Config and specifications

Clean and maintainable tests

Validating Response Headers



Overview

- Project setup
- Typical headers
- Assert and validate various headers
- Leverage the Hamcrest library
- Maps of headers

```
status: 200 OK
content-type: application/json; charset=utf-8
date: Mon, 05 June 2023 16:11:35 GMT
etag: "8ccd6372bcf27b234e53e90d8e0ae9c12efd55c887b556b"
server: GitHub.com
x-frame-options: deny
x-ratelimit-limit: 60
[…]
```

“X” prefix is used for custom headers

Header Groups

1

Standard and ubiquitous

e.g. “content-type”

2

Standard

e.g. “date” or “etag”

3

Custom

prefixed with “x-”

```
.getStatusCode()

.getContentType()

.getHeader(String header)

.getHeaders()
```

Response

assertEquals

(r.getStatusCode(), 200);

ValidatableResponse

.assertThat()
.statusCode(200)

Further Checks

Numbers and dates

String contains X

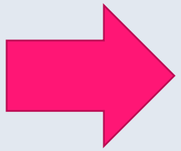
Collection contains Y

Is empty or is null

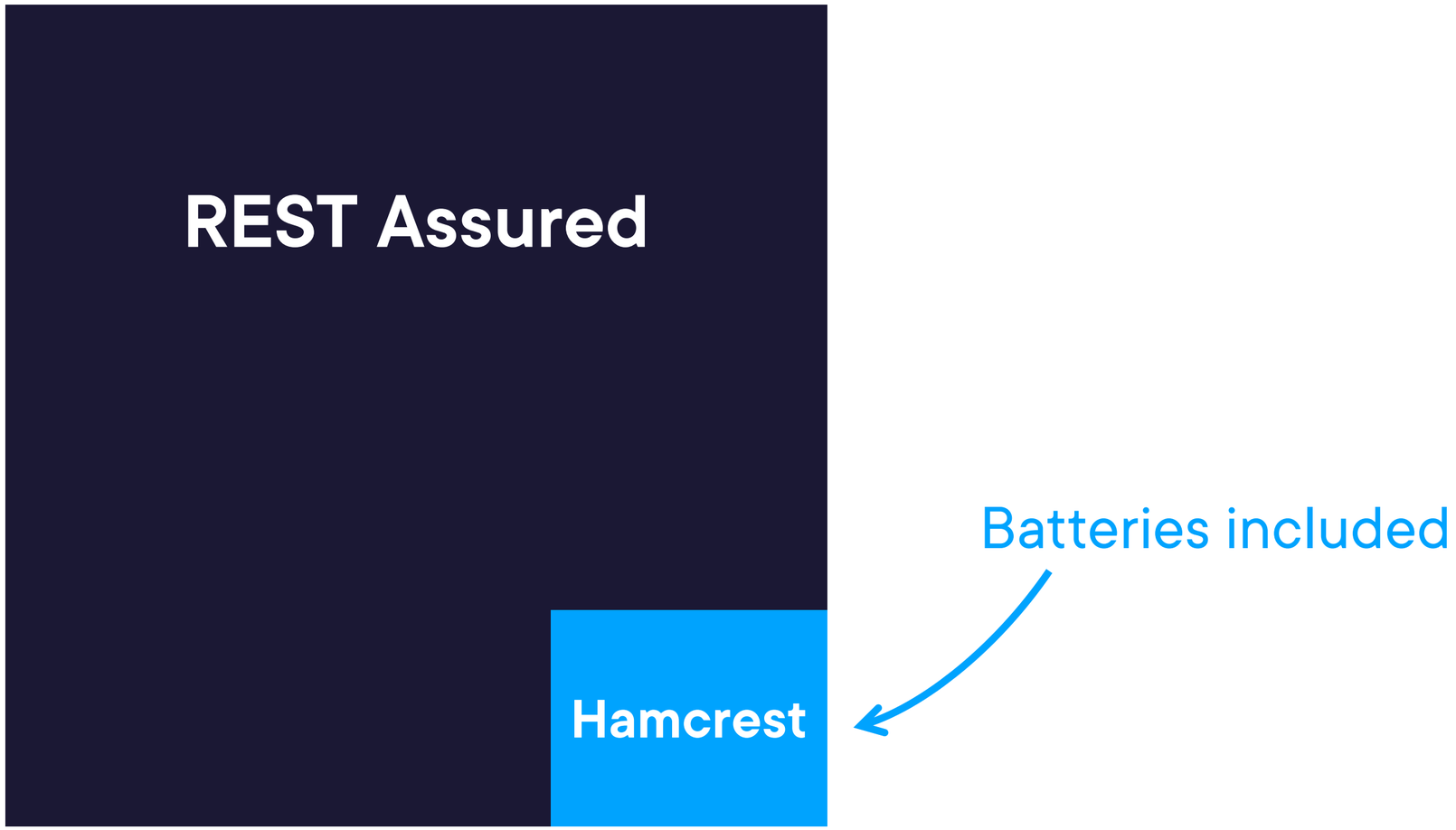
```
.assertTrue()

.assertFalse()

.assertNotNull()
```

```
.assertTrue(resp.getThis() > 199)

.assertFalse(resp.getThat().contains(“x”))

.assertNotNull(resp.getOther())
```

```
.statusCode(int code)

.header(“x”, “y”)
```



```
.statusCode(int code, Matcher m)

.header(“x”, “y”, Matcher m)
```


REST Assured
Hamcrest
Batteries included

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;


@Test
public void hamcrestExample() {

    assertThat(10, lessThan(11));
}
```

```
import static org.hamcrest.Matchers.*;

RestAssured.get(url)
    .then()
    .statusCode(200)
    .statusCode(equalTo(200))
    .statusCode(lessThan(300))
    .header(“x”, containsStringIgnoringCase(“y”))
```

```
import static org.hamcrest.Matchers.*;

RestAssured.get(url)
    .then()
    .header("etag", notNullValue())
    .header("etag", not(emptyString()))
    .statusCode(anyOf(equalTo(200), equalTo(202)))
```

```
.headers(String h1, Object v1, Object... headers)

.headers(Map<String, ?> headers)
```

Summary

Simple response

ValidatableResponse

- Use then()

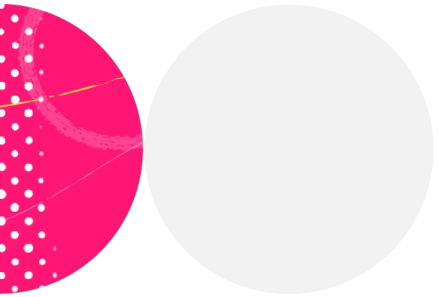
Convenience methods

- StatusCode()
- ContentType()

header()

Hamcrest matchers

Understanding the Different Parameter Settings



Overview

Tests with plain string concatenation

pathParam()

param()

Data-driven tests


Endpoint
https://api.github.com
https://api.github.com/users/{user}
https://api.github.com/users/{user}/repos

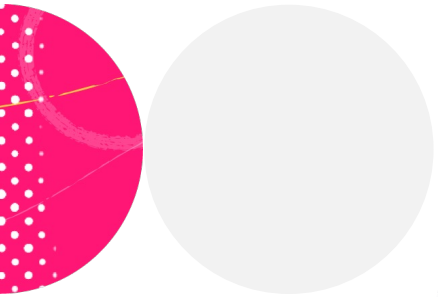

https://api.domain.com/url?query=value&key2=val2
[...]/search/repositories?q=java&per_page=1

```
RestAssured.get("endpoint/{placeholder}", "value")
```

```
RestAssured
    .given()

        .pathParam("user", "some_user")
        .pathParam("repo_name", "repo")

    .get("url/{user}/{repo_name}")
```

```
RestAssured
    .given()

        .param("q","java")
         .param("per_page","1")

    .get("url/search/repositories")
```

Validating the Response Body



Overview

Extract the body from the response

Extract specific values from simple and complex JSON

Use Fluent Interface verifications

```
// headers
status: 200 OK
content-type: application/json; charset=utf-8
// body
{
id: 123
name: xyz
nested: {
   field1: value1
   }
}
```

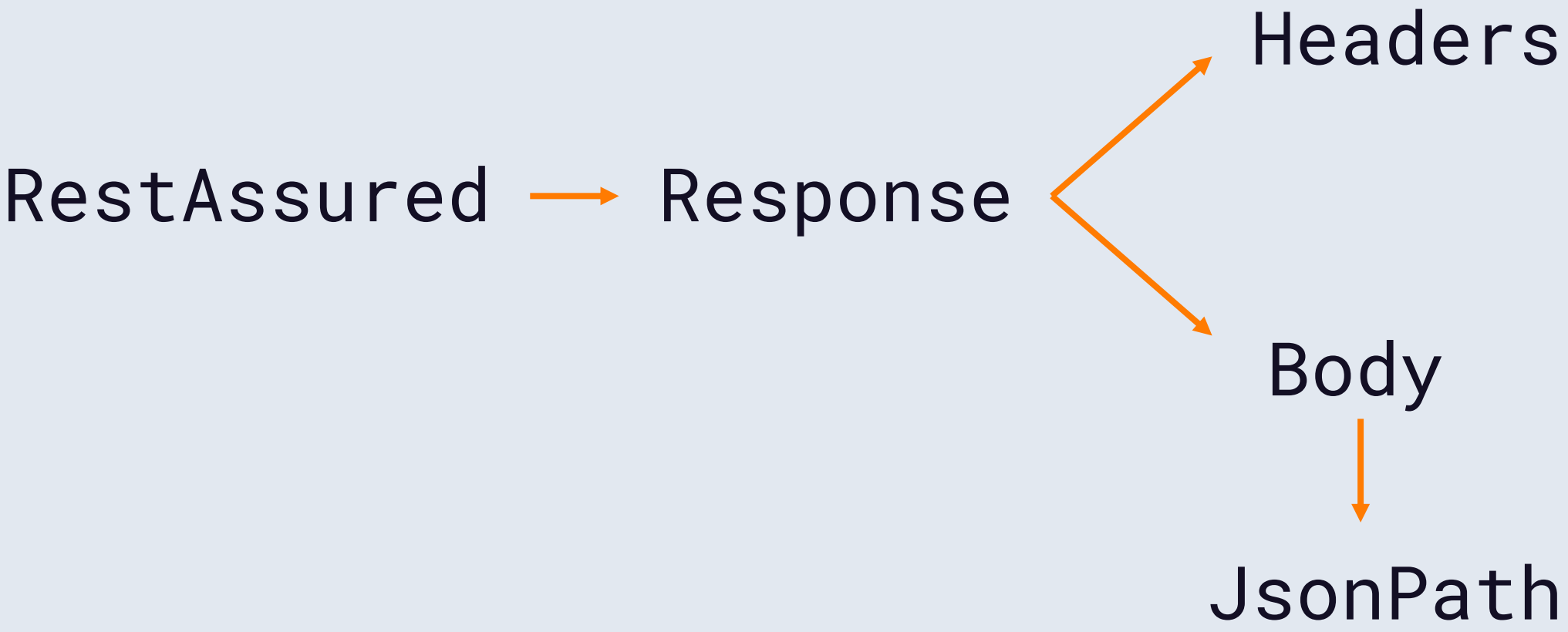
```
// headers
String actual = response.getHeader("headerX");

// body
ResponseBody<?> body = response.getBody();
ResponseBody<?> body1 = response.body();
```

ResponseBody Methods

asByteArray(), asInputStream()

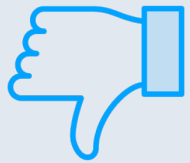
asString()

jsonPath(), xmlPath()


RestAssured
Response
Headers
Body
JsonPath

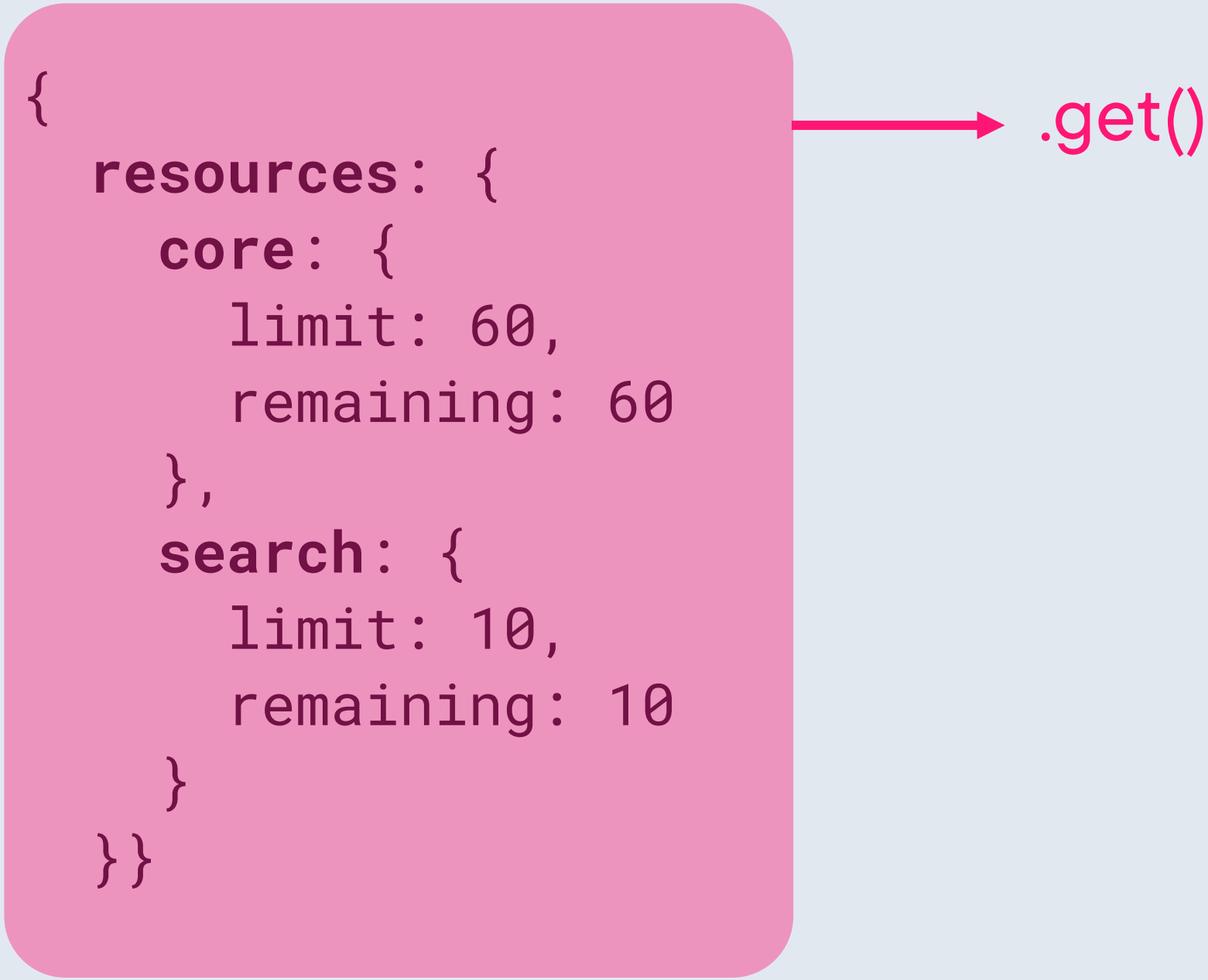
```
{
  resources: {
    core: {
      limit: 60,
      remaining: 60
    },
    search: {
      limit: 10,
      remaining: 10
    }
  }}
```

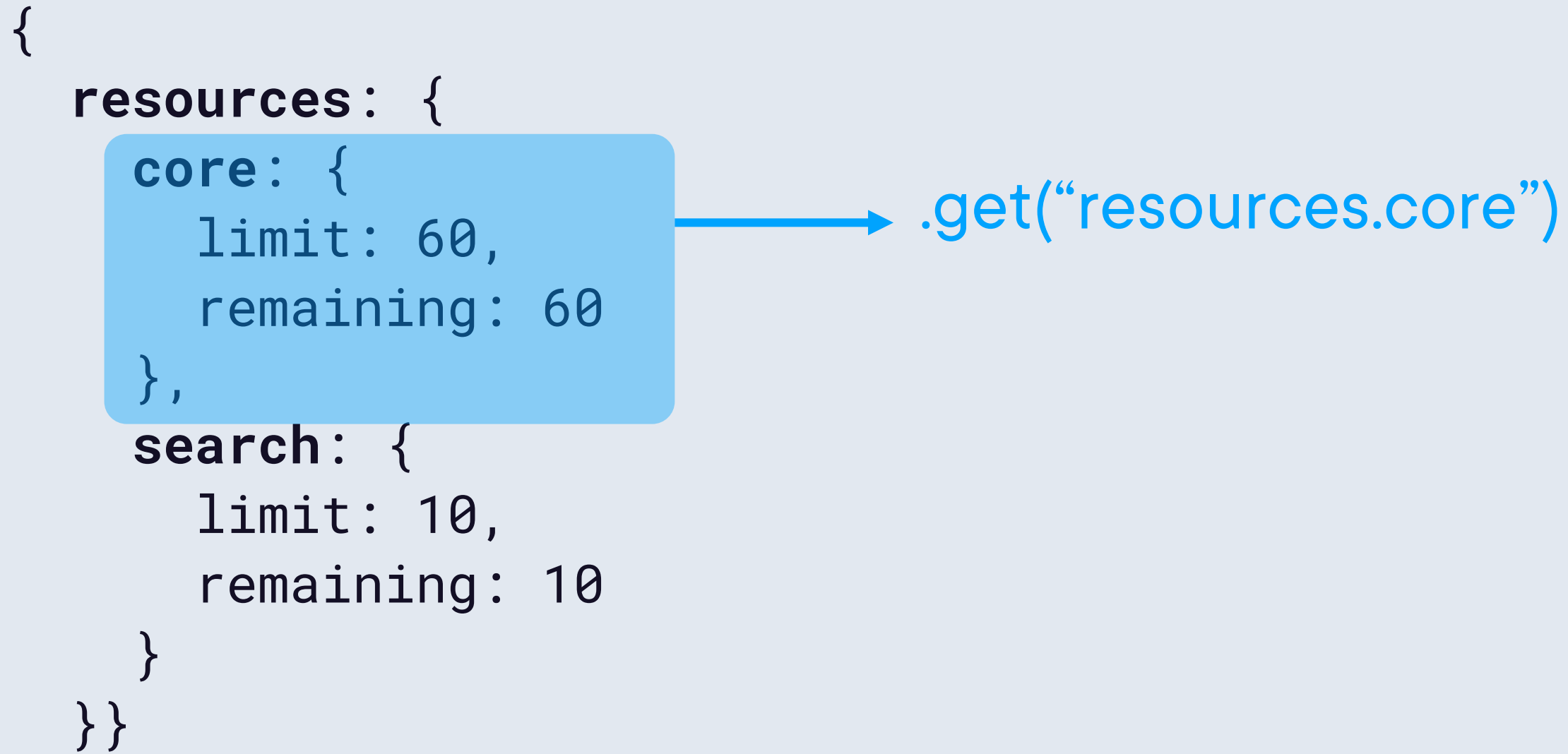
.asString().substring(...).contains("60");

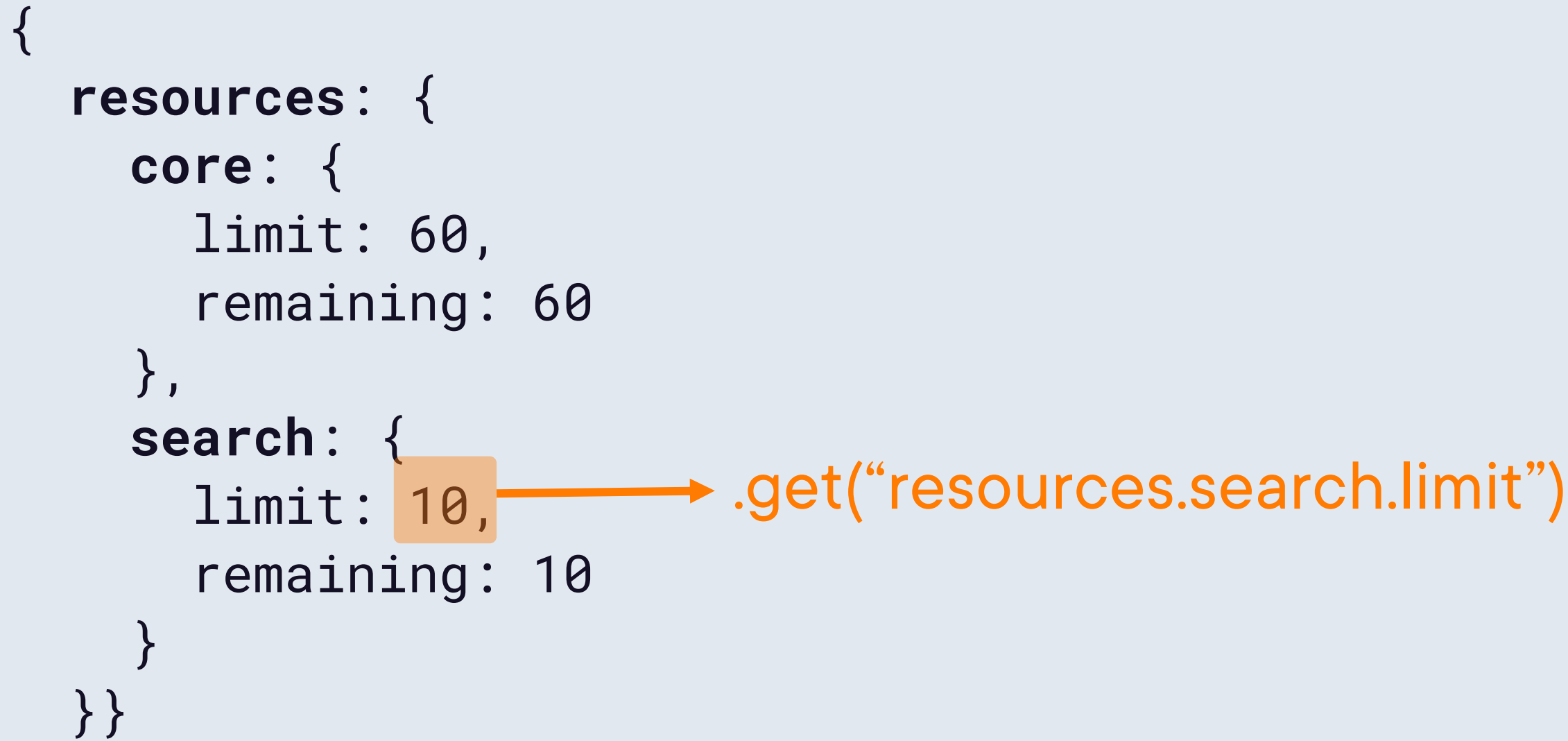


.jsonPath().get() Map > Map > Map

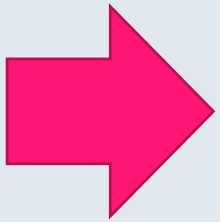
.jsonPath().get("path.to.node") Map > Map


{
resources: {
core: {
limit: 60,
remaining: 60
},
search: {
limit: 10,
remaining: 10
}
}}
.get()


{
resources: {
core: {
limit: 60,
remaining: 60
},
search: {
limit: 10,
remaining: 10
}
}}
.get(“resources.core”)

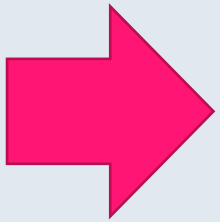

{
resources: {
core: {
limit: 60,
remaining: 60
},
search: {
limit: 10,
remaining: 10
}
}}
.get(“resources.search.limit”)

```
int v =
response.statusCode();
```



```
then().statusCode(200)
```

```
int v =
jPath.get("…");
```



```
then()

.body("…", Matcher m)
```

```
RestAssured.get(url)
           .then()
           .body(Matcher m, Matcher ms)
           .body(String path, Matcher m, Object… ms)
           [...]
```

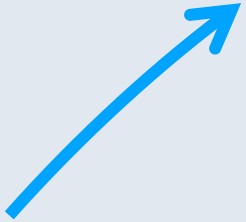
```
{
    field1: “value1”
    field2: “value2”
}
```

→ .get(“field1”)

```
{
    node1: {
        field3: “value3”
    }
}
```

→ .get(“node1.field3”)

String v = “node1.nested;”

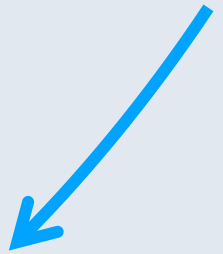


```
RestAssured.get(url)
           .then()
           .body(“node1.nested.field1”, equalTo(x))
           .body(“node1.nested.field2”, equalTo(y))
           .body(“node1.nested.field3”, equalTo(z))
```

```
RestAssured.get(url)
           .then()
           .rootPath("node1.nested")
              .body("field1", equalTo(x))
              .body("field2", equalTo(y))
              .body("field3", equalTo(z))
```

```
RestAssured.get(url)
           .then()
           .rootPath("node1.nested")
              .body("field1", equalTo(x))
              .body("field2", equalTo(y))
           .rootPath("node2.nested2")
              .body("fieldX", equalTo(z))
```

we keep going!



```
RestAssured.get(url)
        .then()
        .rootPath("node1.nested")
           .body("field1", equalTo(x))
        .rootPath("node2.nested2")
           .body("fieldX", equalTo(z))
        .noRootPath()
        .body("full.path.again", equalTo(a))
```

Rule of Thumb:

Tests should be small, focused and self-contained

```
java.lang.AssertionError: 1 expectation failed.
JSON path data.id doesn't match.
Expected: <1>
  Actual: <[1, 2, 3, 4, 5, 6]>
```

```
.body("data.first_name", contains("Eve", "Emma"))
.body("data.first_name", containsInAnyOrder("Eve", "Emma"))

.body("data.first_name", hasItem("Emma"))
.body("data.first_name",hasItem(endsWith("ma")))
```

Summary

ResponseBody useful methods

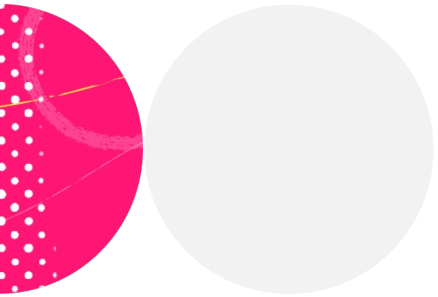
body() and jsonPath()

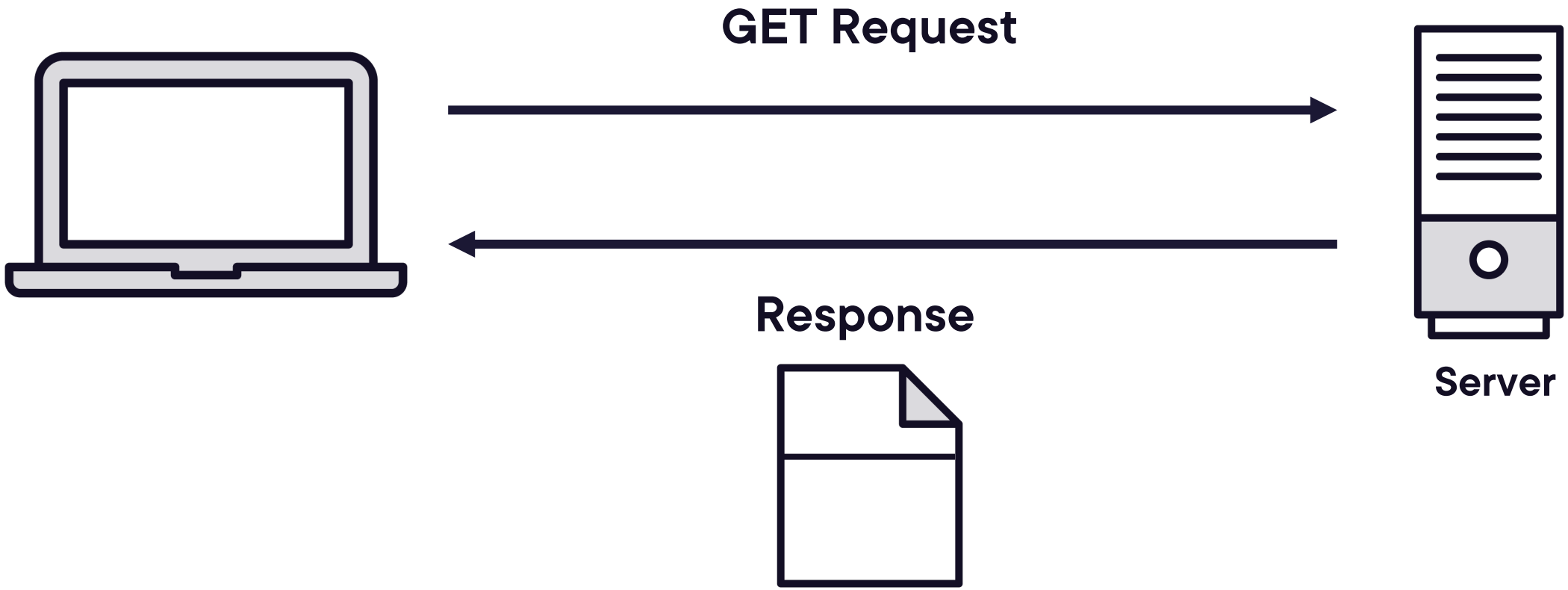
rootPath()

Dealing with repeating items

defaultParser()

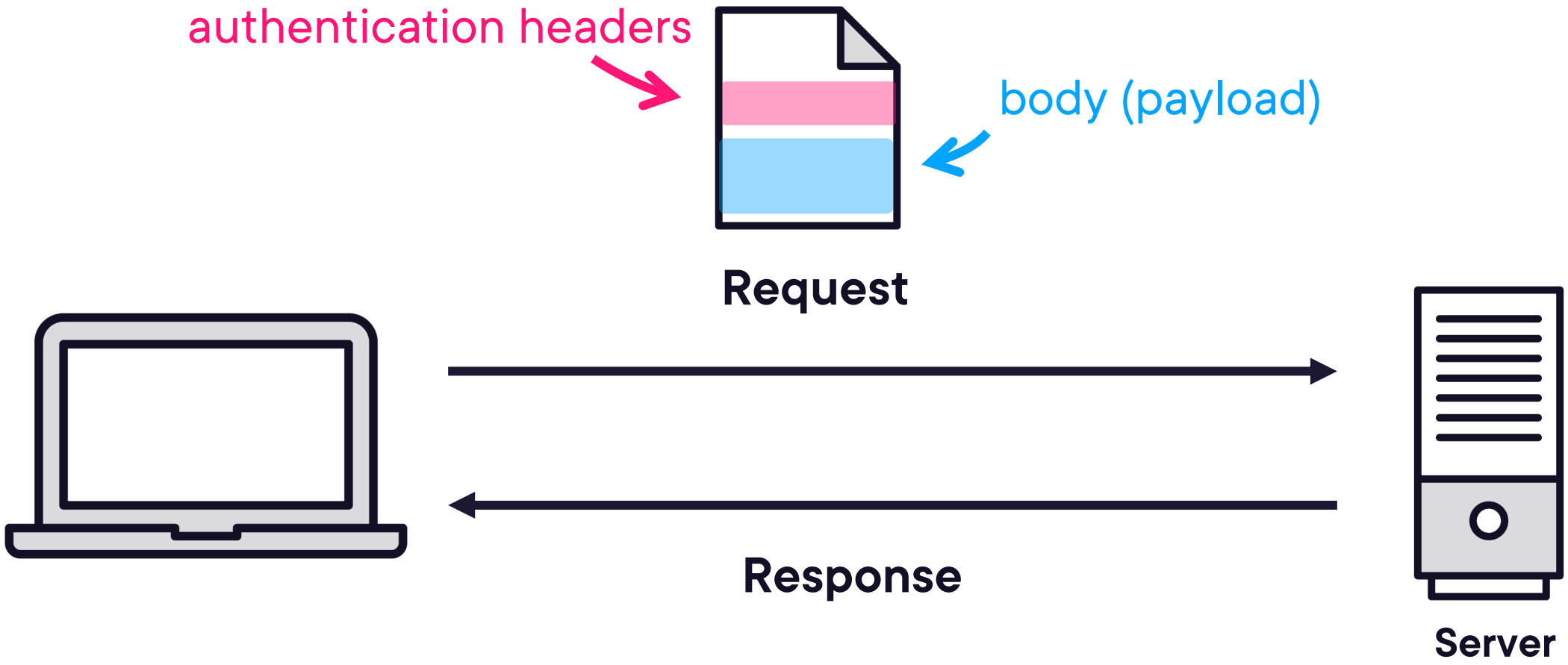
Sending Create, Update, and Delete Requests




GET Request
Response
Server

HTTP method	Request has body	Safe
GET	No	Yes
POST	Yes	No
PUT	Yes	No
DELETE	Yes	No

Should require authentication


authentication headers
body (payload)
Request
Response
Server

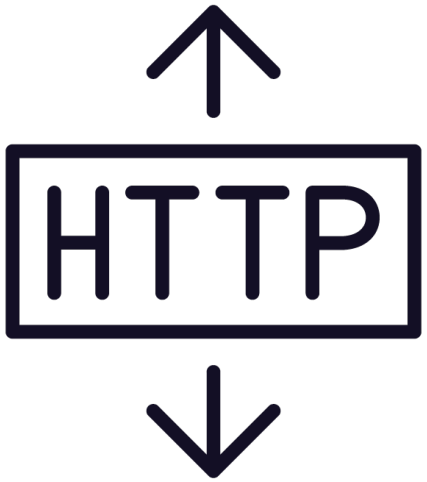
Overview

Review of HTTP methods

Use HEAD and OPTIONS

POST, PATCH, DELETE

Simple and Safe Methods



GET

- Retrieve data only
- No side-effects

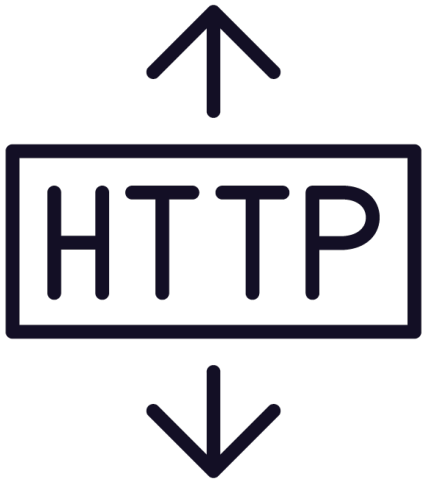
HEAD

- Get headers only, no body
- Useful to fetch metadata fast

OPTIONS

- List of supported HTTP methods

Destructive Methods



POST and PUT

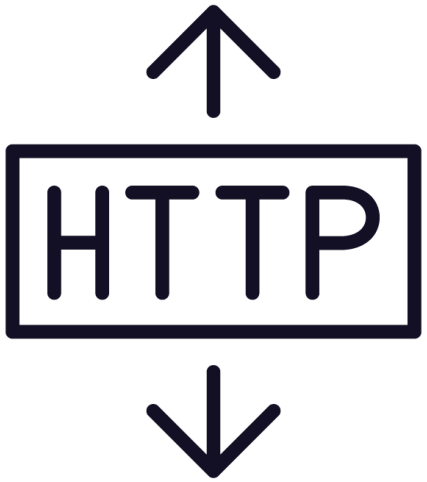
- Create or overwrite from the Request's body

PATCH

- Apply partial modifications

DELETE

Honorable Mention



TRACE

CONNECT

BDD Keywords:
Given
When
Then

Given: <some condition>

When: <action>

Then: <expected output>

Given: API URL is http://api.github.com

When: User sends a GET request

Then: Response status code is 200

```
.given()
  .header(“x”, “y”)
  .body(“payload”)
.when()
   .post(“url”)
.then()
   .statusCode(xyz);
```

```
given()
    .auth()      // AuthenticationSpecification
    .basic(…)    // usr / pwd
    .oauth2(…) // access token
    .ntlm(…)
    .digest(…)
    .form(…)
```

GET

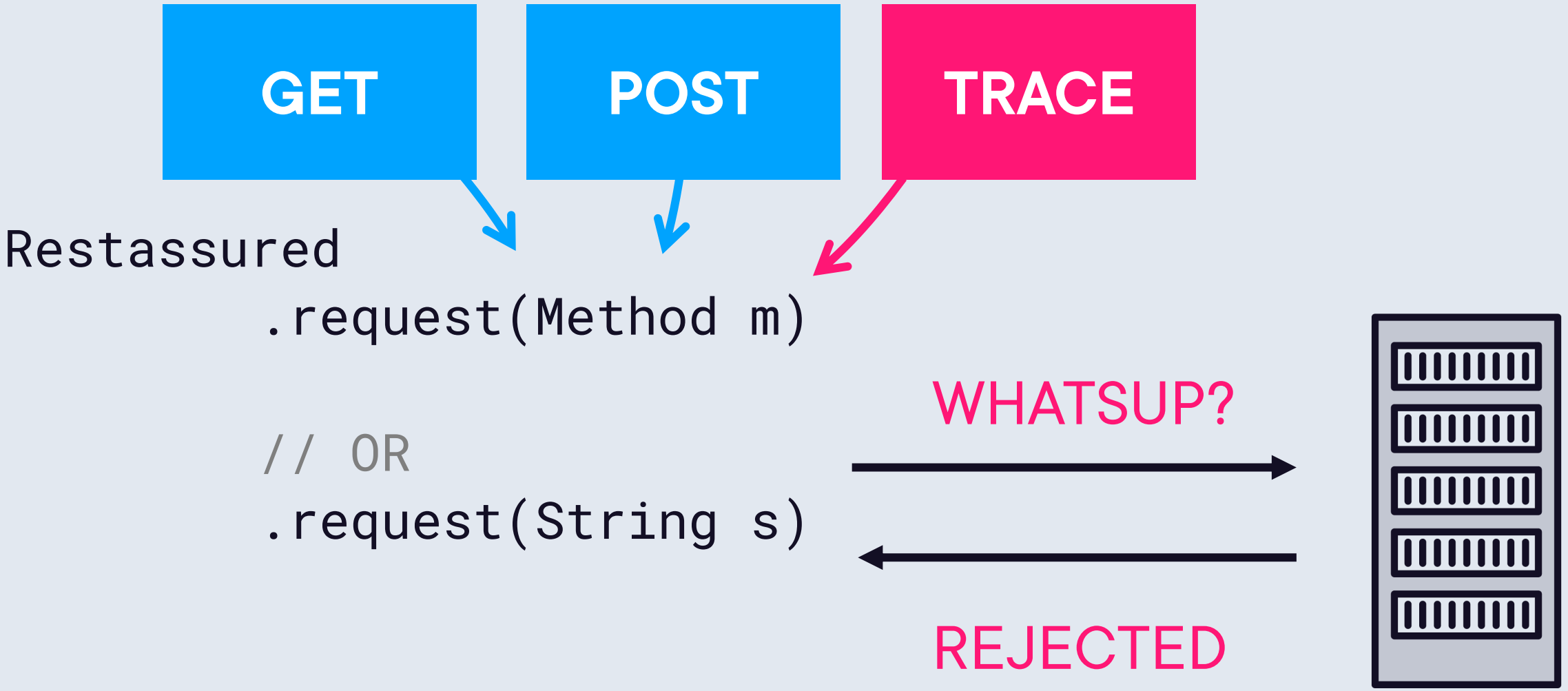
HEAD

OPTIONS

POST

PATCH

DELETE


GET
POST
TRACE
Restassured
.request(Method m)
// OR
.request(String s)
WHATSUP?
REJECTED

Summary

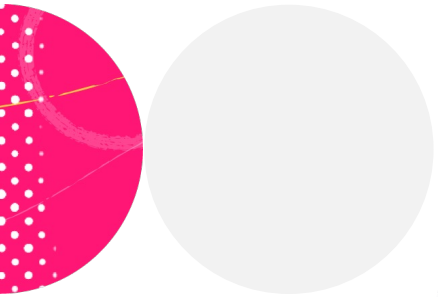
HEAD and OPTIONS: retrieve metadata

Create, Change, and Delete methods

Set authentication in the header

.request() function

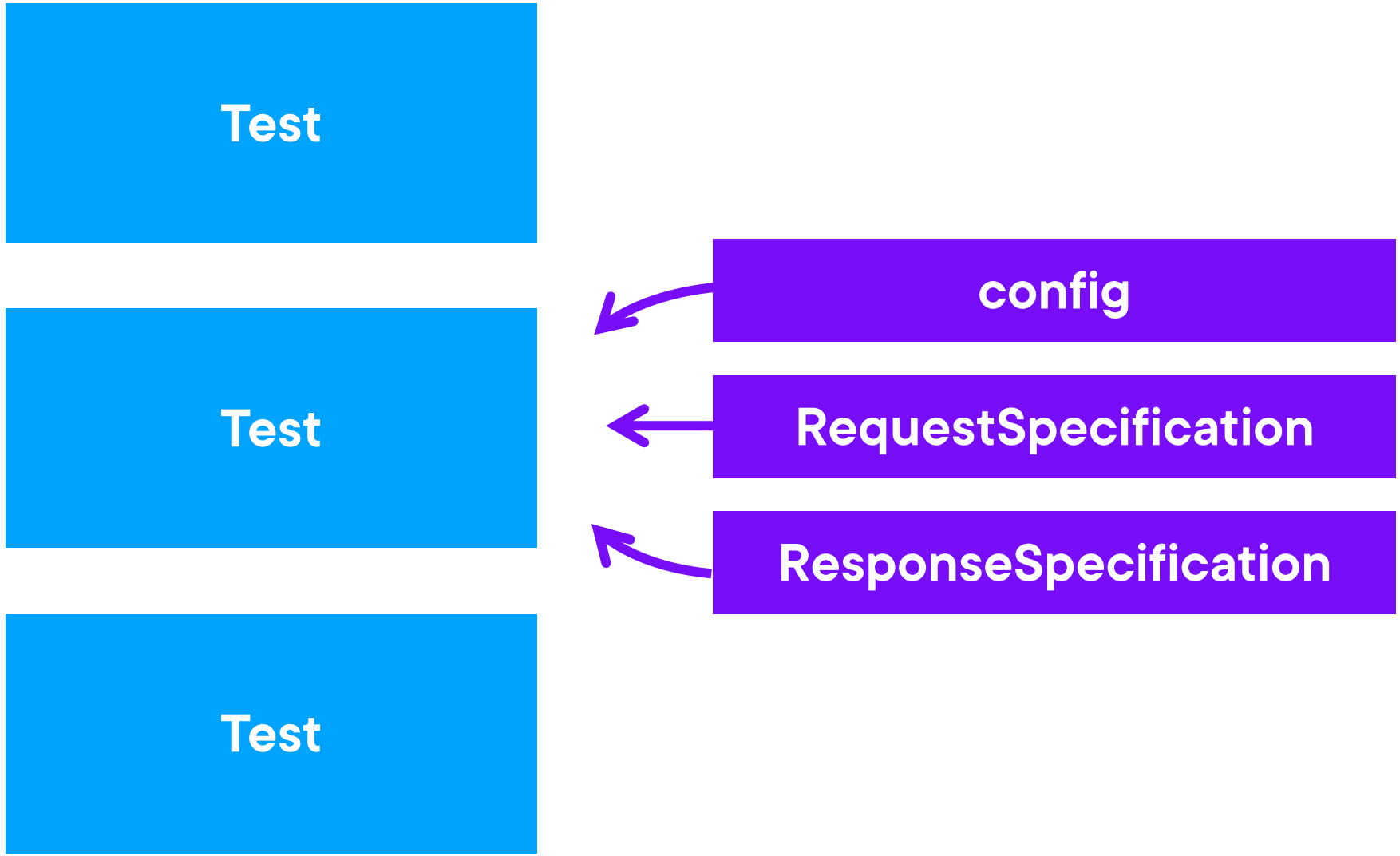
Leveraging REST Assured Configuration



Test
(duplicate code)

Test
(duplicate code)

Test
(duplicate code)

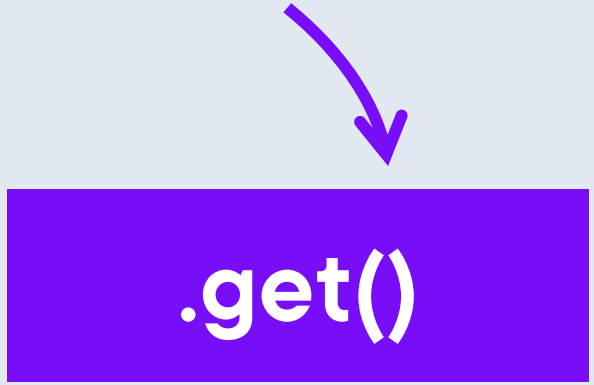

Test
Test
Test
config
RequestSpecification
ResponseSpecification

Overview

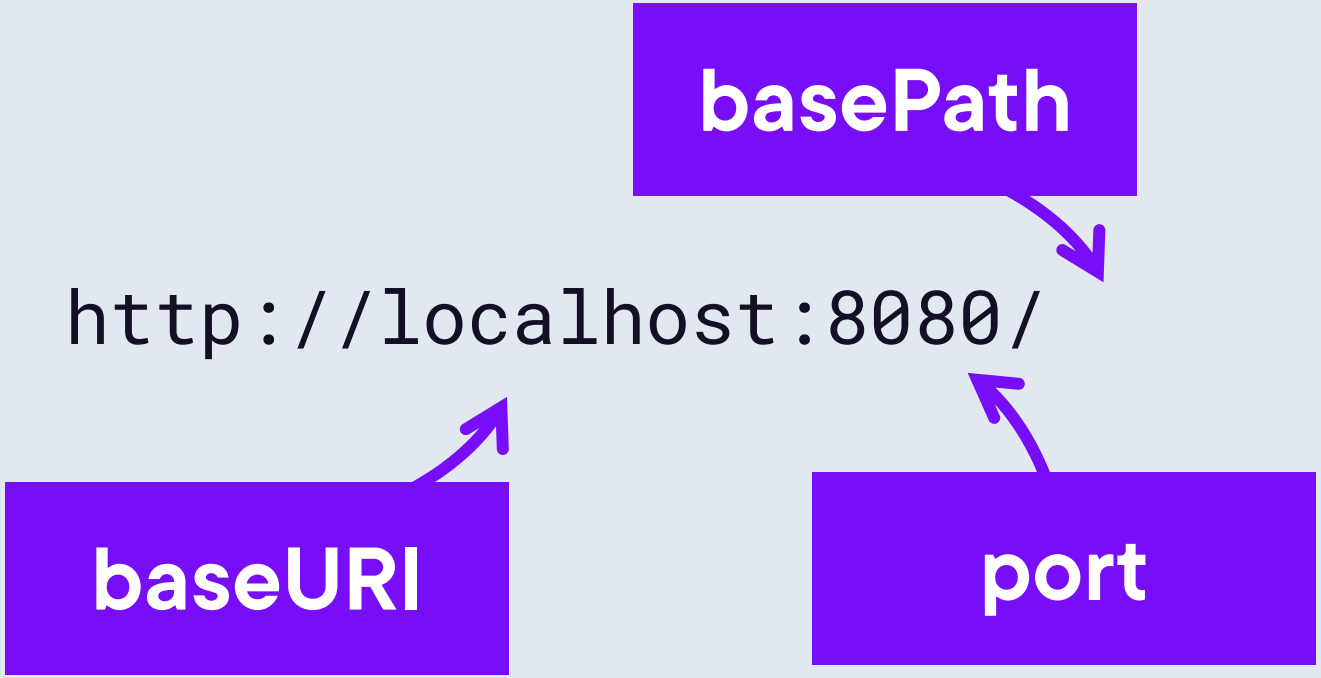
- Global REST Assured variables
- Global config
- Request specification
- Response specification
- Logging

```
@Test
void testOne() {
    RestAssured.get(“url”)
    //…
}
@Test
void testTwo() {
    RestAssured.get(“url”)
    //…
}
```

RestAssured.baseUri

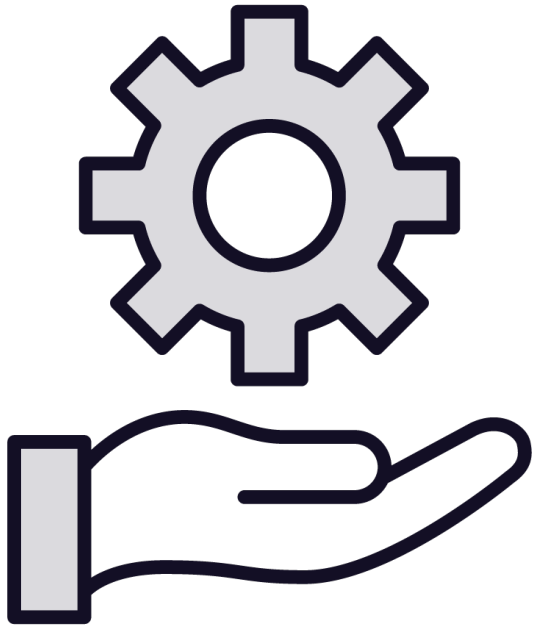


```
public class RestAssured {
    String baseURI
    int port
    String basePath
    String rootPath
    //…
    RestAssuredConfig config
    RequestSpecification requestSpecification
    ResponseSpecification responseSpecification
}
```

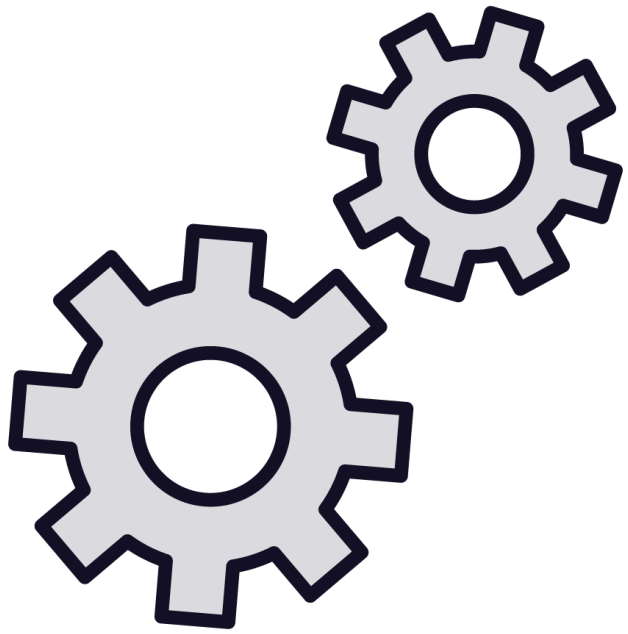

basePath
http://localhost:8080/
baseURI
port

```
java.net.ConnectException: Connection refused: connect
```

RestAssuredConfig



- Underlying HttpClient behavior
- Redirecting
- Connection timeout
- Logging
- Encoding and decoding
- Behavior in case of test failure



RequestSpecification:

- What to send on all or some requests

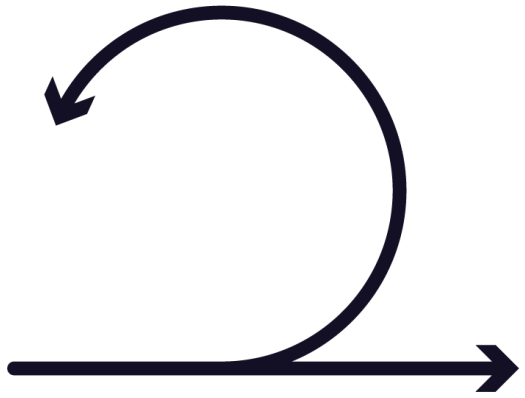
ResponseSpecification:

- What to verify on all or some responses

URL redirection

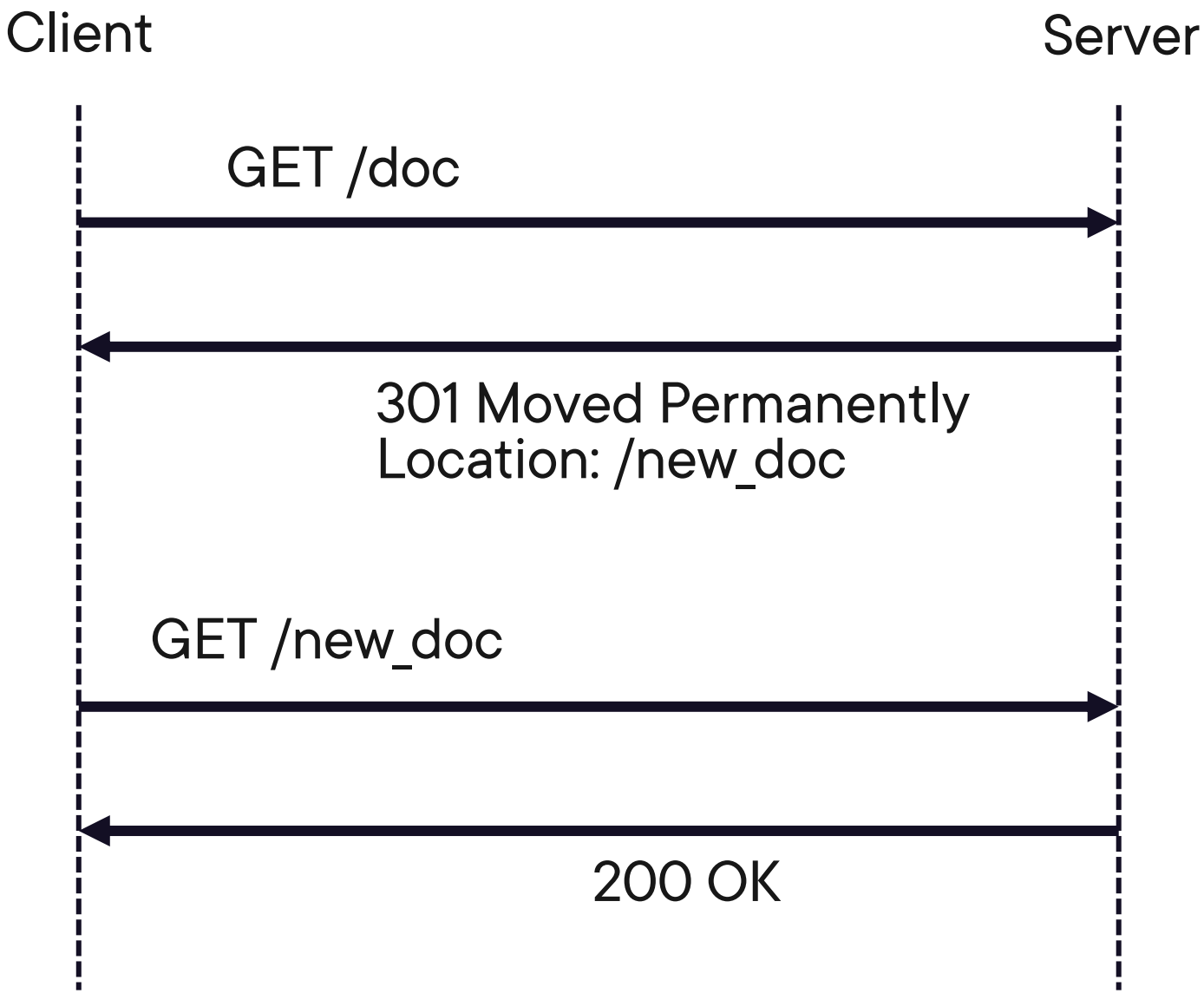
a.k.a. URL forwarding - a technique to give more than one URL address to a resource

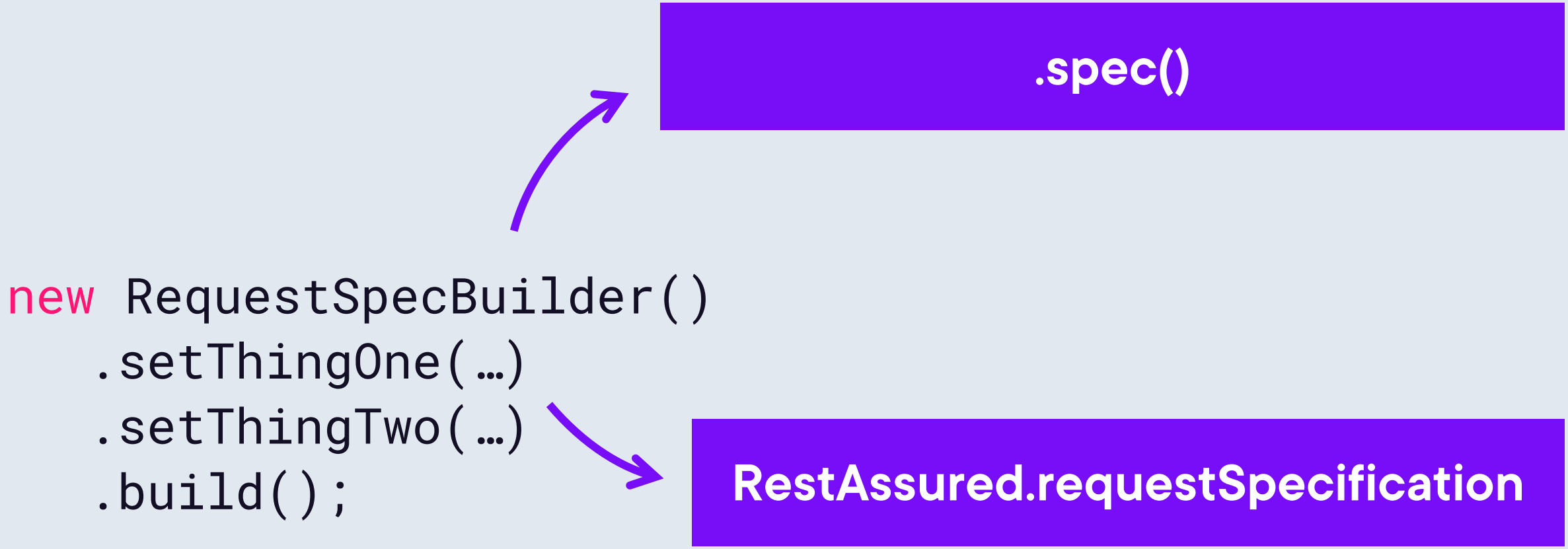
URL Forwarding



Temporarily redirect during site maintenance or downtime

In case of permanent redirects - to preserve existing links after changing the site's URLs


Client
Server
GET /doc
301 Moved Permanently
Location: /new_doc
GET /new_doc
200 OK


.spec()
new RequestSpecBuilder()
.setThingOne(…)
.setThingTwo(…)
.build();
RestAssured.requestSpecification

.then()
.spec()

```
new ResponseSpecBuilder()
    .expectStatusCode(200)
    .expectContentType(JSON)
    .build();
```

RestAssured.responseSpecification

```
RestAssured
    .given()
        .spec(commonRequestSpec)
    .when()
        .get(url)
    .then()
        .spec(commonResponseSpec)
        // more verifications
```

Debugging

peek(), print()

Logging

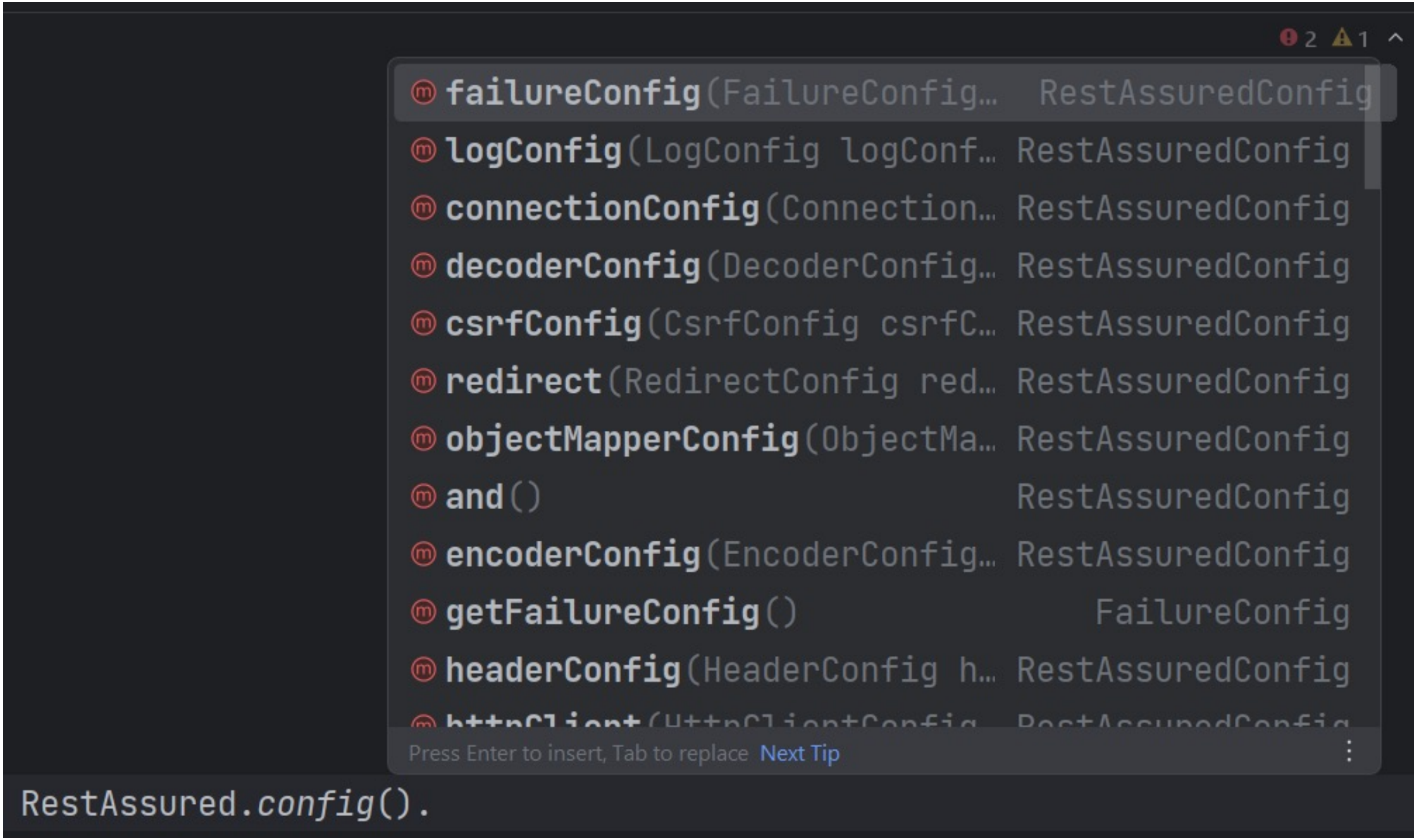
log()

```
RestAssured
    .given()
        .spec(commonRequestSpec)
        .log().all()
    .when()
        .get(url)
    .then()
        .spec(commonResponseSpec)
        .log().headers()
```

```
.log()
    .ifStatusCodeMatches(…)
    .ifError(…)
    .ifValidationFails(…)
```

```
RestAssured.baseURI = "https://your.api/";
RestAssured.basePath = "some/path";
```

```
var commonResponseSpec =
      new ResponseSpecBuilder()
         .expectStatusCode(200)
         .expectContentType(ContentType.JSON)
         . build();

// [...]
@Test
RestAssured.get(url)
   .then()
      .spec(commonResponseSpec)
      // ...
```


2 1
failureConfig(FailureConfig… RestAssuredConfig
logConfig(LogConfig logConf… RestAssuredConfig
connectionConfig(Connection… RestAssuredConfig
decoderConfig(DecoderConfig… RestAssuredConfig
csrfConfig(CsrfConfig csrfC… RestAssuredConfig
redirect(RedirectConfig red… RestAssuredConfig
objectMapperConfig(ObjectMa… RestAssuredConfig
and() RestAssuredConfig
encoderConfig(EncoderConfig… RestAssuredConfig
getFailureConfig() FailureConfig
headerConfig(HeaderConfig h… RestAssuredConfig
Press Enter to insert, Tab to replace Next Tip
RestAssured.config().

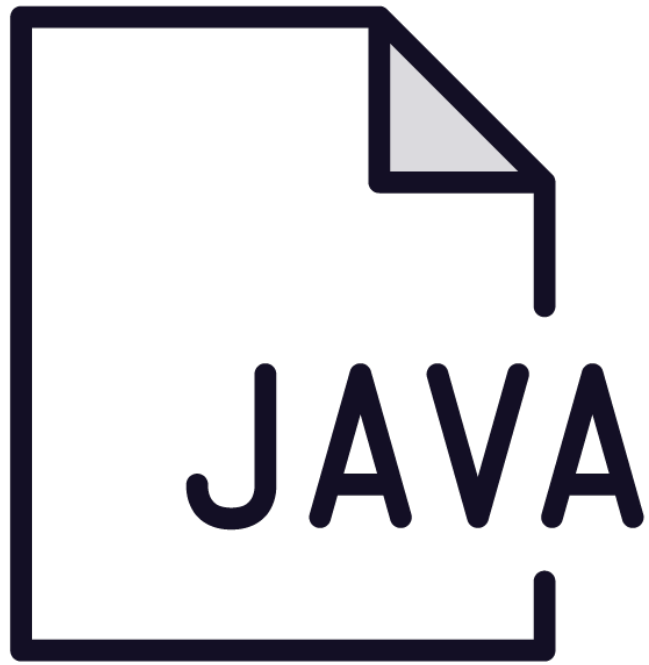
Writing Efficient and Effective Test Automation Code

```
@BeforeMethod
void setup() {

   RestAssured.baseUri = “some/url”
   RestAssured.responseSpecification = getSpec();

}
```

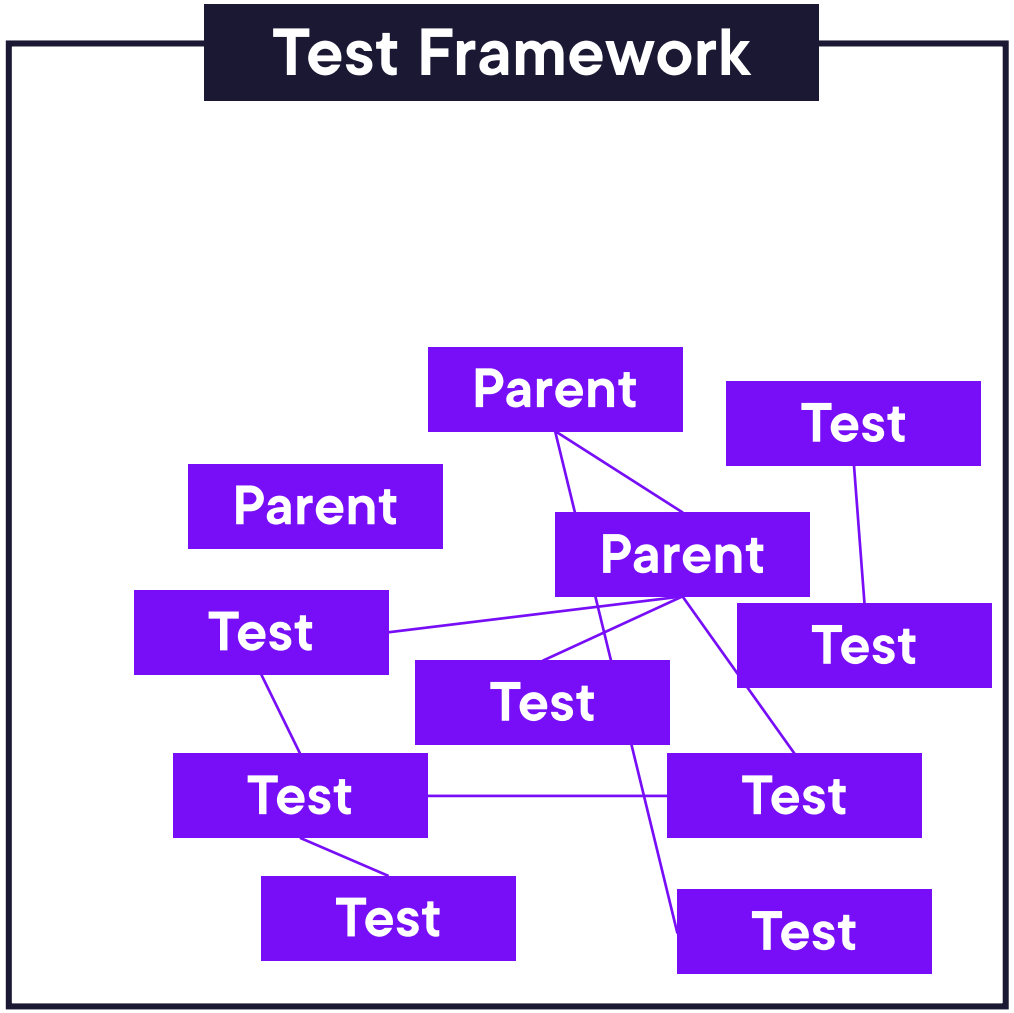
DRY

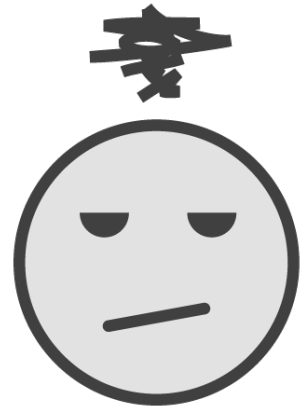
Overview

- Leverage inheritance
- Inheritance vs. Composition
- Test independence
- Further learning

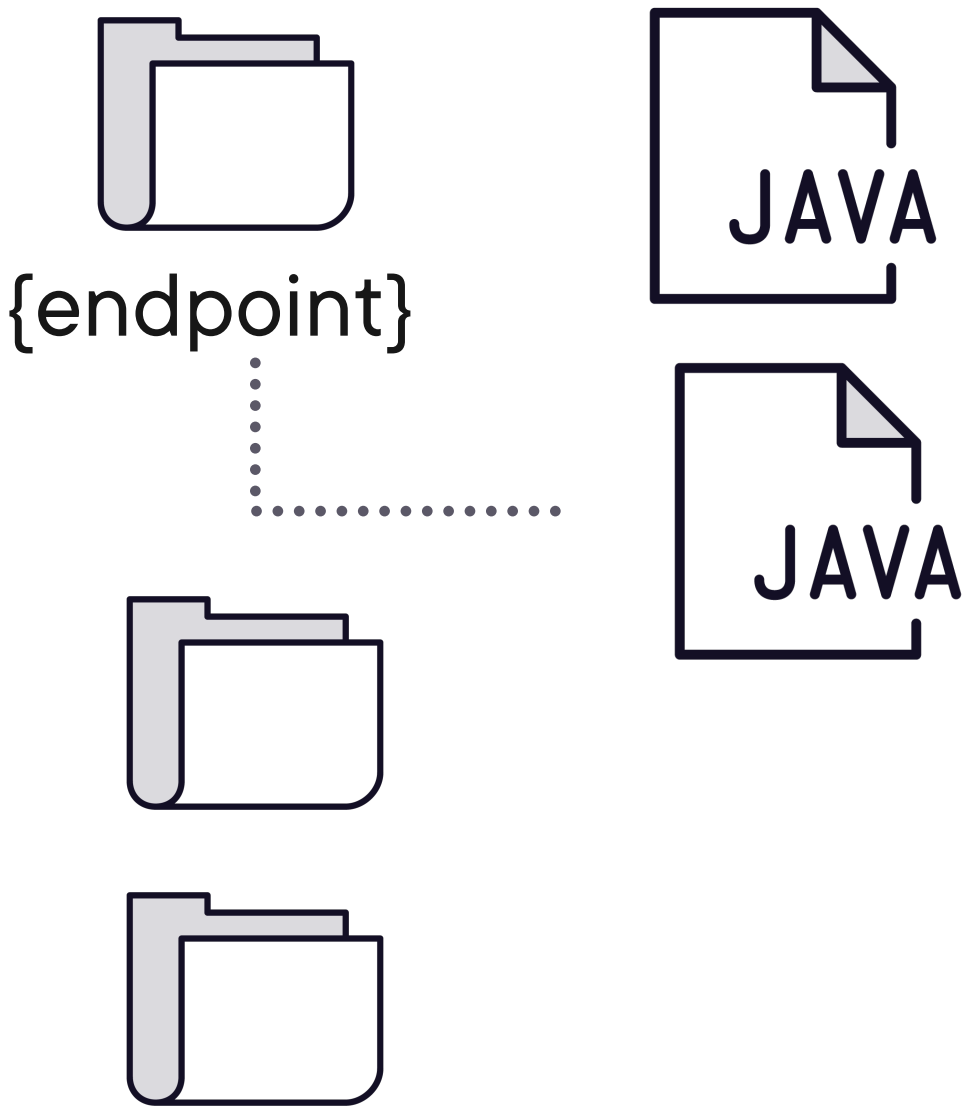

JAVA


JAVA
JAVA
JAVA

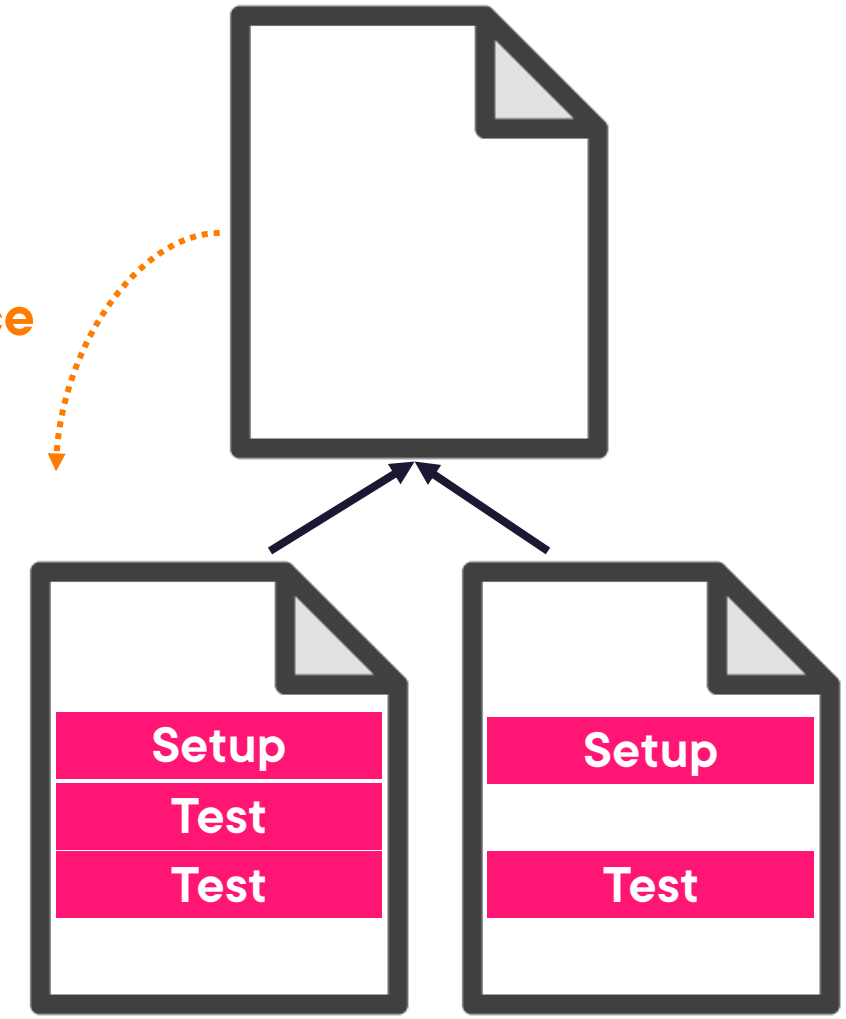

Test Framework
Parent
Test
Parent
Parent
Test
Test
Test
Test
Test
Test
Test
Test

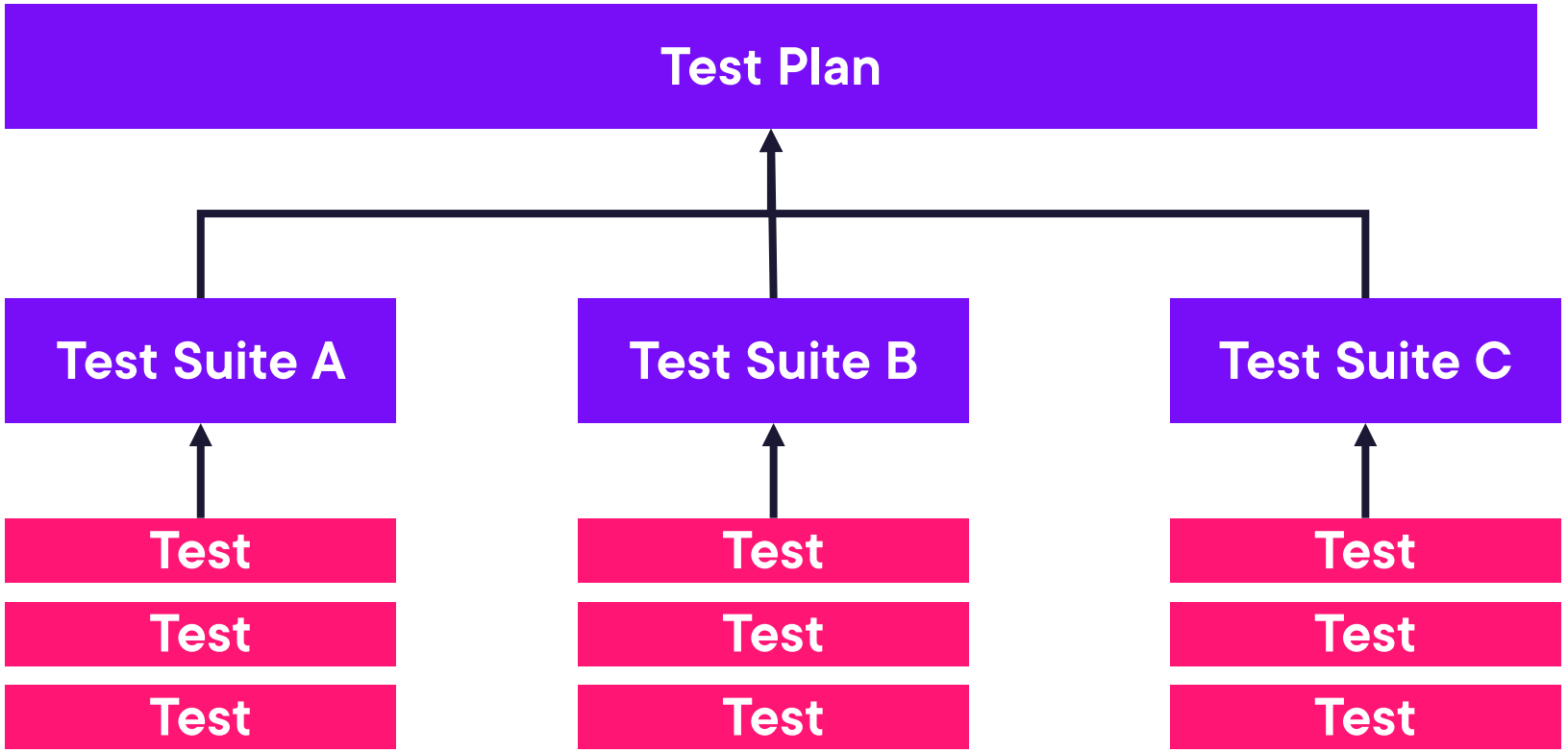


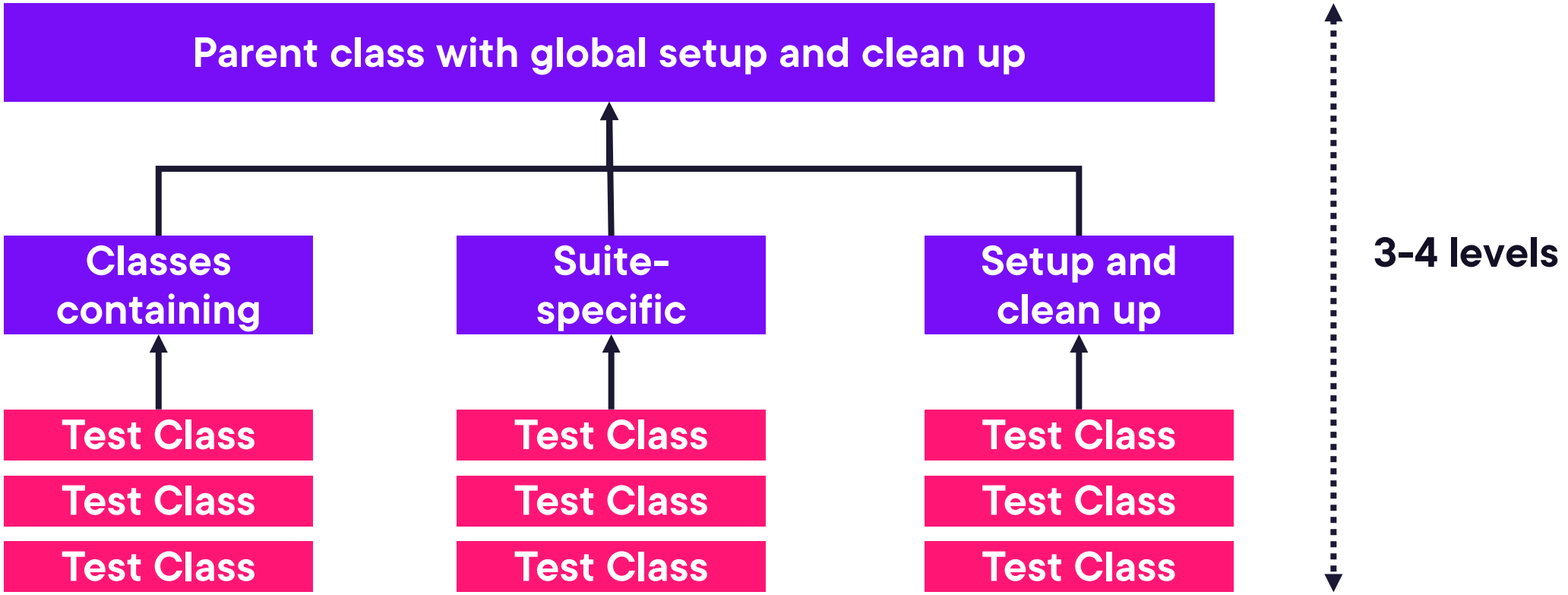
What a mess!


{endpoint}
JAVA
JAVA

OOP: Inheritance

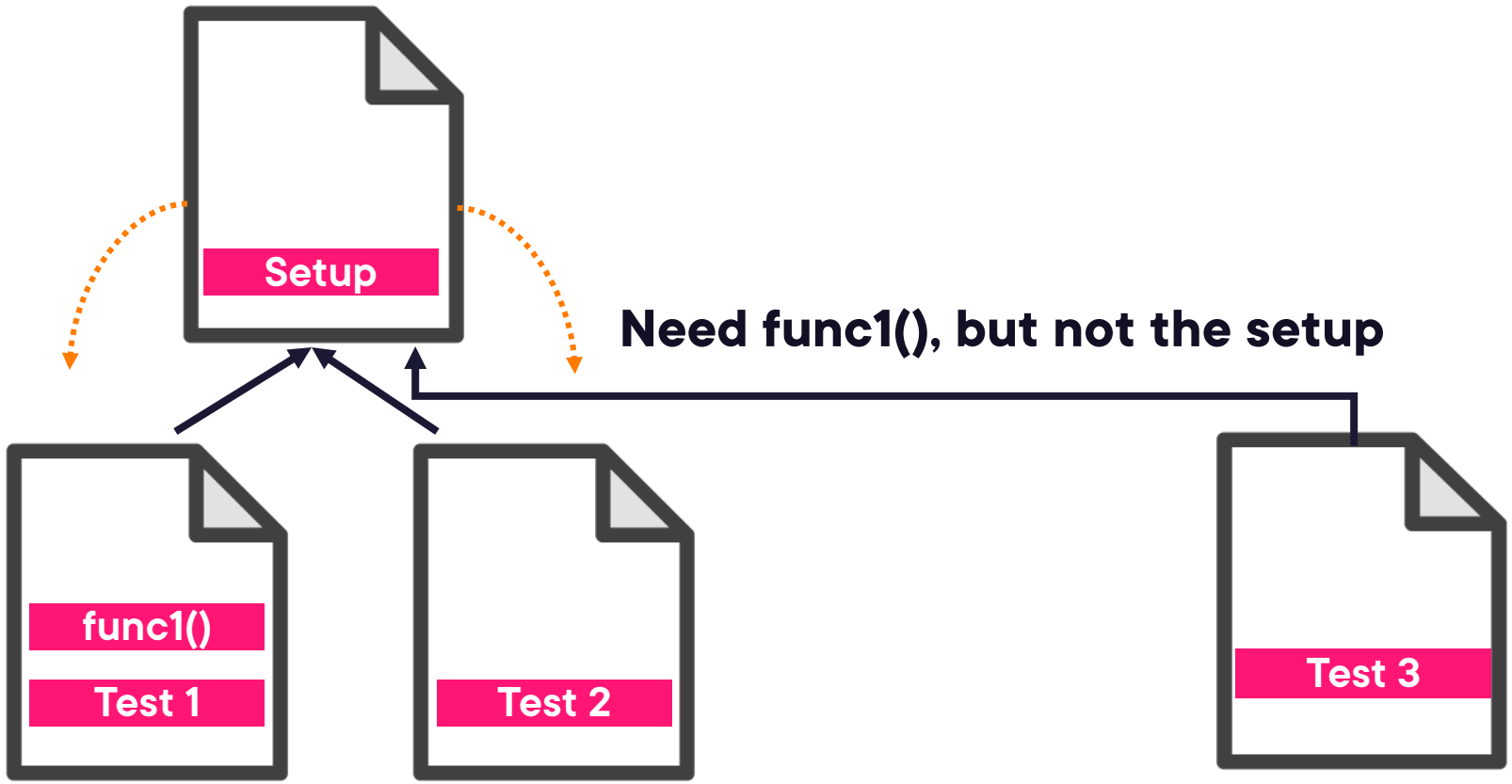


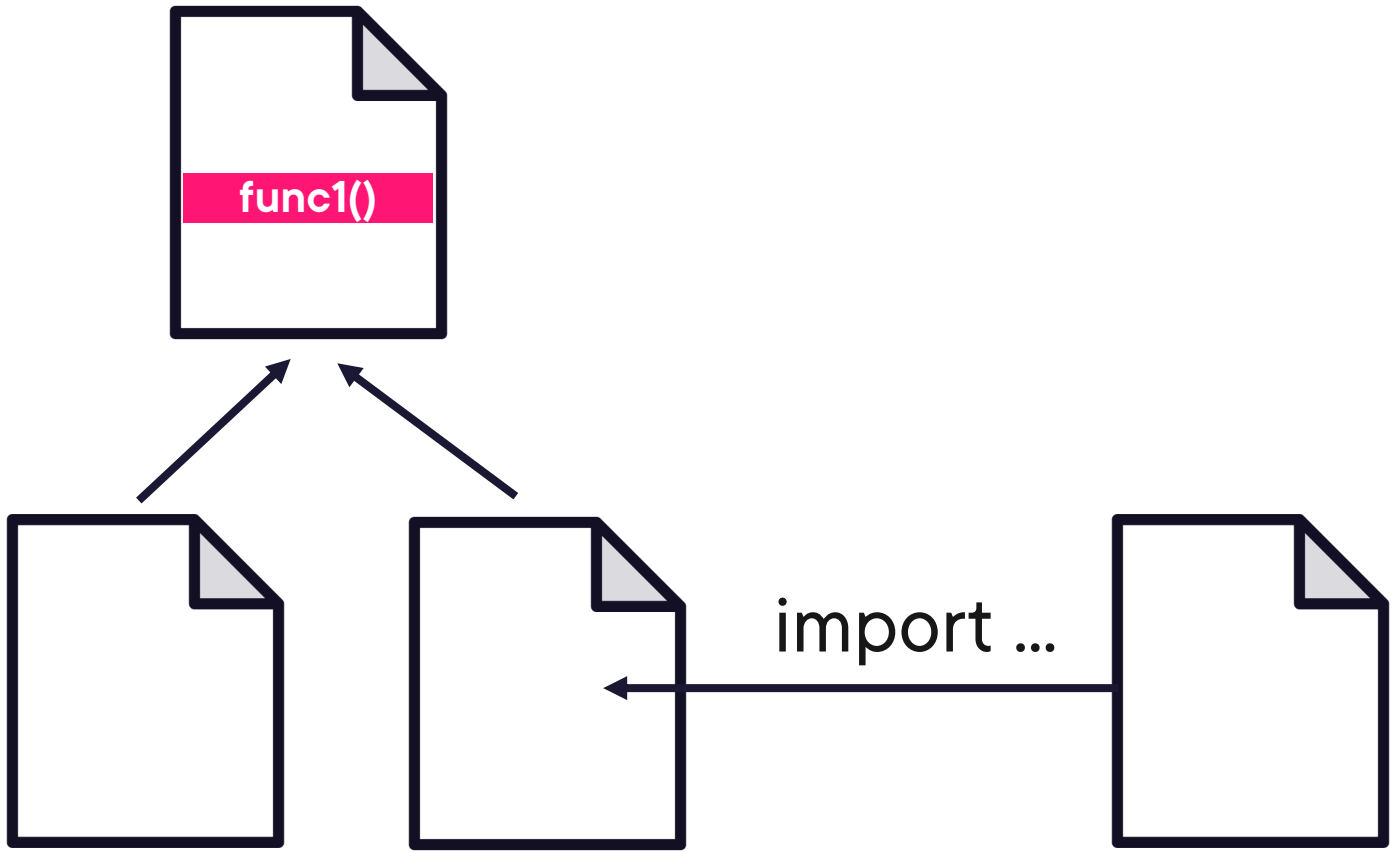

Test Plan
Test Suite A
Test Suite B
Test Suite C
Test
Test
Test
Test
Test
Test
Test
Test
Test

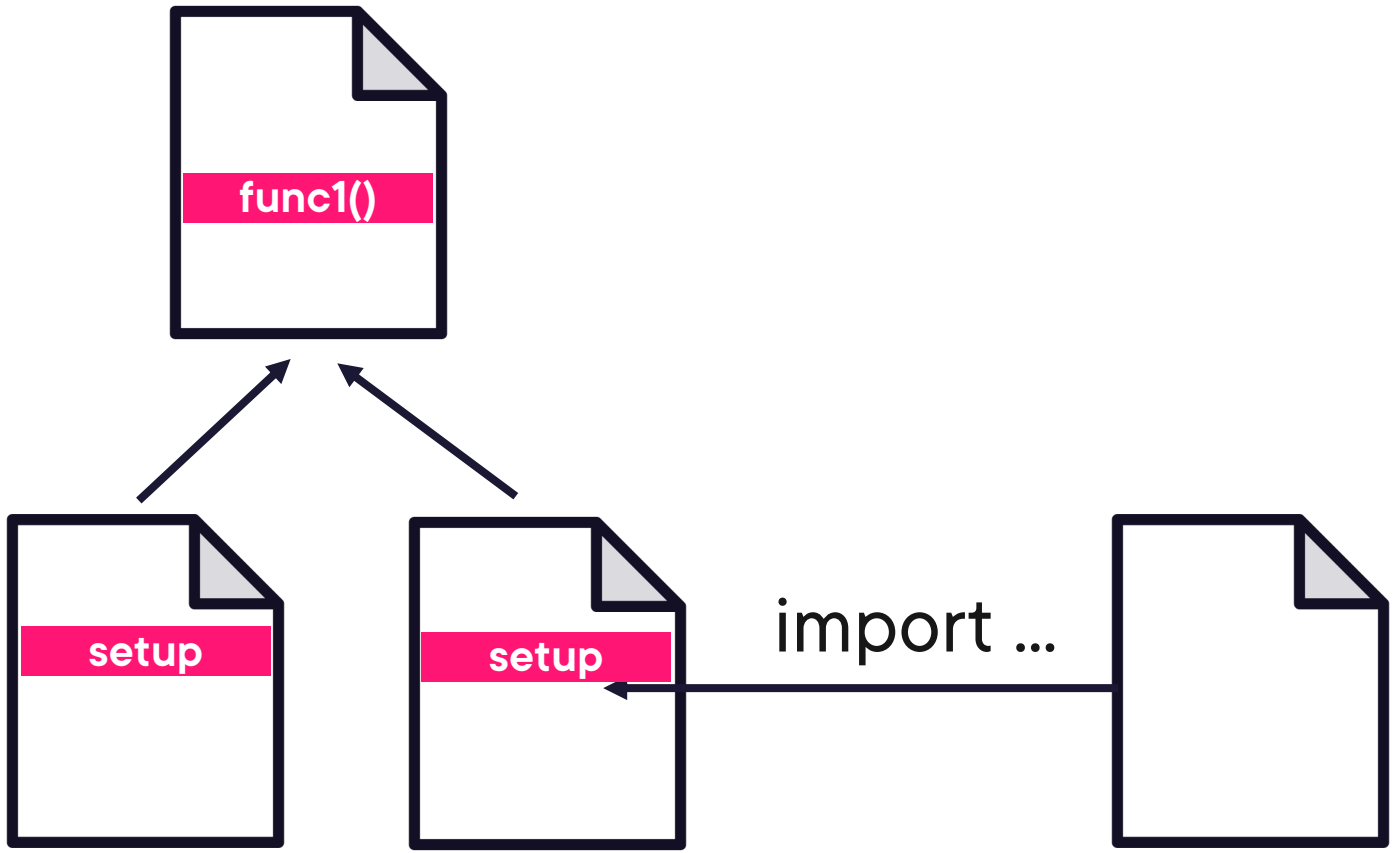

Parent class with global setup and clean up
Classes containing
Suite-specific
Setup and clean up
Test Class
Test Class
Test Class
Test Class
Test Class
Test Class
Test Class
Test Class
Test Class
3-4 levels

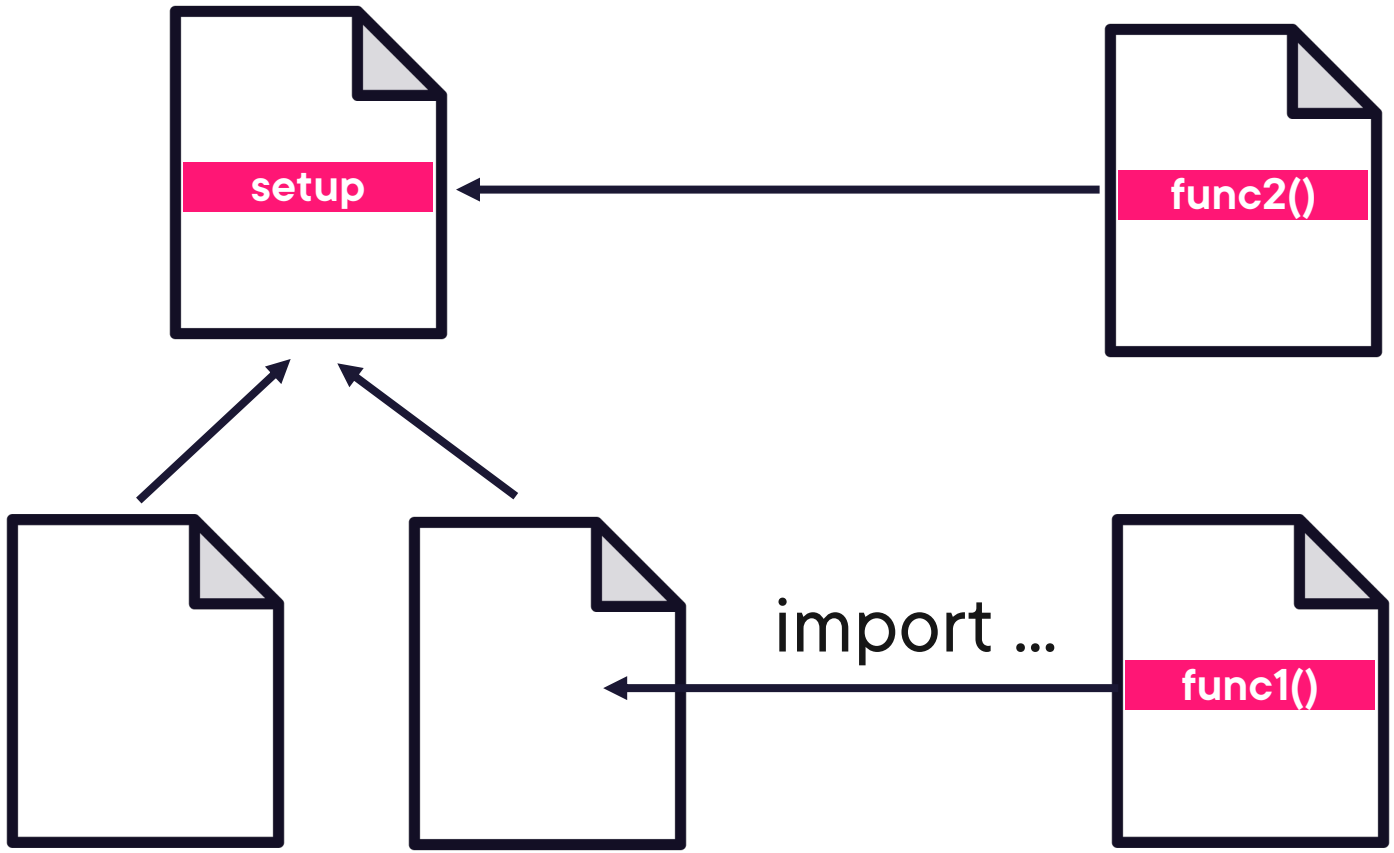
Suggestion:

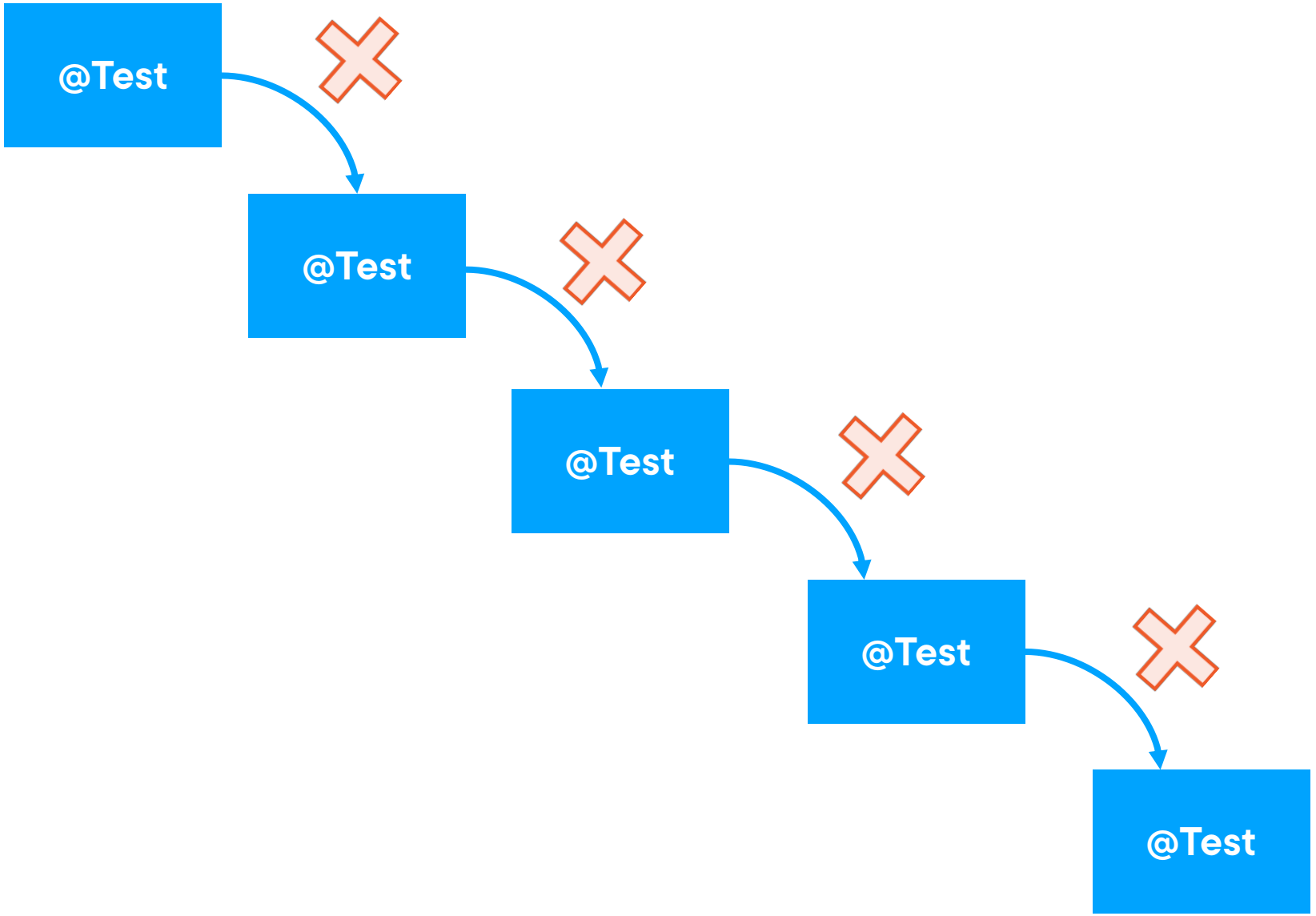
Use inheritance to refactor common setup and cleanup code


Setup
Need func1(), but not the setup
func1()
Test 1
Test 2
Test 3

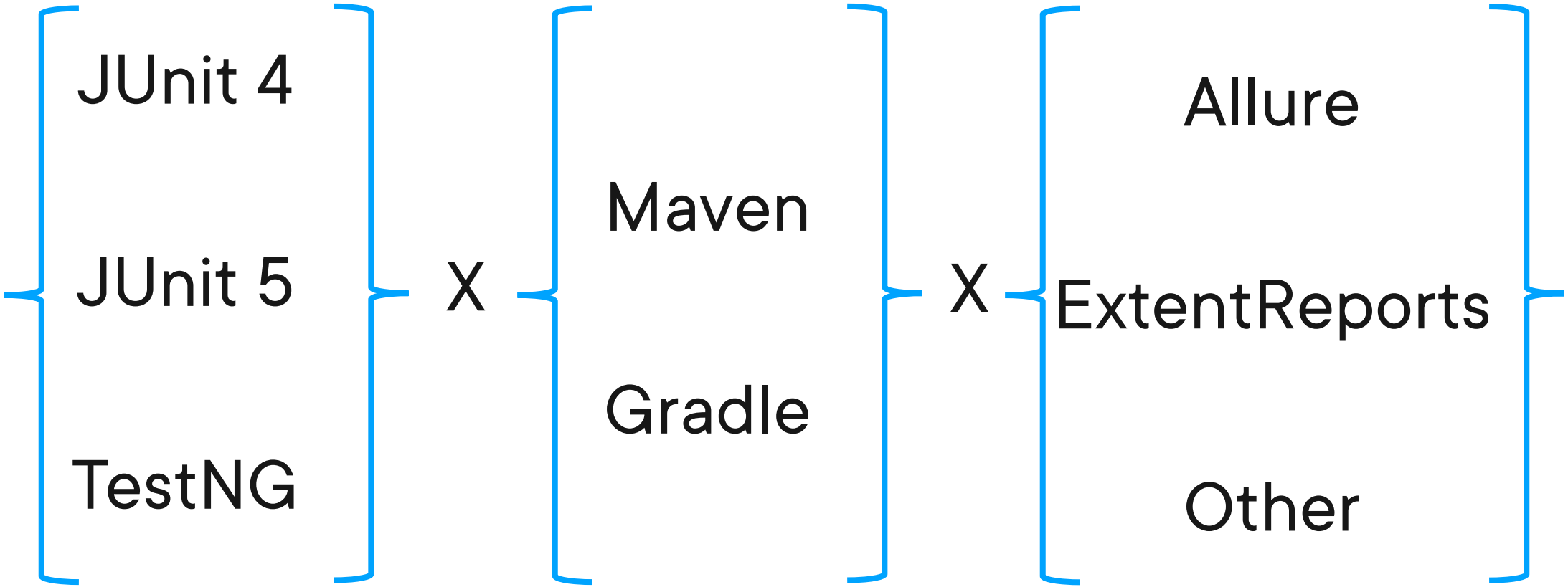

func1()
import ...


func1()
setup
setup
import ...


setup
func2()
import ...
func1()


@Test
@Test
@Test
@Test
@Test

```
@Test
public void postTest() {…}

@Test(dependsOnMethods = "postTest")
public void deleteTest() {…}
```


JUnit 4
JUnit 5
TestNG
X
Maven
Gradle
X
Allure
ExtentReports
Other

Read the official docs!