Writing Your First Web API Tests

Overview

Review Web API and HTTP basics

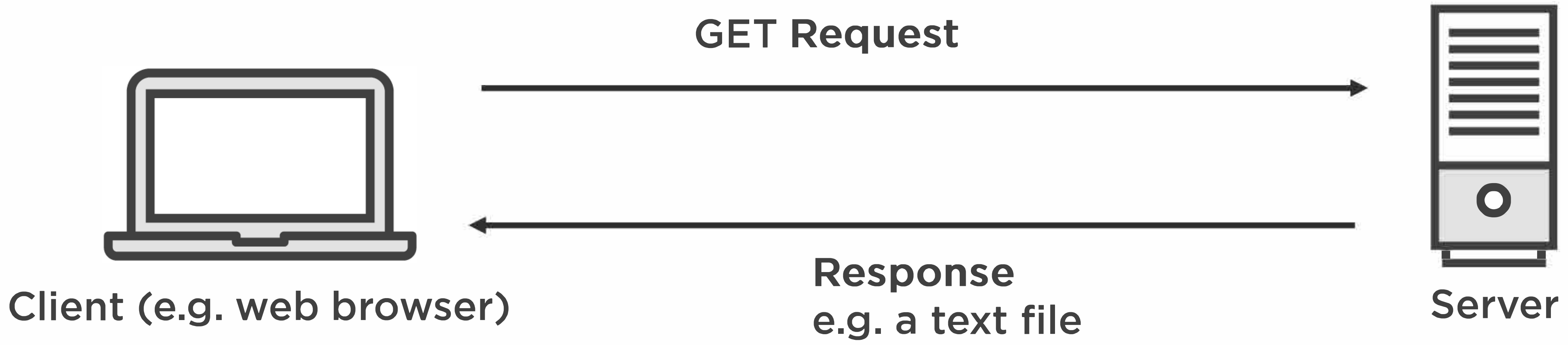
Create Web API tests using Java 11 HttpClient

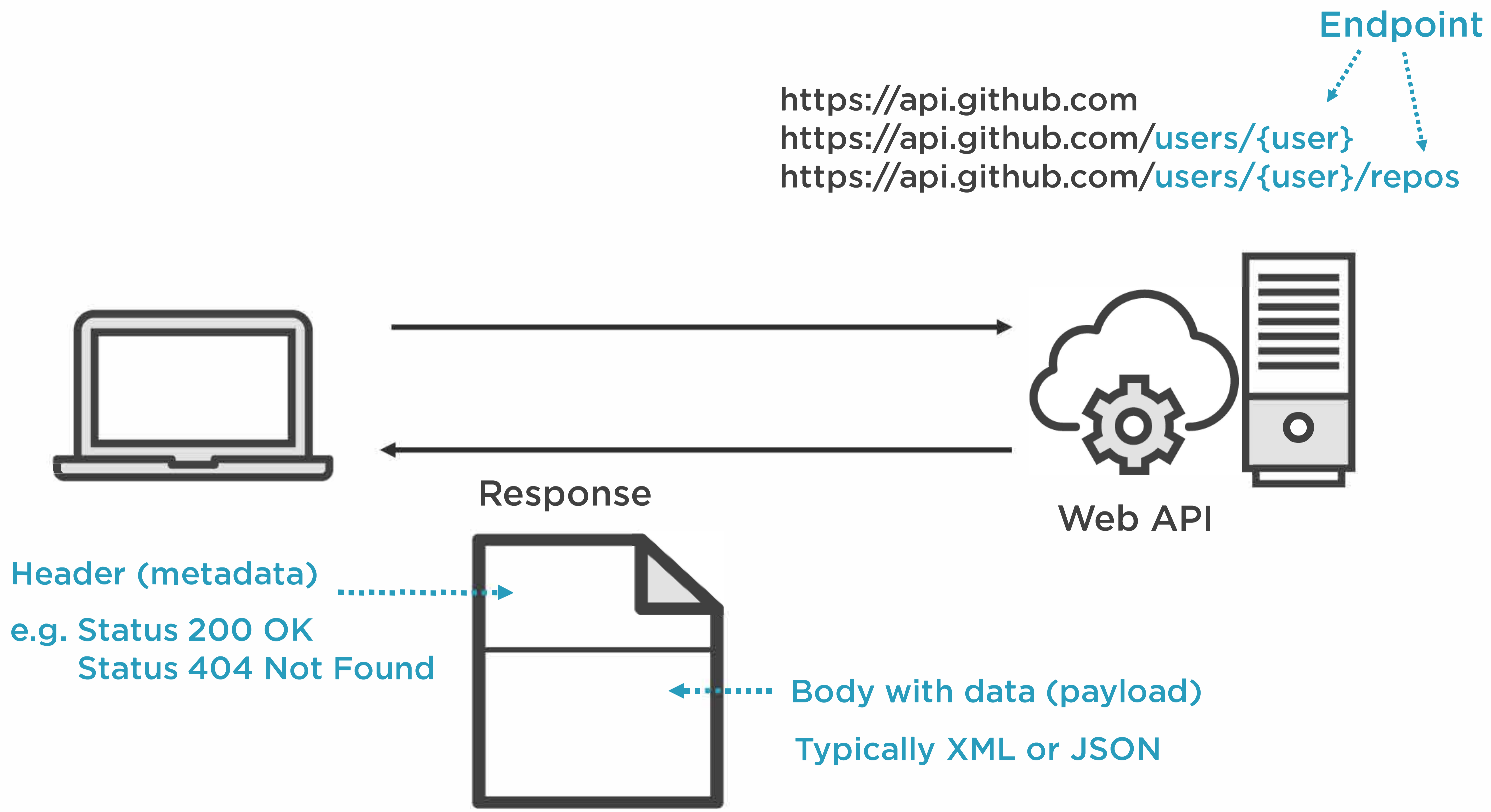
Refactor and improve

Explore an alternative solution

Web API and HTTP Refresher

HTTP is the foundation of data communication for the World Wide Web


GET Request
Client (e.g. web browser)
Response
e.g. a text file
Server


Endpoint
https://api.github.com
https://api.github.com/users/{user}
https://api.github.com/users/{user}/repos
Web API
Response
Header (metadata)
e.g. Status 200 OK
Status 404 Not Found
Body with data (payload)
Typically XML or JSON

XML

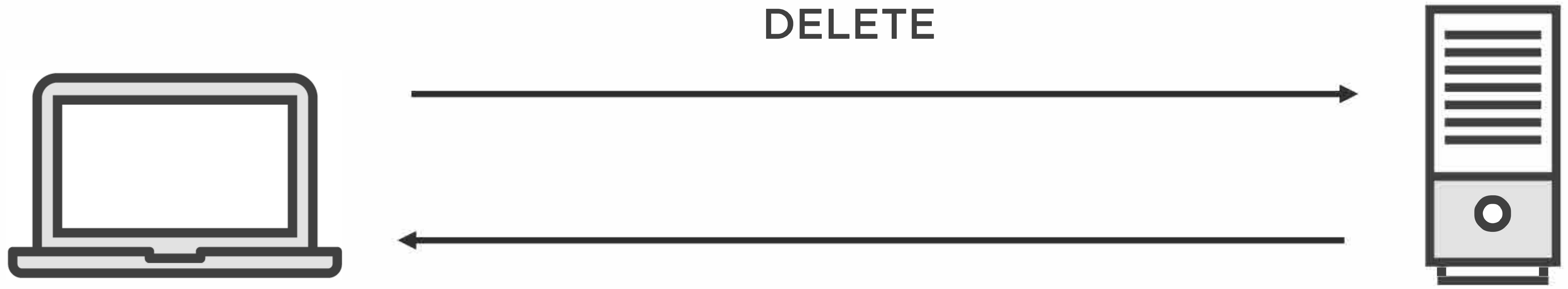
```
<root>
  <login>somename</login>
  <id>12345</id>
  <followers>14</followers>
</root>
```

JSON

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

Other widespread HTTP methods:

GET
POST (to create)
PUT (to update)
DELETE



Web API Test Scenarios

Send a GET to a valid endpoint and verify headers

Send a POST without authorization and verify that it gets rejected (headers)

Send a GET to a valid endpoint and verify the body

Java 11 HttpClient Overview

HttpClient

send

sendAsync

newBuilder

...

HttpRequest

uri

headers

method

newBuilder

...

HttpResponse

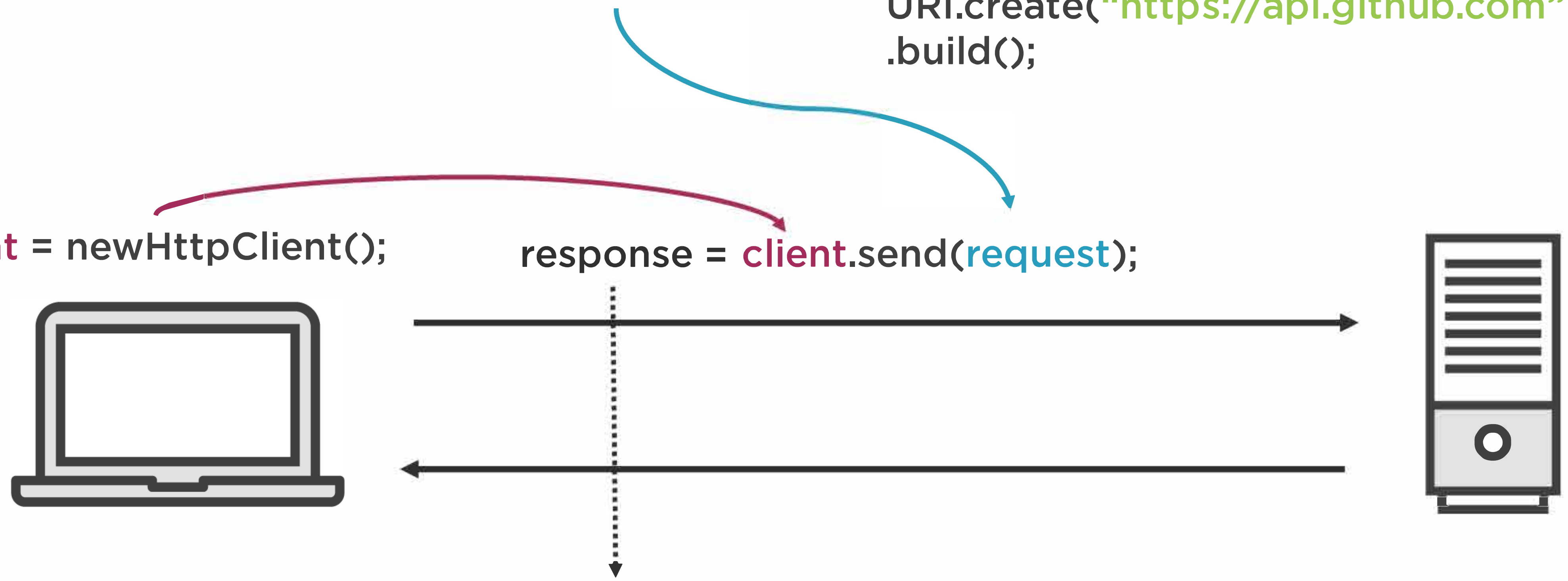
headers

statusCode

body

...

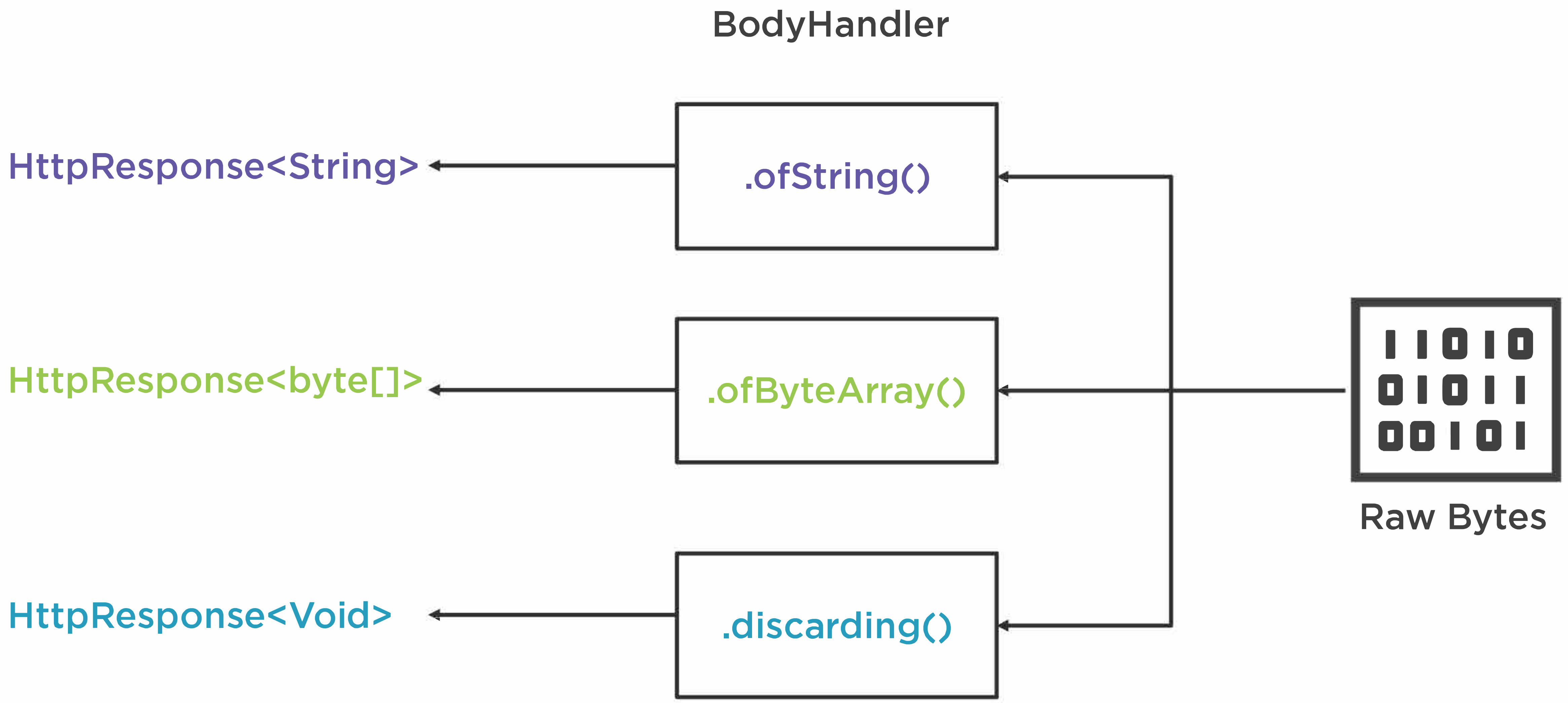
```
request =  HttpRequest.newBuilder(
                   URI.create("https://api.github.com"))
                   .build();
```



```
client = newHttpClient();
```

```
response = client.send(request);
```

```
int code       = response.statusCode();
Headers h      = response.headers();
String body = response.body();
```


BodyHandler
HttpResponse<String>
.ofString()
HttpResponse<byte[]>
.ofByteArray()
HttpResponse<Void>
.discarding()
Raw Bytes

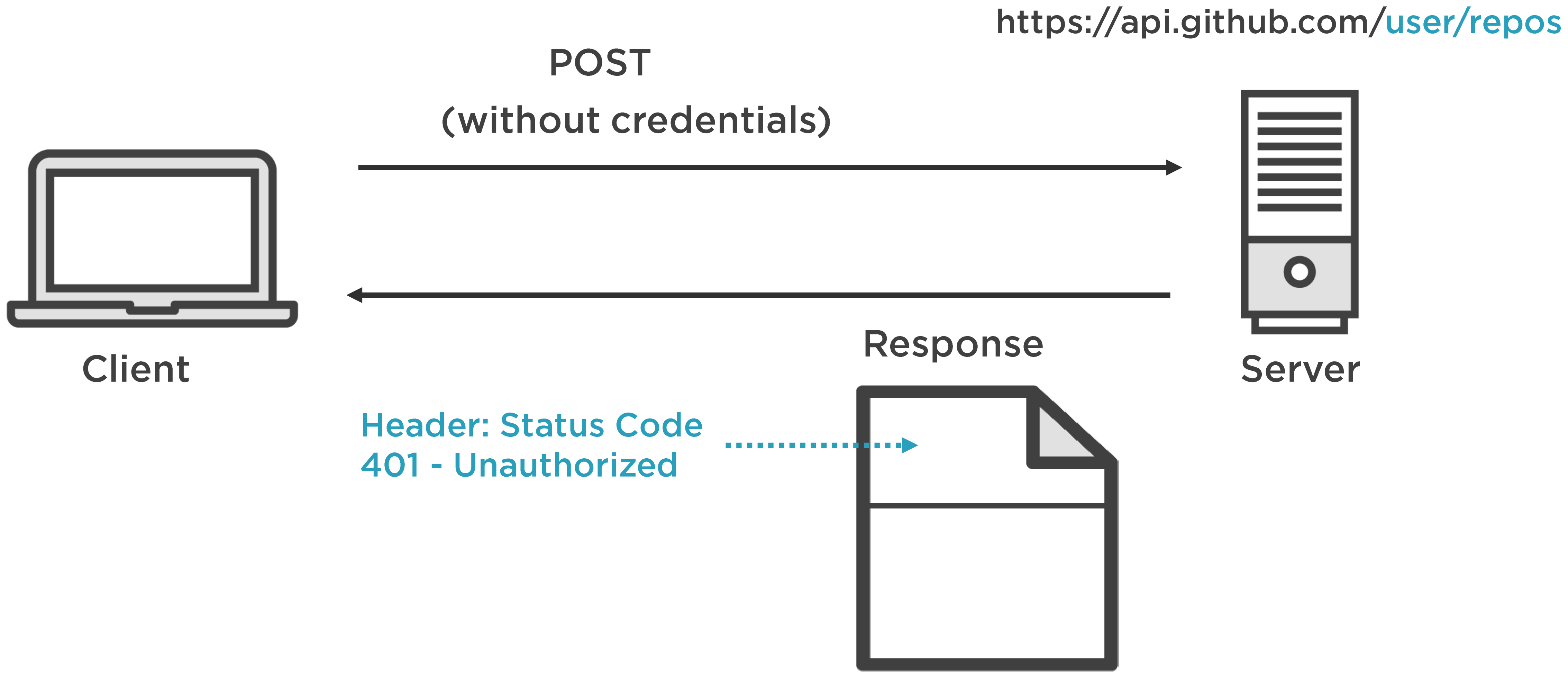
JUnit 5

```
@ParameterizedTest
@CsvSource({"A,1", "B, 2"})
void someTest(String input, String output) {
     String exp = doSomething(input);
     assertEquals(exp, output);
 }
```

TestNG

```
@DataProvider
Object[][] myInputProvider() {
    return new Object[][] {
        {"val1"},
        {"val2"},
        {"val3"}};}

@Test(dataProvider=“myInputProvider”)
void verifyX(String param1){
    doStuff(param1);
}
```


https://api.github.com/user/repos
POST
(without credentials)
Client
Server
Response
Header: Status Code
401 - Unauthorized

Testing the Response Body

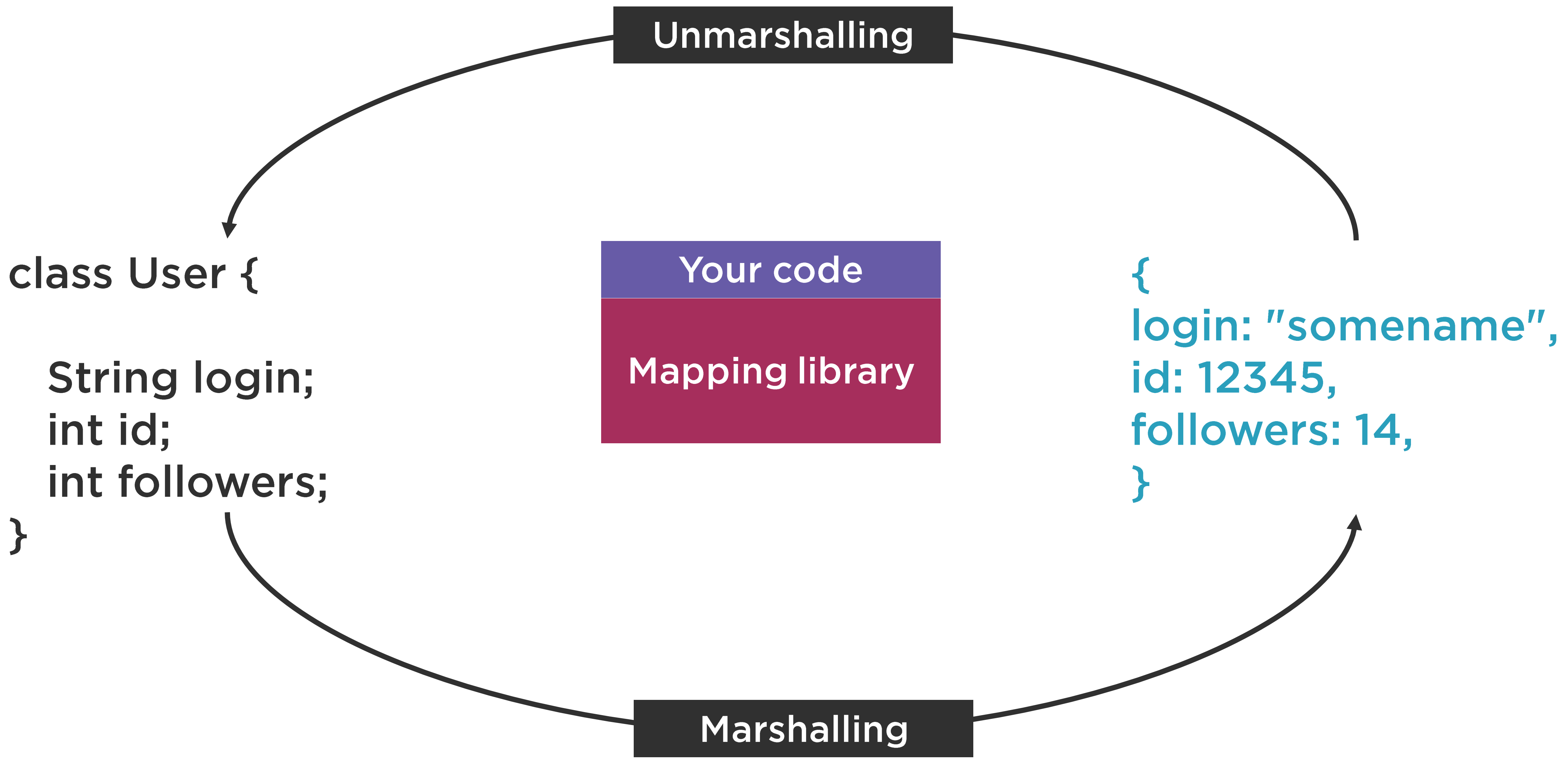
The quick way

String parsing

The right way

Object Mapping

String parsing is unstable, error-prone and difficult to maintain



```
class User {

   String login;
   int id;
   int followers;
}
```

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

HTTP Body Testing

We already have a course on Object Mapping in the context of Web API testing:

- Getting Started with Web API Test Automation in Java

Exploring Alternatives

What we've chosen	Alternatives
JUnit 5	TestNG
Selenium	Other UI tools
Java 11 HttpClient	???

Pre-Java 8

Apache HttpClient

Java 8+

okHttp

RestAssured

Java 11

HttpClient (native)

RestAssured

```
given().
    param("k1", "v1").

when().
    post("/somewhere").


then().
   body(containsString("OK"))
```

If RestAssured is so great, why learn other technologies?

RestAssured vs. DIY

Pros	Cons
Fast to get started with	Keeps your knowledge superficial
Easy to use	

I can write automated tests for Web APIs

Great! But we don’t use RestAssured in our project

...

Knowledge of tools and libraries can only take you so far

(and they are not a substitute for development skills)

Try out Yourself

Send a GET with plain http

Send HEAD, OPTIONS and other methods

- Use .method() on the Builder

Generate a GitHub Web Token (manually) and POST something

Send a DELETE

Further Study and Materials

Courses:

- Java Fundamentals: HttpClient

Dummy Web Services:

- http://dummy.restapiexample.com/
- https://jsonplaceholder.typicode.com/

Summary

HTTP and Web API recap:

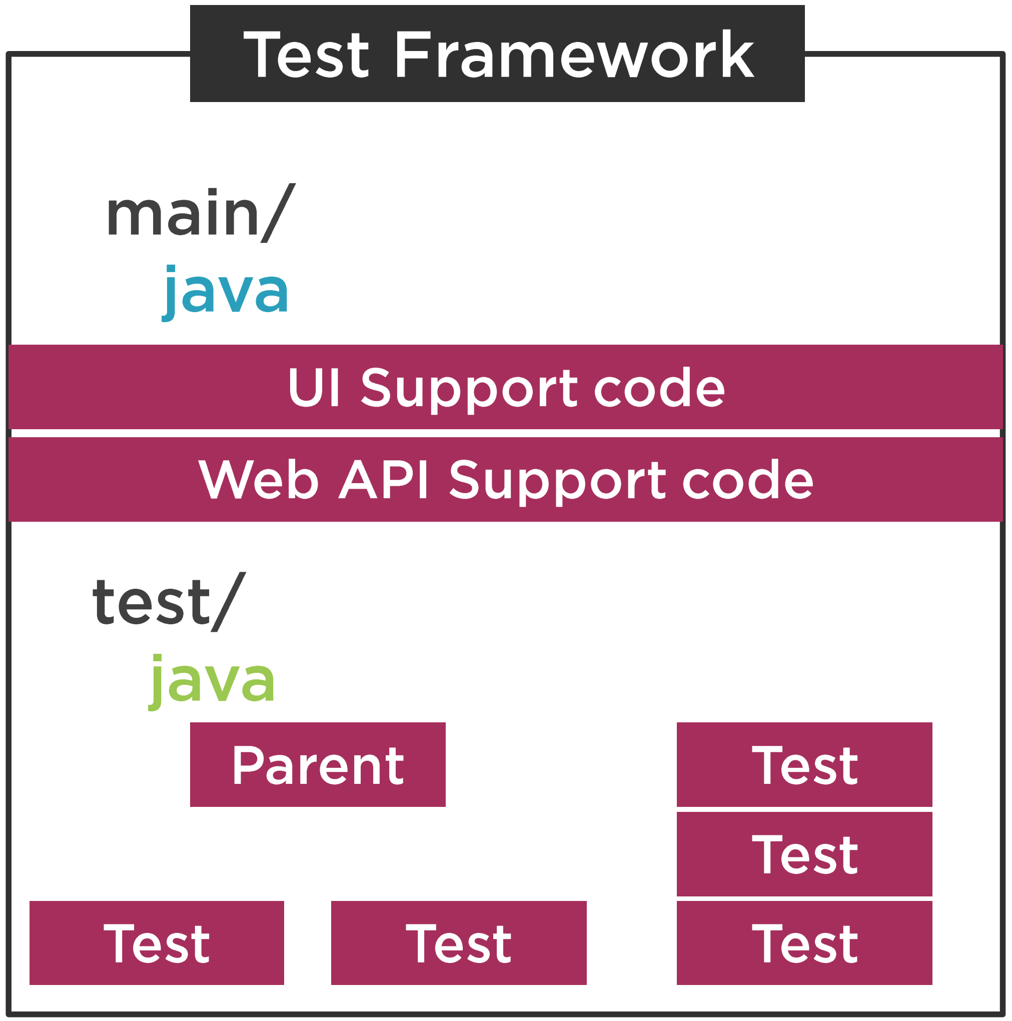
- Client/Server, Request/Response
- HTTP Methods, Header and Body
- Payload, JSON

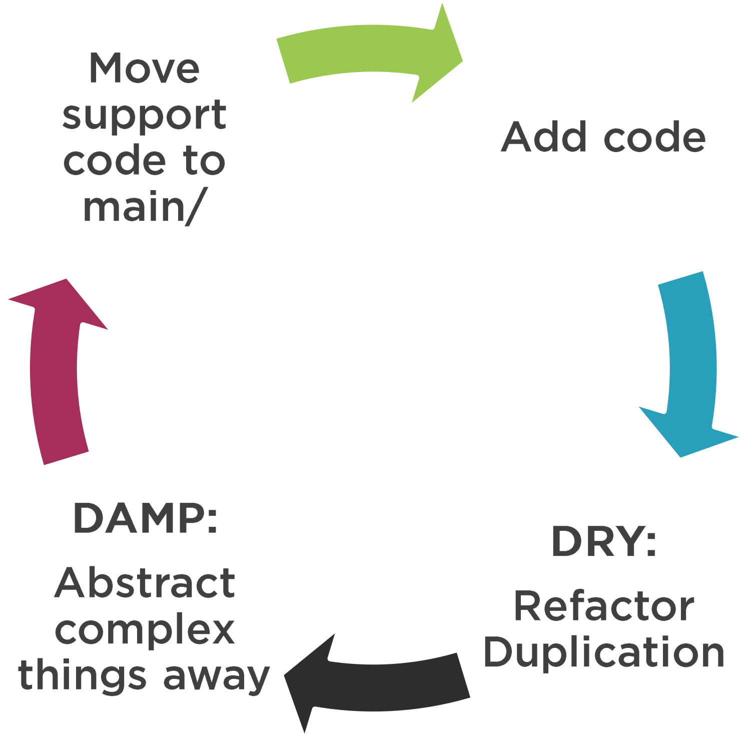
Testing – status codes, header values, body content

Developed support code under main:

- POJOs and BodyHandler for unmarshalling

Explored and evaluated an alternative solution


Test Framework
main/
java
UI Support code
Web API Support code
test/
java
Parent
Test
Test
Test
Test
Test


Move support code to main/
Add code
DRY:
Refactor Duplication
DAMP:
Abstract complex things away

Up next:
Scaling to a multi-module framework