RestTemplate

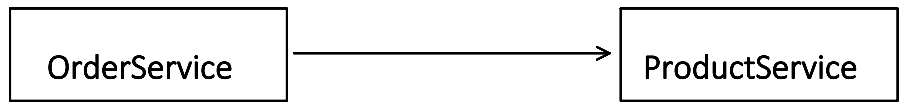
Tuesday, 13 May 2025 3:20 PM

<u>In this topic today, will cover:</u>

1st : Set up of 2 microservice i.e. "OrderService" and "ProductService" running on different port numbers.

OrderService
Port: 8081
ProductService
Port: 8082

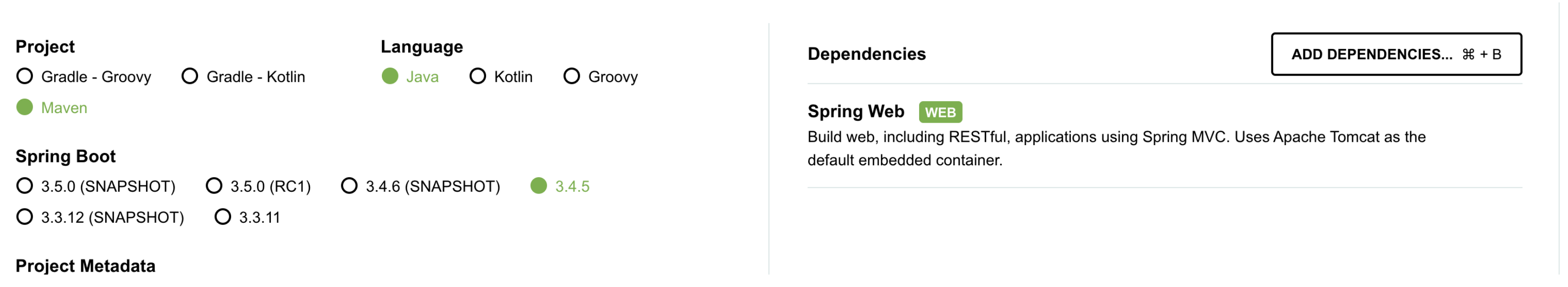
2nd : How two microservices can communicate with each other



<u>Let's start:</u>

1st : Set up of 2 microservice

OrderService Go to Spring Initializer (start.spring.io)



Project
Gradle - Groovy / Gradle - Kotlin / [✓ Maven]

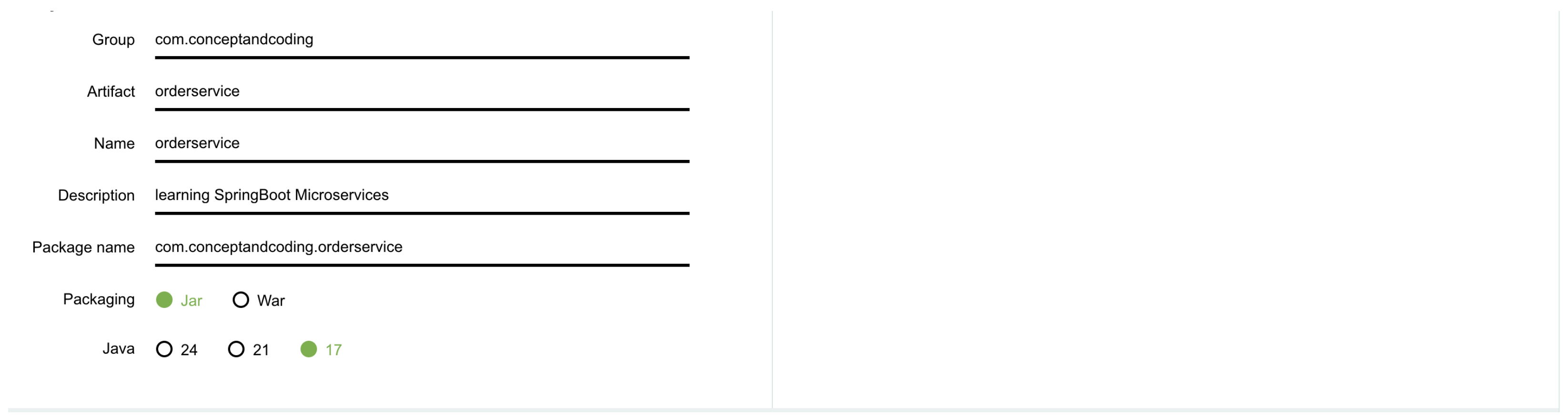
Language
[✓ Java] / Kotlin / Groovy

Spring Boot
3.5.0 (SNAPSHOT) / 3.5.0 (RC1) / 3.4.6 (SNAPSHOT) / [✓ 3.4.5] / 3.3.12 (SNAPSHOT) / 3.3.11

Project Metadata

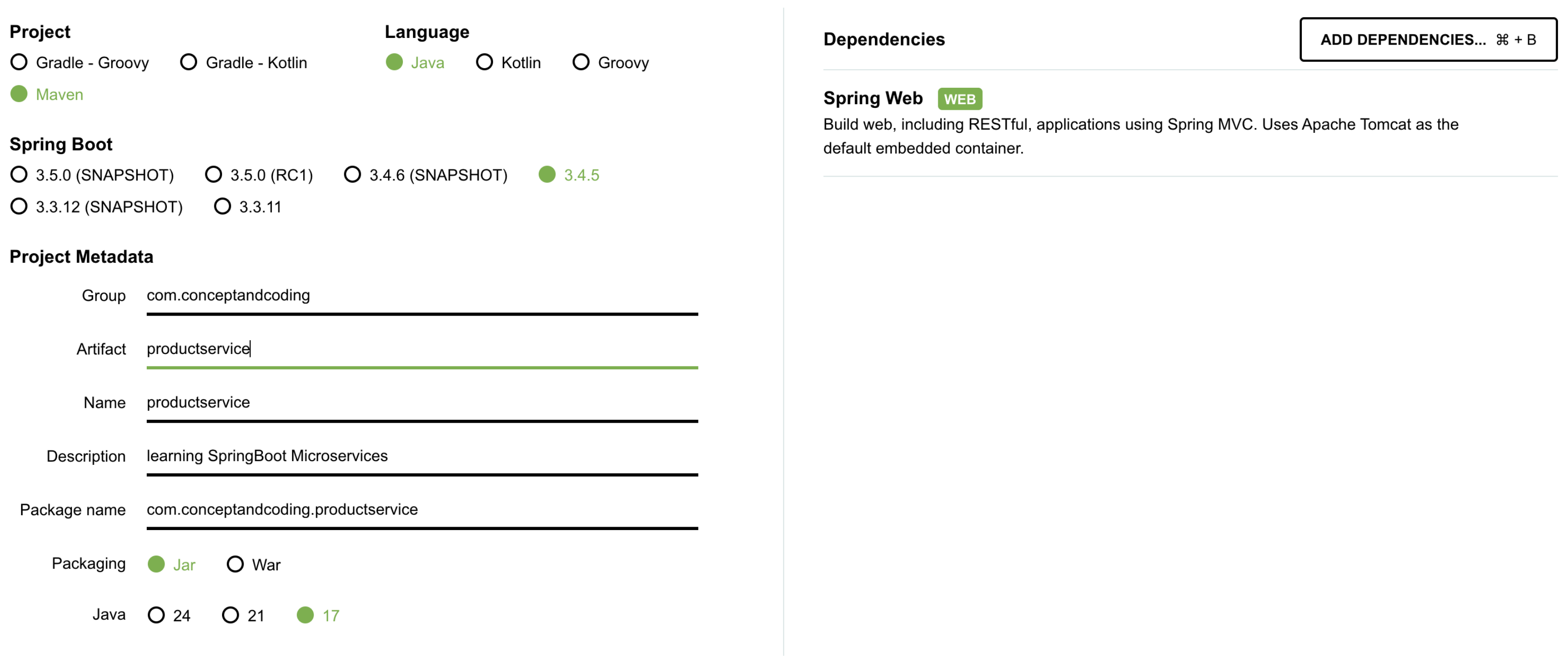
Dependencies ADD DEPENDENCIES... ⌘ + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



Group	com.conceptandcoding
Artifact	orderservice
Name	orderservice
Description	learning SpringBoot Microservices
Package name	com.conceptandcoding.orderservice
Packaging	[✓ Jar] War
Java	24 21 [✓ 17]

Similarly, set up ProductService



Project
Gradle - Groovy Gradle - Kotlin [✓ Maven]

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.5.0 (SNAPSHOT) 3.5.0 (RC1) 3.4.6 (SNAPSHOT) [✓ 3.4.5] 3.3.12 (SNAPSHOT) 3.3.11

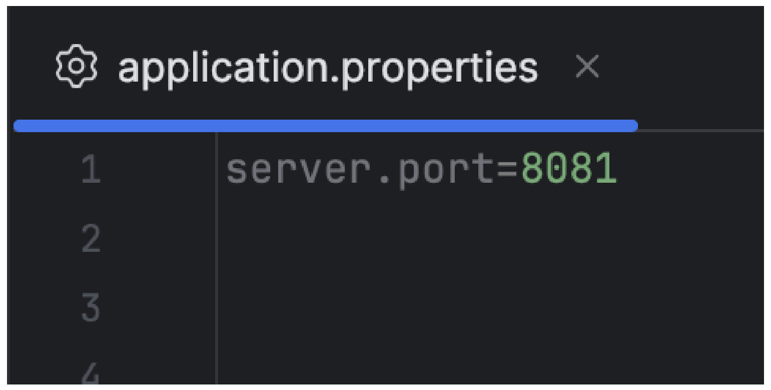
Project Metadata

Group	com.conceptandcoding
Artifact	productservice
Name	productservice
Description	learning SpringBoot Microservices
Package name	com.conceptandcoding.productservice
Packaging	[✓ Jar] War
Java	24 21 [✓ 17]

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

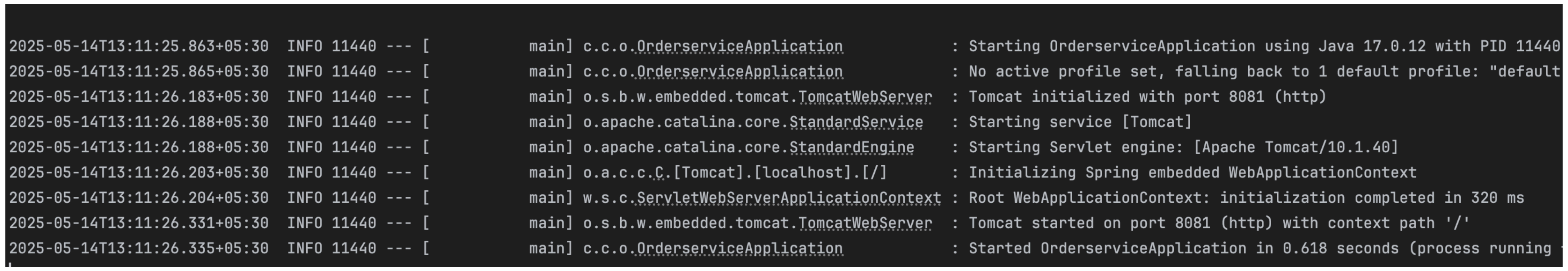
OrderService



application.properties

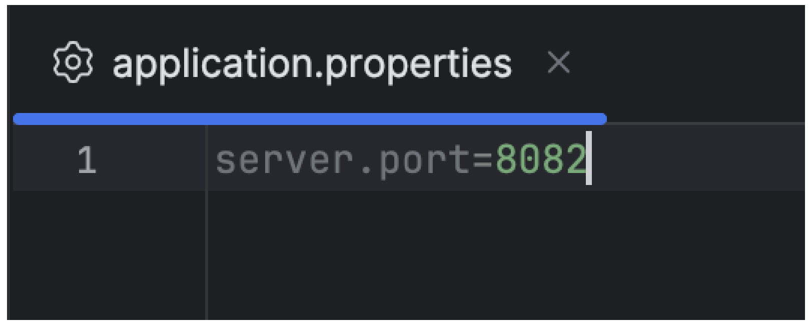
```
server.port=8081
```

```
 :: Spring Boot ::                (v3.4.5)
```



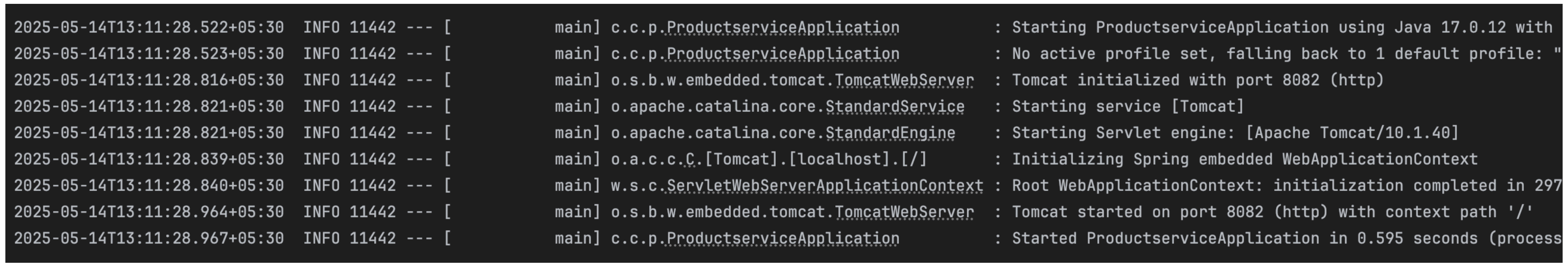
```
2025-05-14T13:11:25.863+05:30  INFO 11440 --- [           main] c.c.o.OrderserviceApplication            : Starting OrderserviceApplication using Java 17.0.12 with PID 11440
2025-05-14T13:11:25.865+05:30  INFO 11440 --- [           main] c.c.o.OrderserviceApplication            : No active profile set, falling back to 1 default profile: "default
2025-05-14T13:11:26.183+05:30  INFO 11440 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8081 (http)
2025-05-14T13:11:26.188+05:30  INFO 11440 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-05-14T13:11:26.188+05:30  INFO 11440 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.40]
2025-05-14T13:11:26.203+05:30  INFO 11440 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-05-14T13:11:26.204+05:30  INFO 11440 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 320 ms
2025-05-14T13:11:26.331+05:30  INFO 11440 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8081 (http) with context path '/'
2025-05-14T13:11:26.335+05:30  INFO 11440 --- [           main] c.c.o.OrderserviceApplication            : Started OrderserviceApplication in 0.618 seconds (process running
```

ProductService



application.properties

```
server.port=8082
```



```
2025-05-14T13:11:28.522+05:30  INFO 11442 --- [           main] c.c.p.ProductserviceApplication          : Starting ProductserviceApplication using Java 17.0.12 with
2025-05-14T13:11:28.523+05:30  INFO 11442 --- [           main] c.c.p.ProductserviceApplication          : No active profile set, falling back to 1 default profile: "
2025-05-14T13:11:28.816+05:30  INFO 11442 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8082 (http)
2025-05-14T13:11:28.821+05:30  INFO 11442 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-05-14T13:11:28.821+05:30  INFO 11442 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.40]
2025-05-14T13:11:28.839+05:30  INFO 11442 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-05-14T13:11:28.840+05:30  INFO 11442 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 297
2025-05-14T13:11:28.964+05:30  INFO 11442 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8082 (http) with context path '/'
2025-05-14T13:11:28.967+05:30  INFO 11442 --- [           main] c.c.p.ProductserviceApplication          : Started ProductserviceApplication in 0.595 seconds (process
```

$$\begin{tikzcd}
 & \text{2nd : Communication between 2 services} \arrow[dl] \arrow[dr] & \\
\text{Synchronous} & & \text{Asynchronous}
\end{tikzcd}$$

In this part, will focus on Synchronous communication

Synchronous Communication :

- Client wait for the response from the Server before continuing.
- Blocking in nature, means thread waits till response is not received.

$$\begin{tikzcd}
 & \text{Synchronous Communication Types in SpringBoot} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{RestTemplate} & \text{RestClient} & \text{FeignClient}
\end{tikzcd}$$

<u>Sample HTTP GET Request call:</u>

HTTP method *Path(URI)* *Protocol version*

GET /orders/1 HTTP/1.1
Host: localhost:8081 → *Target host*

User-Agent: curl/8.7.1 → *Which client/tool is used to make the request*

Accept: application/json → *Format in which client want response*

<u>Sample HTTP POST Request call:</u>

```
POST /products HTTP/1.1
Host: localhost:8081
User-Agent: curl/8.7.1
Accept: application/json
Content-Type: application/json        → Tell server that, request body is JSON
Content-Length: 65                    → Tells the number of bytes in the request body
{
 "name": "Ice-Cream",                 → Actual data, a new product object
 "price": 200
}
```

Sample HTTP GET Response call, with keep alive:

2xx : request is successful
4xx: client sent invalid request
5xx: Server error

```
HTTP/1.1 200 OK
Date: Fri, 16 May 2025 10:00:00 GMT
Content-Type: application/json
Content-Length: 65
Connection: keep-alive
Keep-alive: timeout=5, max=50
{
 "id": 10,
 "name": "SJ"
}
```

Tells client to expect JSON response

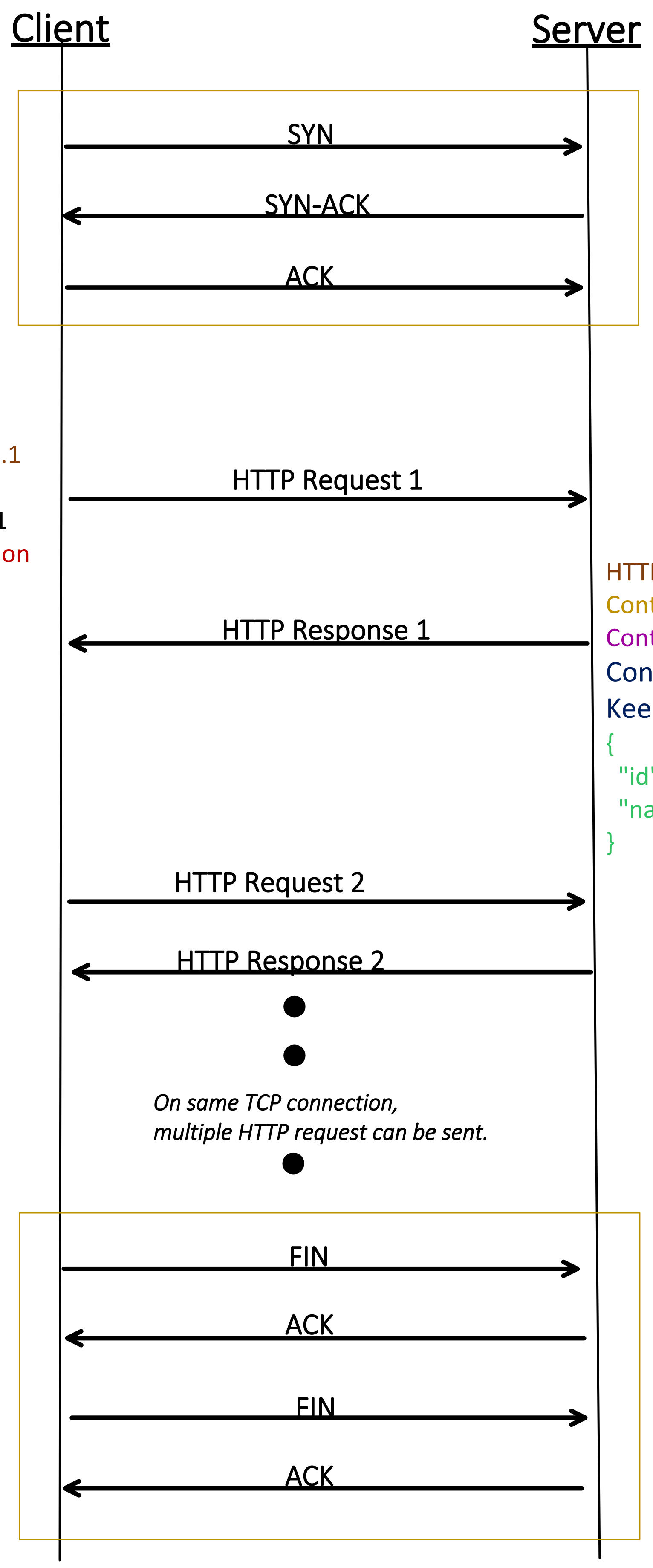
No of bytes in response body

Response body

- By default, connection is set to keep-alive in HTTP/1.1
- In HTTP/1.0 by-default connection is set to close.
- Keep-alive:
  - timeout=5, tells close the TCP connection if its idle for 5 seconds
  - Max=50, tells the maximum number of requests can be send over same TCP connection.

- When Connection: close is set, it tells after every response from the server, TCP connection is closed, its not reused.

Flow, when keep-alive is set:



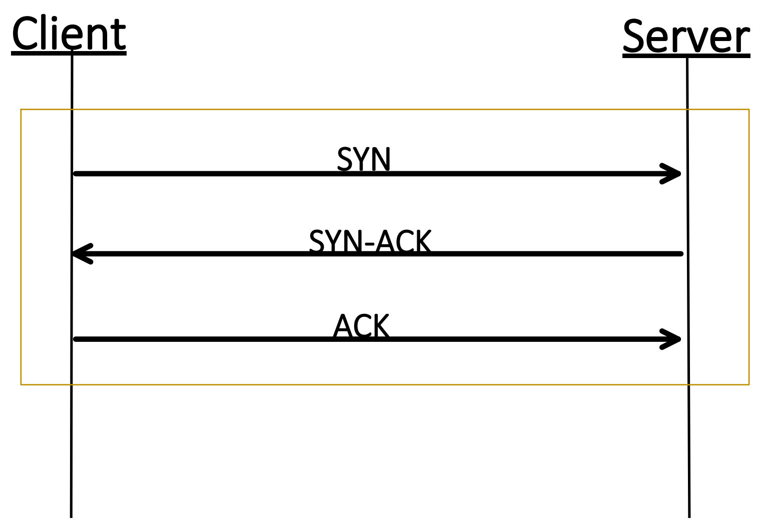
3 way handshake,
Establishes a TCP connection between Client and Server

```
GET /orders/1 HTTP/1.1
 Host: localhost:8081
 User-Agent: curl/8.7.1
 Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 65
Connection: keep-alive
Keep-alive: timeout=5, max=50
{
  "id": 10,
  "name": "SJ"
}
```

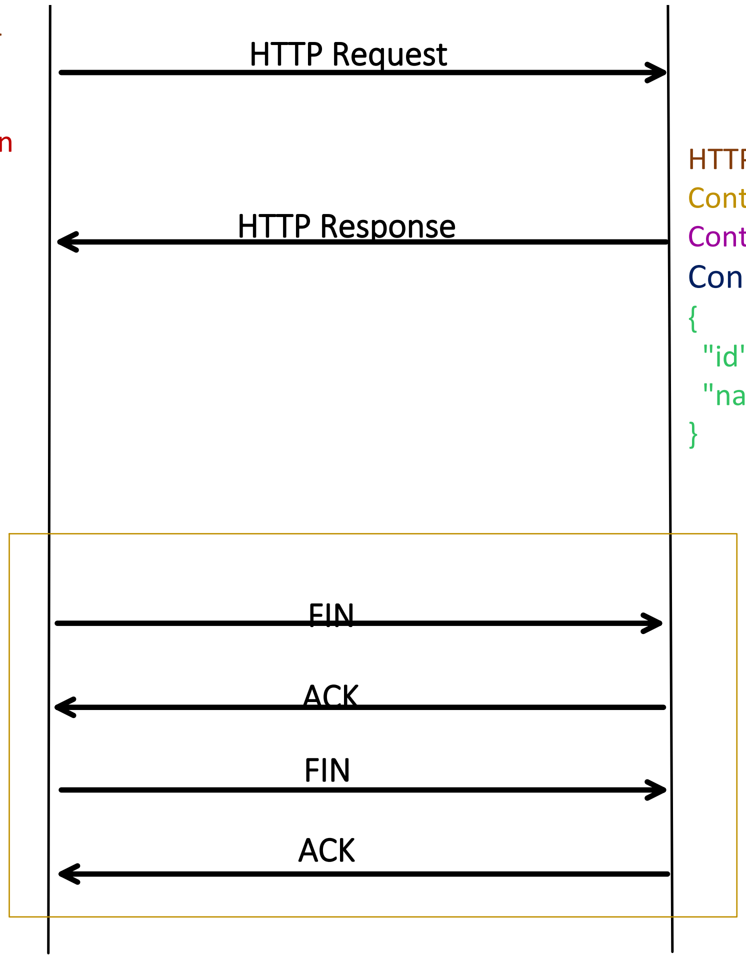
4 way Termination to close the TCP connection
(any one client or server can initiate the close request)

Flow, when connection: close is set:



3 way handshake,
Establishes a TCP connection between Client and Server

```
GET /orders/1 HTTP/1.1
Host: localhost:8081
User-Agent: curl/8.7.1
Accept: application/json
```



```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 65
Connection: close
{
  "id": 10,
  "name": "SJ"
}
```

4 way Termination to close the TCP connection, after each response
(any one client or server can initiate the close request)

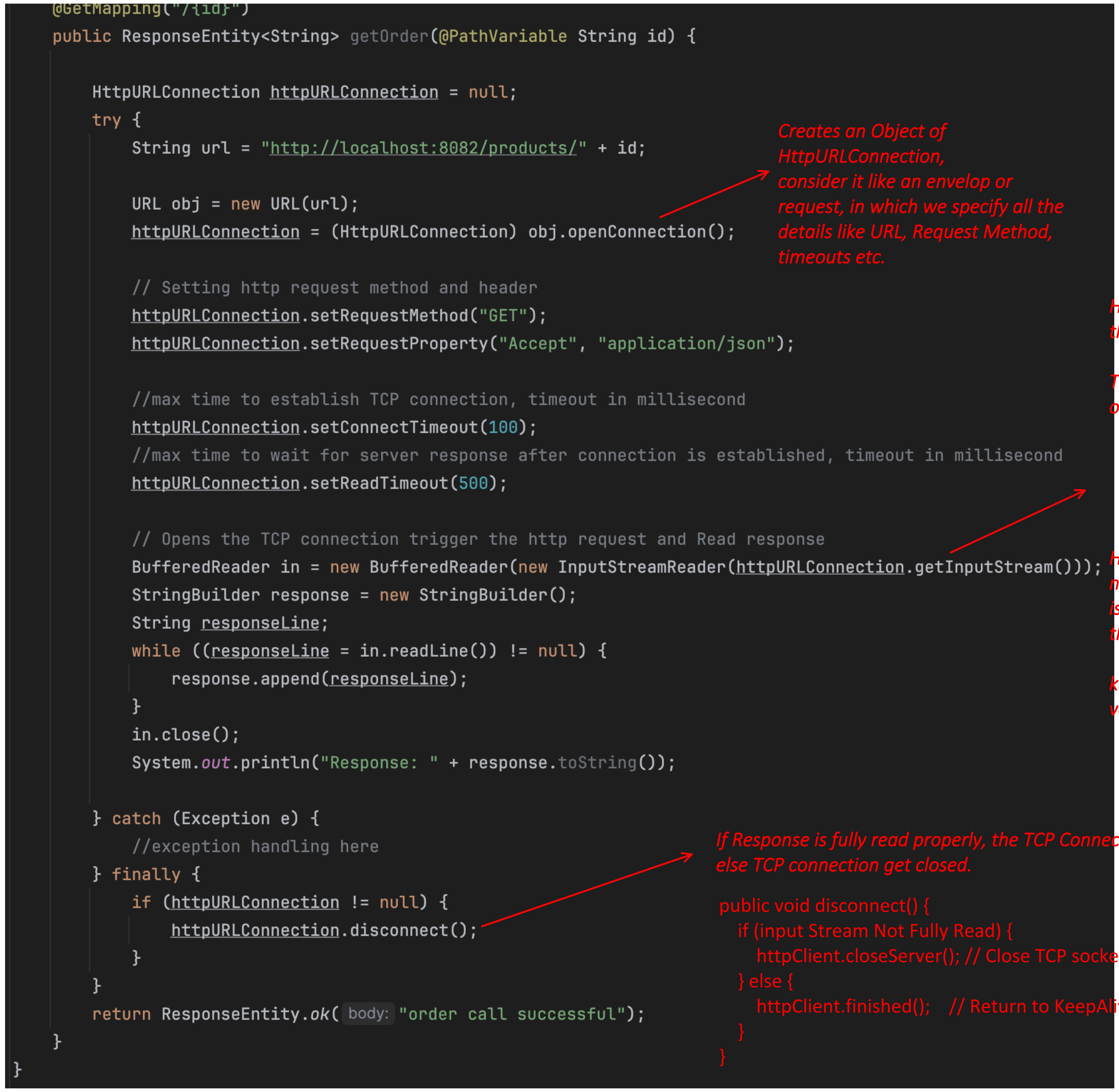
Let's first see, without using above SpringBoot communication types, what it takes to invoke the REST endpoint just using plain JAVA.

OrderService

```
@RestController
@RequestMapping("/orders")
public class OrderController {
```

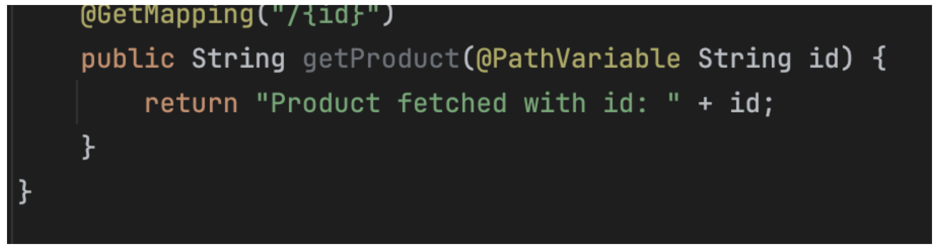
ProductService

```
@RestController
@RequestMapping("/products")
public class ProductController {
```



```java
    @GetMapping("/{id}")
    public ResponseEntity<String> getOrder(@PathVariable String id) {

        HttpURLConnection httpURLConnection = null;
        try {
            String url = "http://localhost:8082/products/" + id;

            URL obj = new URL(url);
            httpURLConnection = (HttpURLConnection) obj.openConnection();

            // Setting http request method and header
            httpURLConnection.setRequestMethod("GET");
            httpURLConnection.setRequestProperty("Accept", "application/json");

            //max time to establish TCP connection, timeout in millisecond
            httpURLConnection.setConnectTimeout(100);
            //max time to wait for server response after connection is established, timeout in millisecond
            httpURLConnection.setReadTimeout(500);

            // Opens the TCP connection trigger the http request and Read response
            BufferedReader in = new BufferedReader(new InputStreamReader(httpURLConnection.getInputStream()));
            StringBuilder response = new StringBuilder();
            String responseLine;
            while ((responseLine = in.readLine()) != null) {
                response.append(responseLine);
            }
            in.close();
            System.out.println("Response: " + response.toString());

        } catch (Exception e) {
            //exception handling here
        } finally {
            if (httpURLConnection != null) {
                httpURLConnection.disconnect();
            }
        }
        return ResponseEntity.ok( body: "order call successful");
    }
}
```

Creates an Object of HttpURLConnection, consider it like an envelop or request, in which we specify all the details like URL, Request Method, timeouts etc.



```java
    @GetMapping("/{id}")
    public String getProduct(@PathVariable String id) {
        return "Product fetched with id: " + id;
    }
}
```

Here it opens up a TCP connection and send the HTTP request, also reads the response.

TCP connection is created, when we make the first Input/Output request on HttpURLConnection such as:

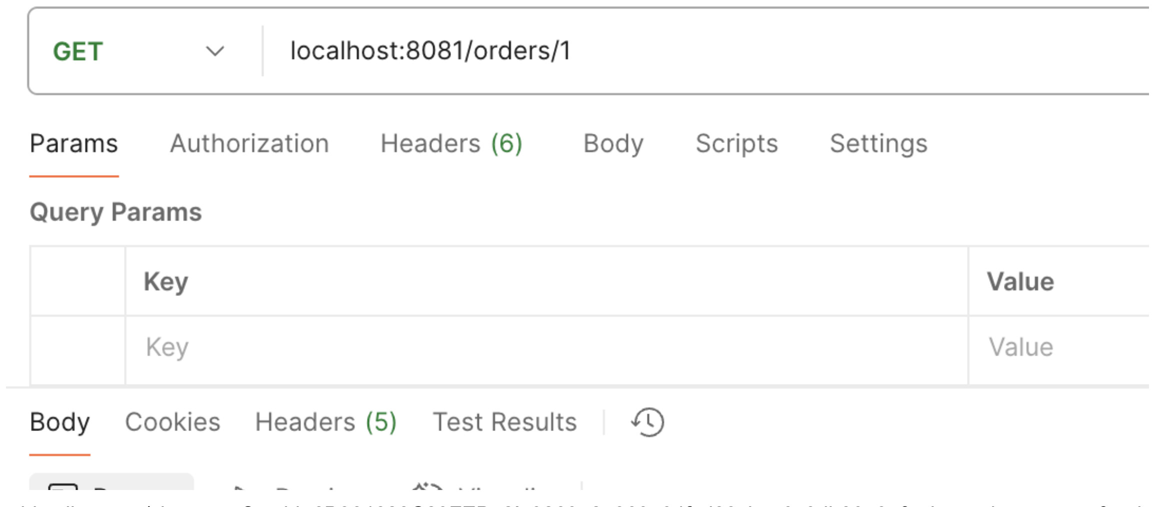
- *getInputStream()*
- *getResponseCode()*
- *connect() etc.…*

HttpClient object is a wrapper around TCP connection, so before creating new HttpClient object, if first checks with "KeepAliveCache" class, if there is already an object present, if not it creates one object and also puts into the cache.

key -> host:port
value -> httpClient object

If Response is fully read properly, the TCP Connection i.e. HttpClient is returned back to KeepAlive cache else TCP connection get closed.

```
public void disconnect() {
   if (input Stream Not Fully Read) {
      httpClient.closeServer(); // Close TCP socket
   } else {
      httpClient.finished();    // Return to KeepAlive Cache
   }
}
```



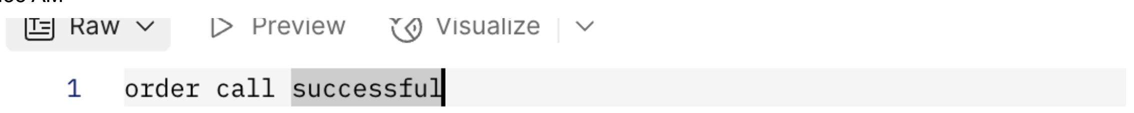
GET localhost:8081/orders/1

Params | Authorization | Headers (6) | Body | Scripts | Settings

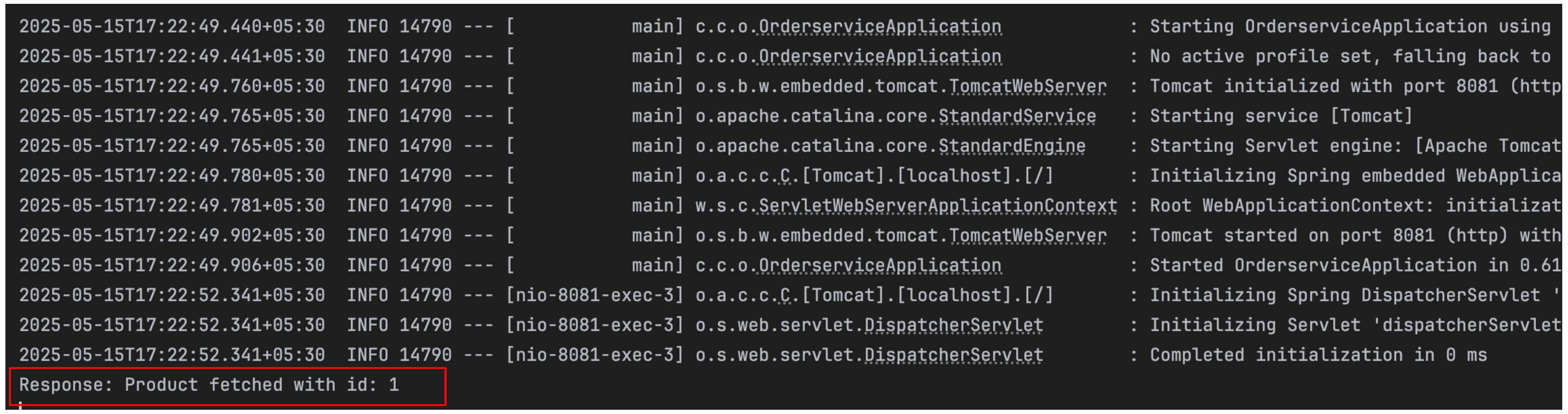
Query Params

	Key	Value
	Key	Value

Body | Cookies | Headers (5) | Test Results



```
order call successful
```



```
2025-05-15T17:22:49.440+05:30  INFO 14790 --- [           main] c.c.o.OrderserviceApplication            : Starting OrderserviceApplication using
2025-05-15T17:22:49.441+05:30  INFO 14790 --- [           main] c.c.o.OrderserviceApplication            : No active profile set, falling back to
2025-05-15T17:22:49.760+05:30  INFO 14790 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8081 (http
2025-05-15T17:22:49.765+05:30  INFO 14790 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-05-15T17:22:49.765+05:30  INFO 14790 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat
2025-05-15T17:22:49.780+05:30  INFO 14790 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplica
2025-05-15T17:22:49.781+05:30  INFO 14790 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializat
2025-05-15T17:22:49.902+05:30  INFO 14790 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8081 (http) with
2025-05-15T17:22:49.906+05:30  INFO 14790 --- [           main] c.c.o.OrderserviceApplication            : Started OrderserviceApplication in 0.61
2025-05-15T17:22:52.341+05:30  INFO 14790 --- [nio-8081-exec-3] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet '
2025-05-15T17:22:52.341+05:30  INFO 14790 --- [nio-8081-exec-3] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet
2025-05-15T17:22:52.341+05:30  INFO 14790 --- [nio-8081-exec-3] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
Response: Product fetched with id: 1
```

Couple of Disadvantage of above approach is:

- Too much Boilerplate code:
  - Open connection
  - Setting headers
  - Reading response
  - Closing streams and connections.

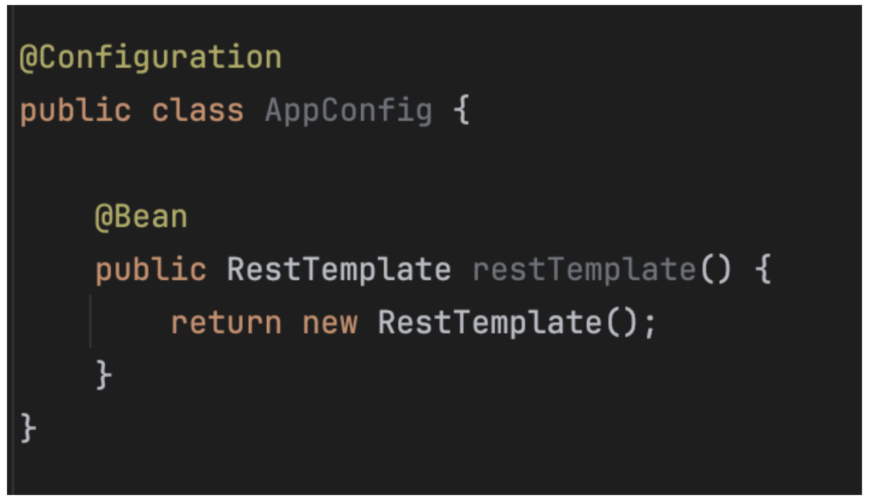
- Response should be handled manually.
  - No automatic mapping to some Objects.

- Limited support for Advance features like
  - Connection pooling
  - Interceptors etc.

RestTemplate

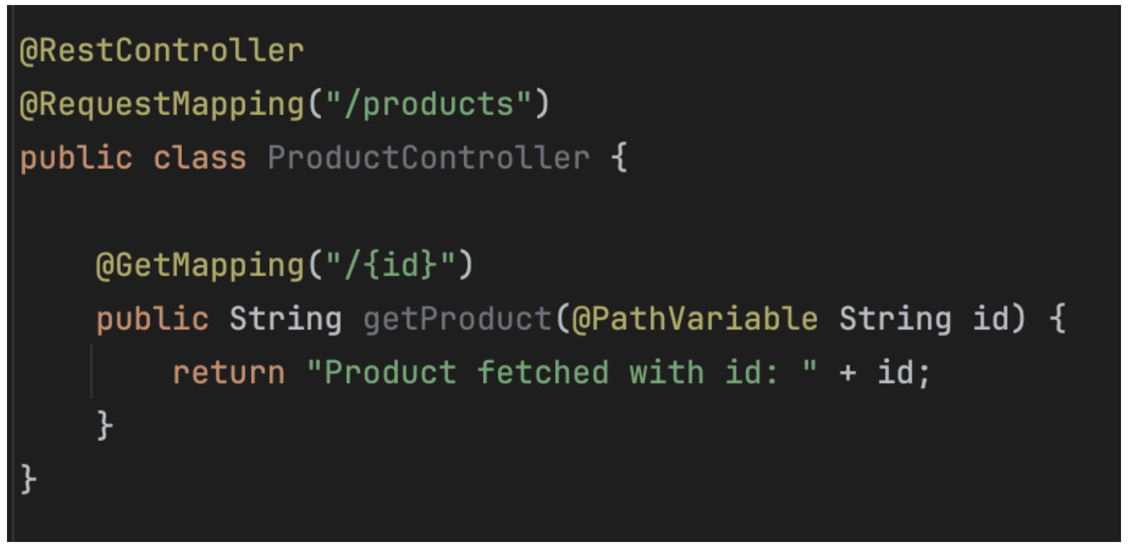
- Abstract low level code like creating HttpURLConnection object etc.
- Traditional/Legacy way to call REST APIs in Spring application.

OrderService



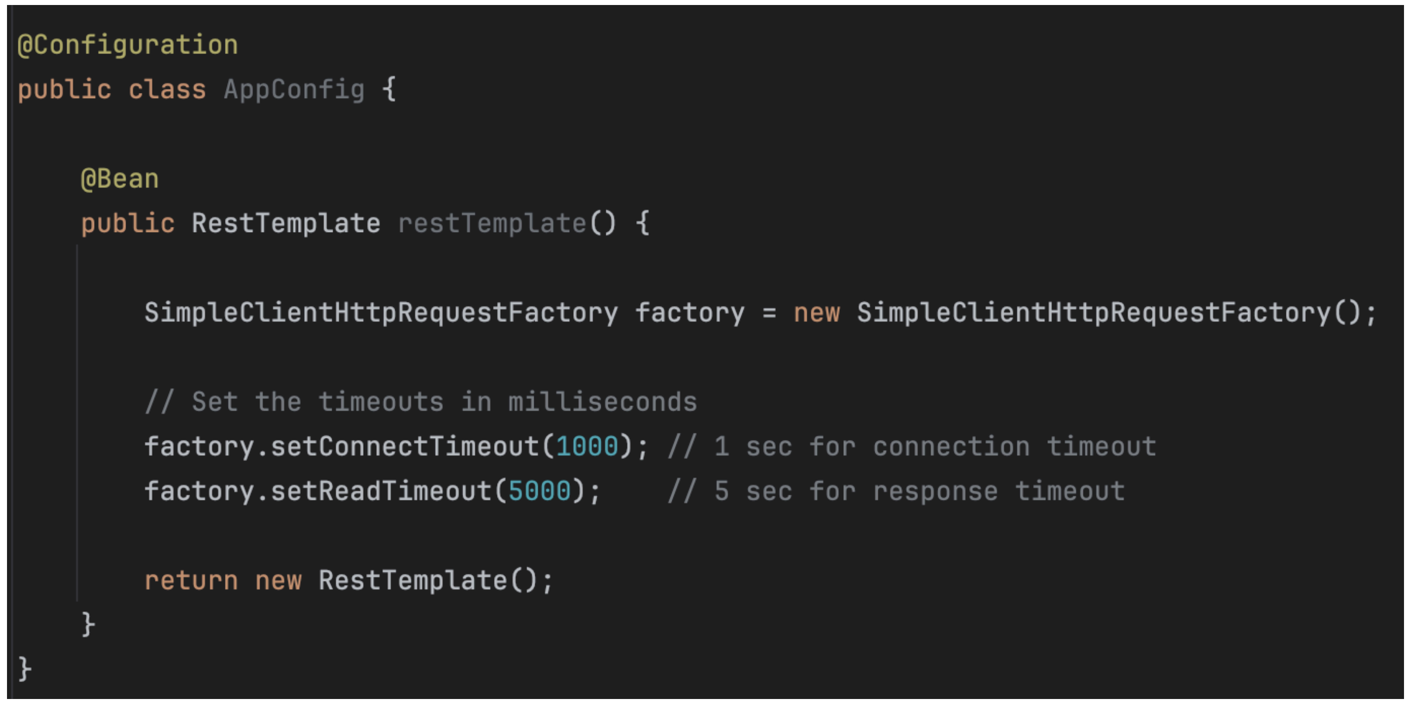
```java
@Configuration
public class AppConfig {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

ProductService

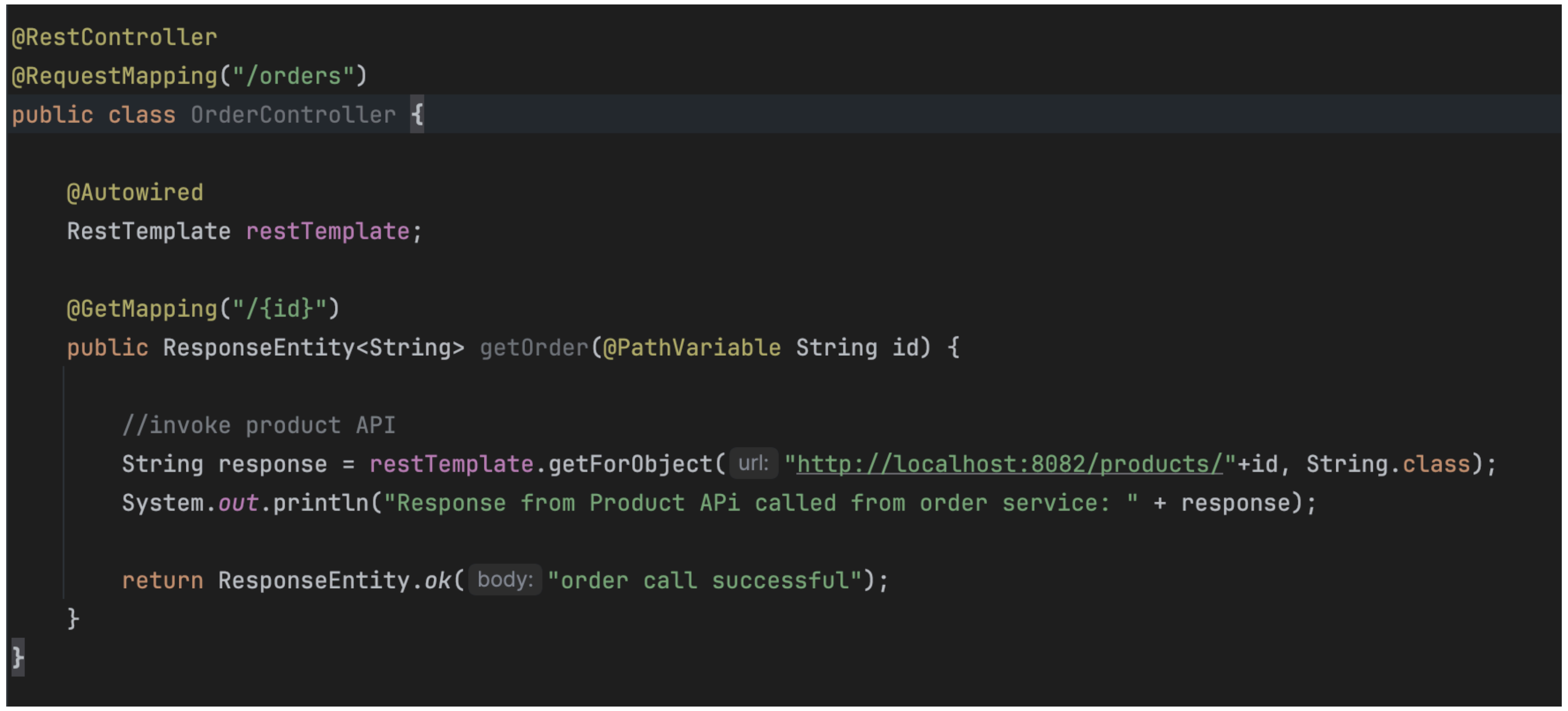


```java
@RestController
@RequestMapping("/products")
public class ProductController {

    @GetMapping("/{id}")
    public String getProduct(@PathVariable String id) {
        return "Product fetched with id: " + id;
    }
}
```

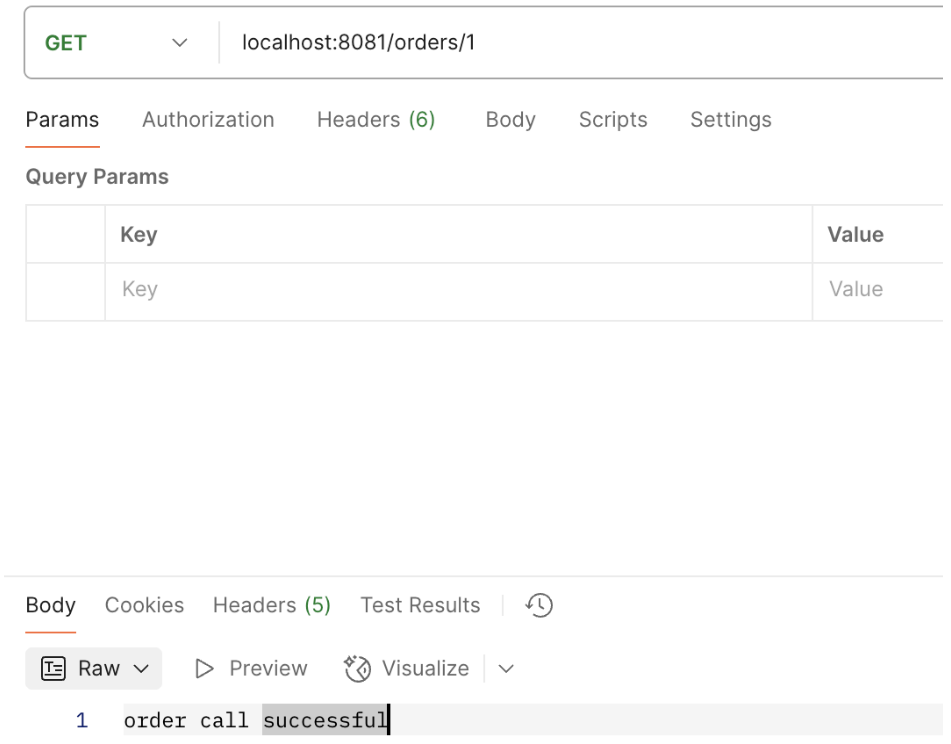
Or, use below, if we want to set timeouts too



```java
@Configuration
public class AppConfig {

    @Bean
    public RestTemplate restTemplate() {

        SimpleClientHttpRequestFactory factory = new SimpleClientHttpRequestFactory();

        // Set the timeouts in milliseconds
        factory.setConnectTimeout(1000); // 1 sec for connection timeout
        factory.setReadTimeout(5000);    // 5 sec for response timeout

        return new RestTemplate();
    }
}
```



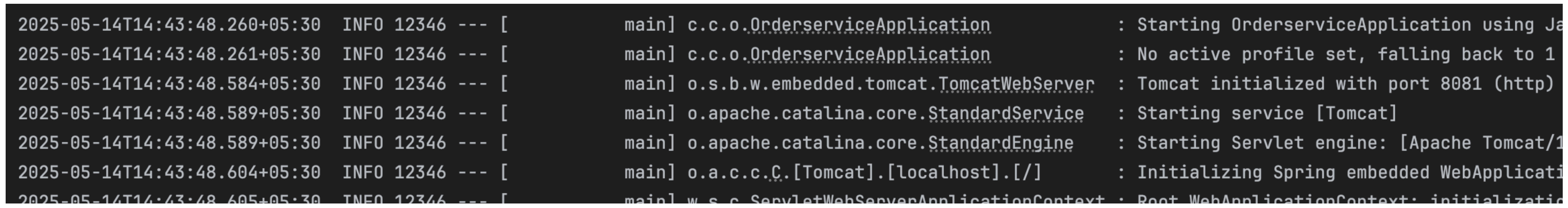
```java
@RestController
@RequestMapping("/orders")
public class OrderController {

    @Autowired
    RestTemplate restTemplate;

    @GetMapping("/{id}")
    public ResponseEntity<String> getOrder(@PathVariable String id) {

        //invoke product API
        String response = restTemplate.getForObject( url: "http://localhost:8082/products/"+id, String.class);
        System.out.println("Response from Product APi called from order service: " + response);

        return ResponseEntity.ok( body: "order call successful");
    }
}
```



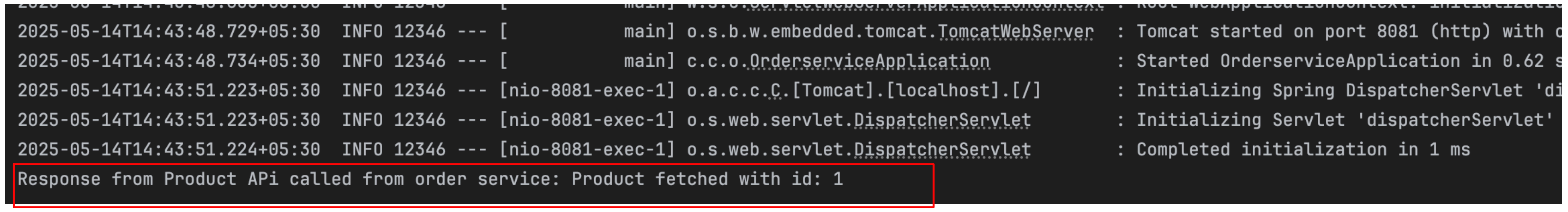
GET localhost:8081/orders/1

Key	Value
Key	Value

```
1  order call successful
```



```
2025-05-14T14:43:48.260+05:30  INFO 12346 --- [           main] c.c.o.OrderserviceApplication            : Starting OrderserviceApplication using Ja
2025-05-14T14:43:48.261+05:30  INFO 12346 --- [           main] c.c.o.OrderserviceApplication            : No active profile set, falling back to 1
2025-05-14T14:43:48.584+05:30  INFO 12346 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8081 (http)
2025-05-14T14:43:48.589+05:30  INFO 12346 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-05-14T14:43:48.589+05:30  INFO 12346 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/1
2025-05-14T14:43:48.604+05:30  INFO 12346 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicati
2025-05-14T14:43:48.605+05:30  INFO 12346 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializatio
```

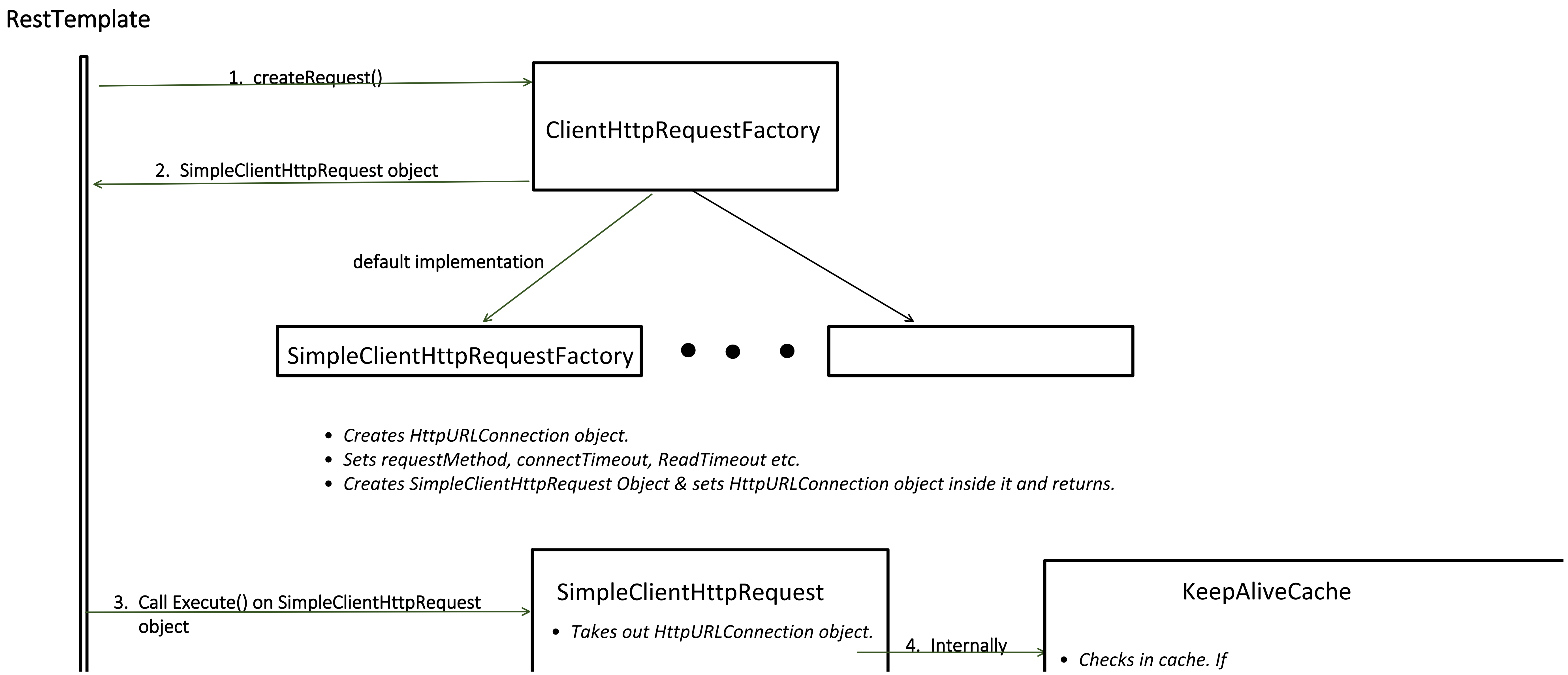


```
2025-05-14T14:43:48.729+05:30  INFO 12346 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8081 (http) with c
2025-05-14T14:43:48.734+05:30  INFO 12346 --- [           main] c.c.o.OrderserviceApplication            : Started OrderserviceApplication in 0.62 s
2025-05-14T14:43:51.223+05:30  INFO 12346 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'di
2025-05-14T14:43:51.223+05:30  INFO 12346 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-05-14T14:43:51.224+05:30  INFO 12346 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
Response from Product APi called from order service: Product fetched with id: 1
```

So, what exactly happened, RestTemplate works internally:

When below method *"getForObject"* invoked:

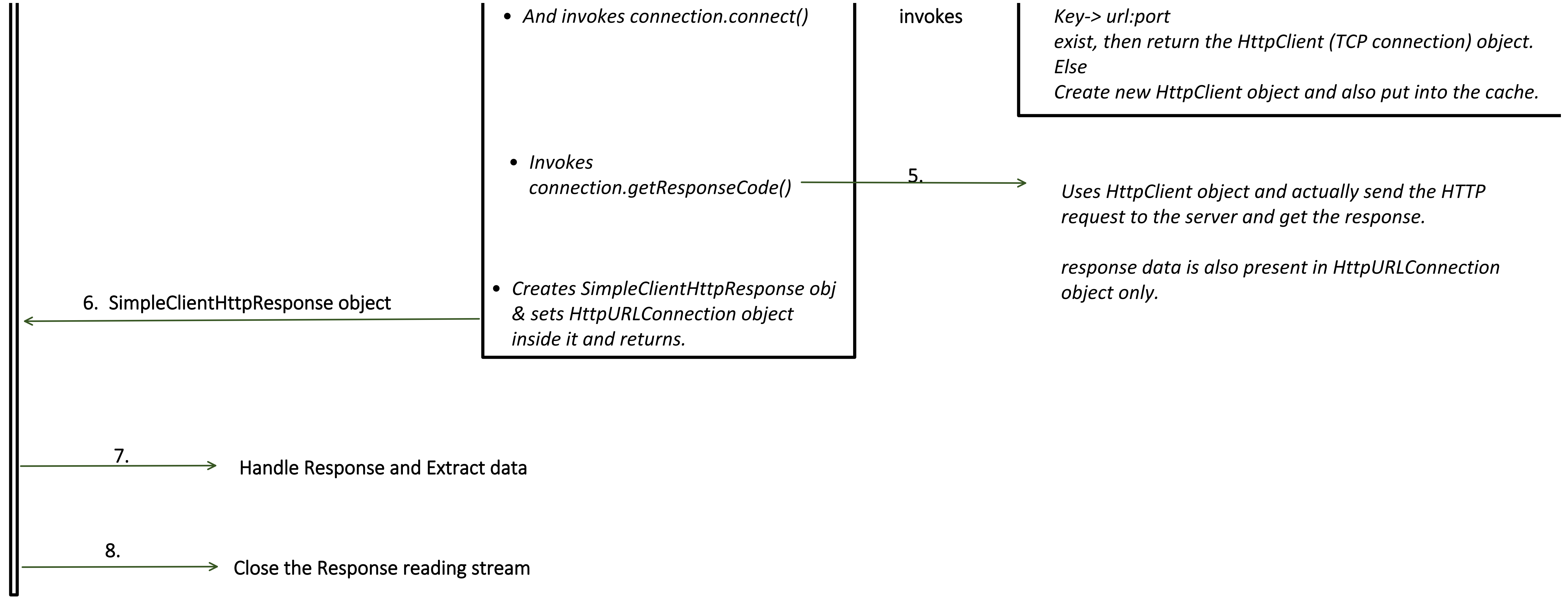
String response = restTemplate.getForObject("http://localhost:8082/products/"+id, String.class);



- *Creates HttpURLConnection object.*
- *Sets requestMethod, connectTimeout, ReadTimeout etc.*
- *Creates SimpleClientHttpRequest Object & sets HttpURLConnection object inside it and returns.*

- *Takes out HttpURLConnection object.*

- *Checks in cache. If*



- *And invokes connection.connect()*

invokes

Key-> url:port
exist, then return the HttpClient (TCP connection) object.
Else
Create new HttpClient object and also put into the cache.

- *Invokes connection.getResponseCode()*

5.

Uses HttpClient object and actually send the HTTP request to the server and get the response.

response data is also present in HttpURLConnection object only.

6. SimpleClientHttpResponse object

- *Creates SimpleClientHttpResponse obj & sets HttpURLConnection object inside it and returns.*

7. Handle Response and Extract data

8. Close the Response reading stream

Notice one thing that:

- RestTemplate do not close the TCP connection explicitly.
- Once Response stream is closed, TCP connection (i.e. HttpClient object, is ready to be re-used till is not expired based on idle Timeout or max Connection configuration)

Lets see, some of the other methods which are available in RestTemplate

Method Name	Description
GET	

getForObject(String url, Class<T> responseType)	Returns the response body as an Object.
	
	String url = "http://localhost:8080/api/products/1";
	Product product = restTemplate.getForObject(url, Product.class);

getForEntity(String url, Class<T> responseType)	Returns full ResponseEntity with status and header
	
	String url = "http://localhost:8080/api/products/1";
	ResponseEntity<Product> response = restTemplate.getForEntity(url, Product.class);
	HttpStatus status = response.getStatusCode();
	Product product = response.getBody();

POST	

postForObject(String url, Object request, Class<T> responseType)	Sends POST and get just the response body.
	
	String url = "http://localhost:8080/api/products";
	Product newProduct = new Product("Ice-cream", 100);
	Product createdProduct = restTemplate.postForObject(url, newProduct, Product.class);

postForEntity(String url, Object request, Class<T> responseType)	Sends POST and get just the full ResponseEntity object.
	
	String url = "http://localhost:8080/api/products";
	Product newProduct = new Product("Ice-cream", 100);
	ResponseEntity<Product> response = restTemplate.postForEntity(url, newProduct, Product.class);
	Product createdProduct = response.getBody();
	HttpStatus status = response.getStatusCode();

PUT	

put(String url, Object request)	Sends PUT and no response body is expected.
	
	String url = "http://localhost:8080/api/products/1";
	Product updatedProduct = new Product("Ice-cream", 150);

	restTemplate.put(url, updatedProduct);
DELETE	

delete(String url)	Sends DELETE request and no response body is expected.
	
	String url = "http://localhost:8080/api/products/1";
	restTemplate.delete(url);

GENERAL PURPOSE	

exchange(String url, HttpMethod method, HttpEntity<?> requestEntity, Class<T> responseType)	When we want to customize :
	• HTTP method (GET, PUT, POST etc.)
	• HTTP header and body (HttpEntity)
	• But want Spring automatic Conversion
	
	String url = "http://localhost:8080/api/products/";
	
	//customizing header
	HttpHeaders headers = new HttpHeaders();
	headers.setContentType(MediaType.APPLICATION_JSON)headers.set("Authorization", "Bearer my-token");
	
	//preparing http request body
	Product product = new Product();
	product.setName("Ice-cream");
	product.setPrice(100);
	
	//setting both header and body in the HttpEntity
	HttpEntity<Product > requestEntity = new HttpEntity<>(product, headers);
	
	ResponseEntity<Product> response = restTemplate.exchange(
	  url,
	  HttpMethod.POST,
	  requestEntity,
	  Product.class
	);
	
	Product product = response.getBody();
	HttpStatus status = response.getStatusCode();

execute(String url, HttpMethod method, RequestCallback requestCallback, ResponseExtractor<T> responseExtractor)	When we want full control like in plain java we use HttpURLConnection object. Header, body, Request, Response, serialization etc. need to be handled manually.
	
	RequestCallback interface, gives us full control over the request. We can set header, write body etc.
	
	@FunctionalInterface
	public interface RequestCallback {
	  void doWithRequest(ClientHttpRequest request) throws IOException;
	}
	
	Similarly, ResponseExtractor, gives us full control over, how the response is read and converted to desired object.

	```
	@FunctionalInterface
	public interface ResponseExtractor<T> {
	    T extractData(ClientHttpResponse response) throws IOException;
	}

	RestTemplate restTemplate = new RestTemplate();

	String url = "http://localhost:8080/api/products";

	//setting both header and body
	RequestCallback requestCallback =  request -> {
	    request.getHeaders().setContentType(MediaType.APPLICATION_JSON);
	    Product product = new Product("Ice-cream", 100);
	    ObjectMapper mapper = new ObjectMapper();
	    byte[] body = mapper.writeValueAsBytes(product);
	    StreamUtils.copy(body, request.getBody());
	};

	//parsing the response
	ResponseExtractor<String> responseExtractor = response -> {
	    return StreamUtils.copyToString(response.getBody(), StandardCharsets.UTF_8);
	};

	String response = restTemplate.execute(
	    url,
	    HttpMethod.POST,
	    requestCallback,
	    responseExtractor
	);

	System.out.println("response is: " + response);
	```

Limitation of RestTemplate:

- In RestTemplate, there are already so many overloaded methods, so its hard to remember and maintain.(Above we have just covered few)
- RestTemplate was build before concepts like Retry, circuit breaker etc.. So adding support means more overloaded methods and not user friendly.

- RestTemplate is in Maintenance mode - means no new feature, only bug fixes.

That's where latest RestClient comes into the picture:

- Introduction of Fluent, builder-style API (more readable and user friendly way of configuring and invoking the endpoint)
- RestClient supports easy integration with interceptors, filters etc.