selsoft
build better, deliver faster

Developing RESTful Web Services with Java

Chapter 6: Fundamentals of JAX-RS API

Eğitmen:

Akın Kaldıroğlu

Çevik Yazılım Geliştirme ve Java Uzmanı

Topics

- **JAX-RS API**
- **Path and Its Properties**
  - ApplicationPath & Path
  - Resource
  - Path and Query Parameter
- **Content Negotiation**
  - MIME Types and Produces
- **Request Method Designators**
  - Properties of the Resource Classes and Methods
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
  - PATCH
  - OPTIONS

JAX-RS API

selsoft

build better, deliver faster

JAX-RS

- **Java API for RESTful Web Services** or **JAX-RS** in short is Java EE's specification for RESTful web services.
- **JSR-370** (https://jcp.org/en/jsr/detail?id=370) specifies the JAX-RS 2.1 which is part of Java EE 8.
- It is part of Glassfish AS 5.

Eclipse Jersey - I

- **Eclipse Jersey** is the reference implementation of JAX-RS 2.1.
  - https://eclipse-ee4j.github.io/jersey/
- Its latest version is 2.33 as of February 2021.
- Its source is available at https://github.com/eclipse-ee4j/jersey
- Its binary and source jars can be downloaded from https://repo1.maven.org/maven2/org/glassfish/jersey/bundles/jaxrs-ri/ or https://jakarta.oss.sonatype.org/content/groups/public/org/glassfish/jersey/bundles/jaxrs-ri/

Eclipse Jersey - II

- Its user guide is at https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/index.html

Jakarta RESTful Web Services

- As part of JEE 9 or Jakarta EE 9 in new home, its new name is **Jakarta RESTful Web Services**.
  - https://jakarta.ee/specifications/restful-ws/3.0/
  - Its version is 3.0
- It is part of Glassfish AS 6.

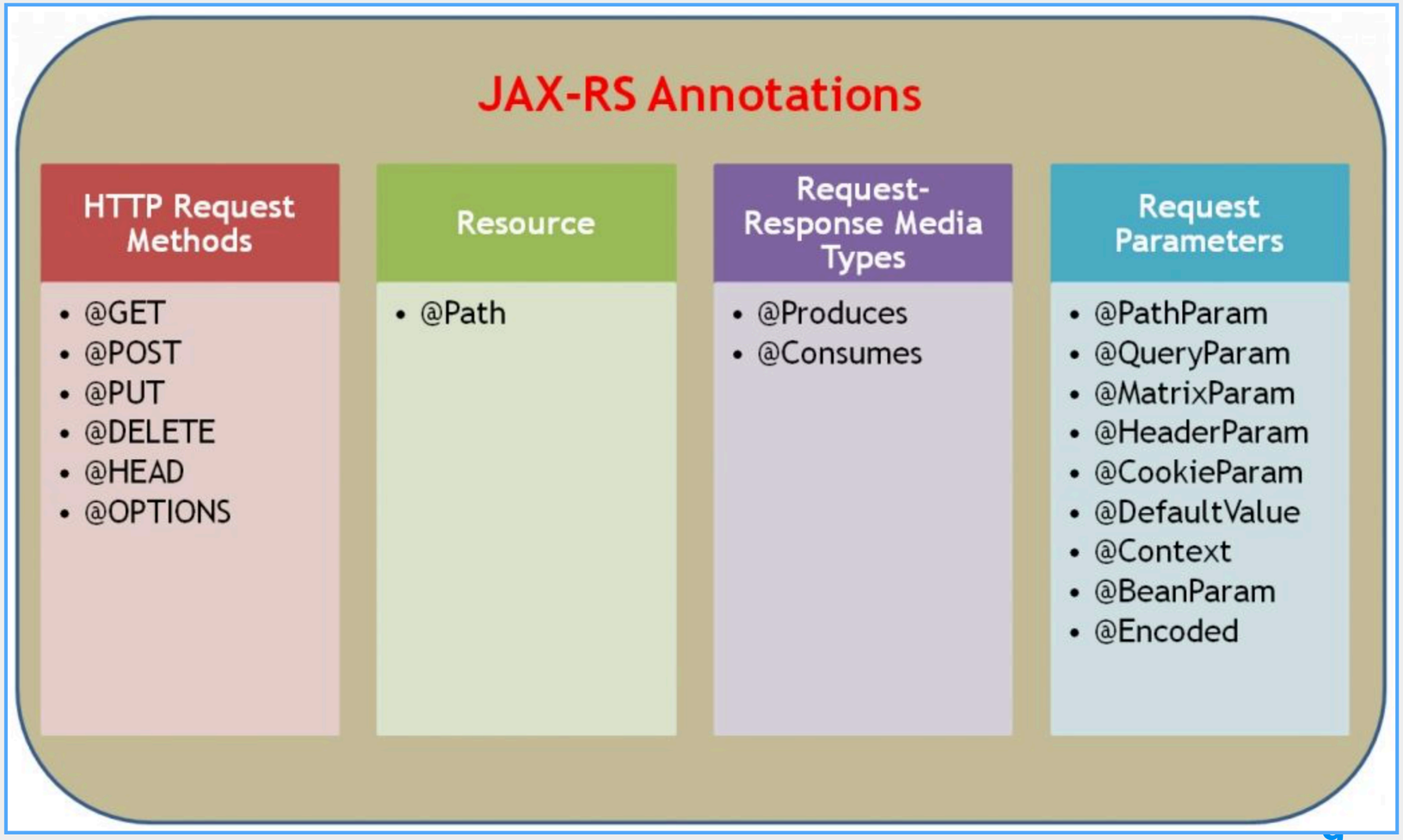
JAX-RS API

- JAX-RS's latest version has following packages:
  - **`javax.ws.rs`** has high-level interfaces and annotations.
  - **`javax.ws.rs.client`** has a client API.
  - **`javax.ws.rs.container`** has a container specific API.
  - **`javax.ws.rs.core`** has low-level interfaces and annotations.
  - **`javax.ws.rs.ext`** is for extensions.
  - **`javax.ws.rs.sse`** is server-sent events related API.

Main JAX-RS Annotations

- From the book **RESTful Java Web Services 3rd Ed**.

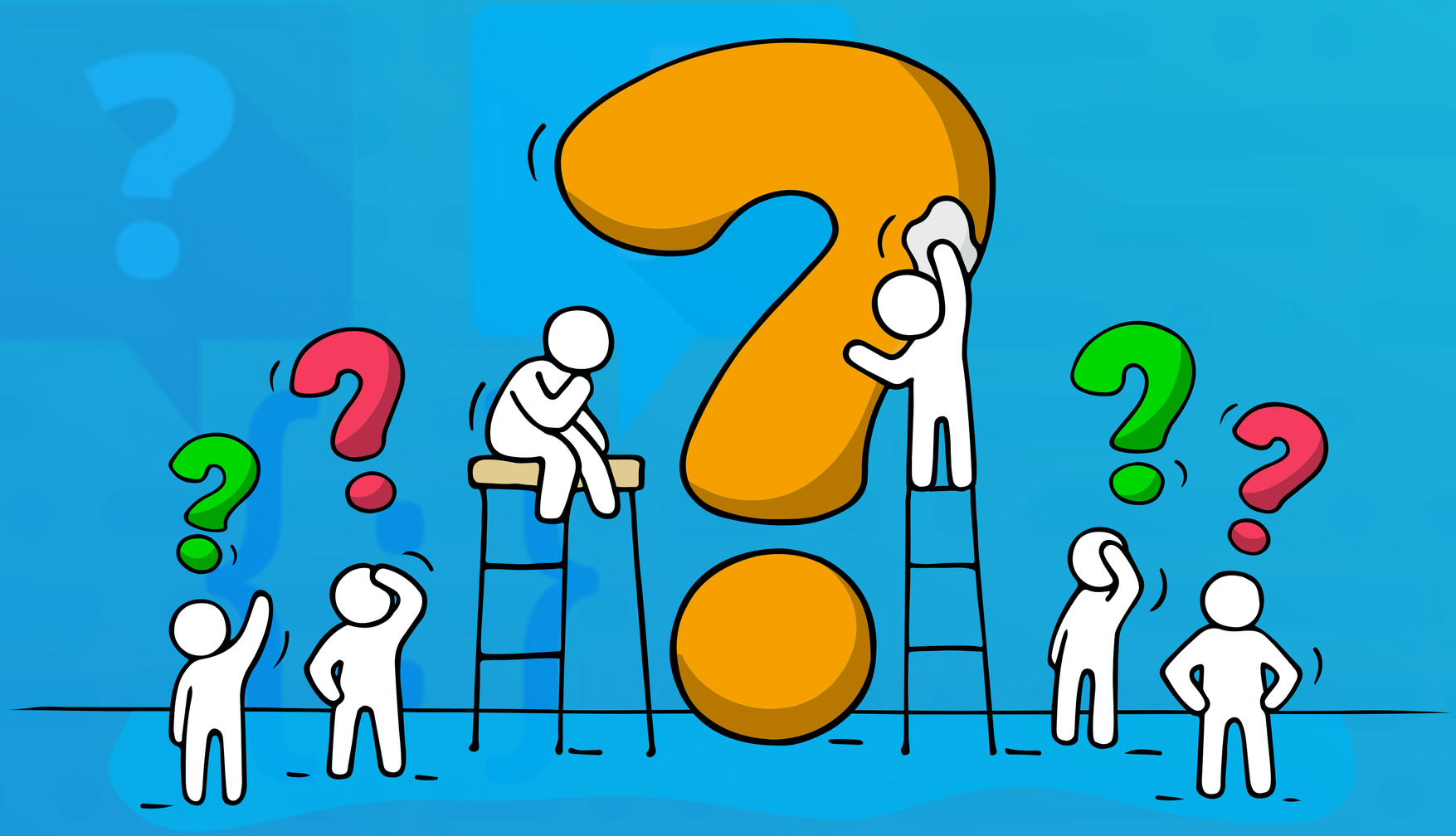
- This chart would be more complete if `@ApplicationPath` existed.



JAX-RS Annotations

- JAX-RS annotations at Java EE 8 tutorial: https://javaee.github.io/tutorial/jaxrs002.html
- There is a cheat sheet for JAX-RS annotations at http://www.masterthebOss.com/jboss-frameworks/resteasy/jax-rs-cheatsheet

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Path and Its Properties

selsoft
build better, deliver faster

Path and Document Root

- The top-level directory of a web application is called **document root** or just **root** of the application.
- A **context root** identifies a web application in a server and is specified when the web application is deployed.
- Context root points to document root of the application on the server.
- A context root must start with a forward slash `/` and end with a string.

```
/WAP4.0
/GreetingRest
/manager
/examples
```

Base URL

- **Base URL** of a web application specifies its location and points to the context root of the application.
- All resources can only be reached on the base URL.

```
http://localhost:8080/WAP4.0
http://localhost:8080/WAP4.0/SelamServlet

http://localhost:8080/GreetingRest
http://localhost:8080/GreetingRest/greetings
http://localhost:8080/GreetingRest/turkish

http://localhost:8080/manager
http://localhost:8080/examples
```

ApplicationPath & Path

ApplicationPath - I

- **`javax.ws.rs.ApplicationPath`** is an annotation that is used to determine the base path for REST resources.
  - Base path is part of the base URL of the web application.
- It has an attribute, **`value`** of type **`String`** which identifies the base URI.
- **`ApplicationPath`** can only be applied to a subclass of **`javax.ws.rs.core.Application`**.
- A typical REST application provides a class that extends **`Application`** and has **`ApplicationPath`** annotation with a path such as **`resources`**.

ApplicationPath - II

- A trailing `/` character will be automatically appended if one is not present.
- So if the base URI given by **`ApplicationPath`** is **`resources`** and the **GreetingRest** application is deployed so that its root URL is http://www.javaturk.org/GreetingRest/ then all REST endpoints would be under http://www.javaturk.org/GreetingRest/resources/.
- We'll see more on **`Application`** class.

```
@ApplicationPath("resources")
public class AppConfig extends Application {
    …
}
```

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.AppConfig`
- Run project and view `index.html`

@PATH - I

- Main annotation of **`javax.ws.rs`** is **`Path`**.
- **`Path`** can be used for resource classes and resource methods.
- It has an attribute, **`value`** of type **`String`** which identifies the URI path that a resource class or method will serve.
- All paths are relative to the application path, which is specified by **`ApplicationPath`**.

```
@Path("greetings")
public class GreetingResource {
    …
}
```

@Path - II

- For the purposes of absolutizing a path against the base URI, a leading `/` in a path is ignored and base URIs are treated as if they ended in `/`.
- So if the base URI given by **`ApplicationPath`** is **`resources`** and **`@Path("greetings")`** is used before **`GreetingResource`** class then all URIs in this class would start with **`/resources/greetings`**.
- Do not put `/` in paths, leading or trailing, using it causes confusion, JAX-RS handles all `/` needs.
  - So **`greetings`** , **`greetings/`**, and **`/greetings/`** are the same.

@Path - III

- Ands all resources will be reachable on the URI **`GreetingRest/resources/`**greetings.

```
@Path("greetings")
public class GreetingResource {
    …
}
```

- So for **GreetingRest**, all resources in class **`GreetingResource`** would be under the URL http://www.javaturk.org/GreetingRest/resources/greetings.

@Path - IV

- If a resource method is annotated by **`@Path`** then its path would be based on the path of its class.
  - In this case the method is called **sub-resource** where the main resource is the class which has its own **`@Path`**.
- So the request URI template that would match the URI of the method would be a URI template created by concatenating the URI of the resource class with the URI of the method.

@Path - V

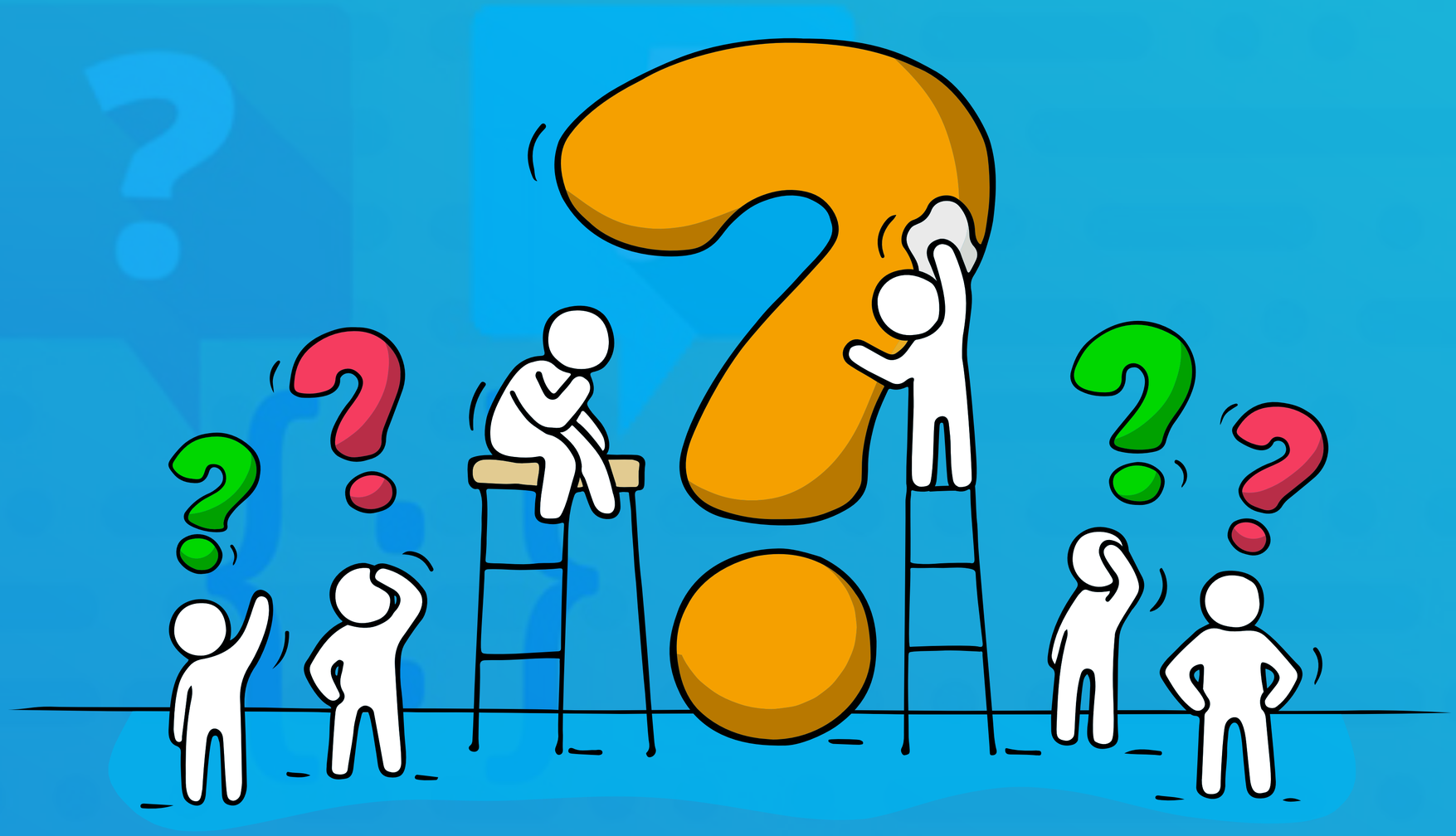
- So the URI of **count** resource would be **`greetings/count`**.
- So URL for **count** resource in class **`GreetingResource`** would be http://www.javaturk.org/GreetingRest/resources/greetings/count.

```
@Path("greetings")
public class GreetingResource{
    …
    @GET
    @Path("count")
    public int getGreetingCount(){…}
    …
}
```

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.DefaultGreetingResource` and `GreetingResource`
- Have a look at `@Path` annotations on classes and their resource methods.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Resource

Resource - I

- REST is about serving resources.
- A REST resource is an object which probably has associations with other resources.
- REST resources are mostly POJO classes whose objects mostly have unique ids and designated with JAX-RS annotations.
- In Java EE world, stateless or singleton EJBs can be defined as REST resources as well.

Resource - II

- Resources mostly have plural nouns such as **`greetings`** or **`users`**.
- Actions on resources are represented by HTTP methods which are verbs.
- Remember that resources are identified by URIs uniquely.
- So URIs are composed of mainly plural nouns.

```
@Path("greetings")
public class GreetingResource {
   …
   @GET
   @Path("objects")
   public List<Greeting> getAllGreetingObjects() { … }
}
```

```
@Path("departments")
@Stateless
public class DepartmentService { … }
```

Resource - III

- So class names can be different, important thing is the name of the resource which is identified by URIs.

```
@Path("greetings")
public class GreetingResource {
    …
}
```

```
@Path("departments")
@Stateless
public class DepartmentService {
    …
}
```

```
GET /greetings
GET /departments
```

Application Programming Interface

- Resources can be **c**reated, **r**ead, **u**pdated and **d**eleted.
- So a RESTful service consists of **CRUD** actions on resources.
- So all CRUD operations regarding resources in a web service are called **Application Programming Interface** (**API**).
- API of a web service is what it presents to its clients regarding its resources.

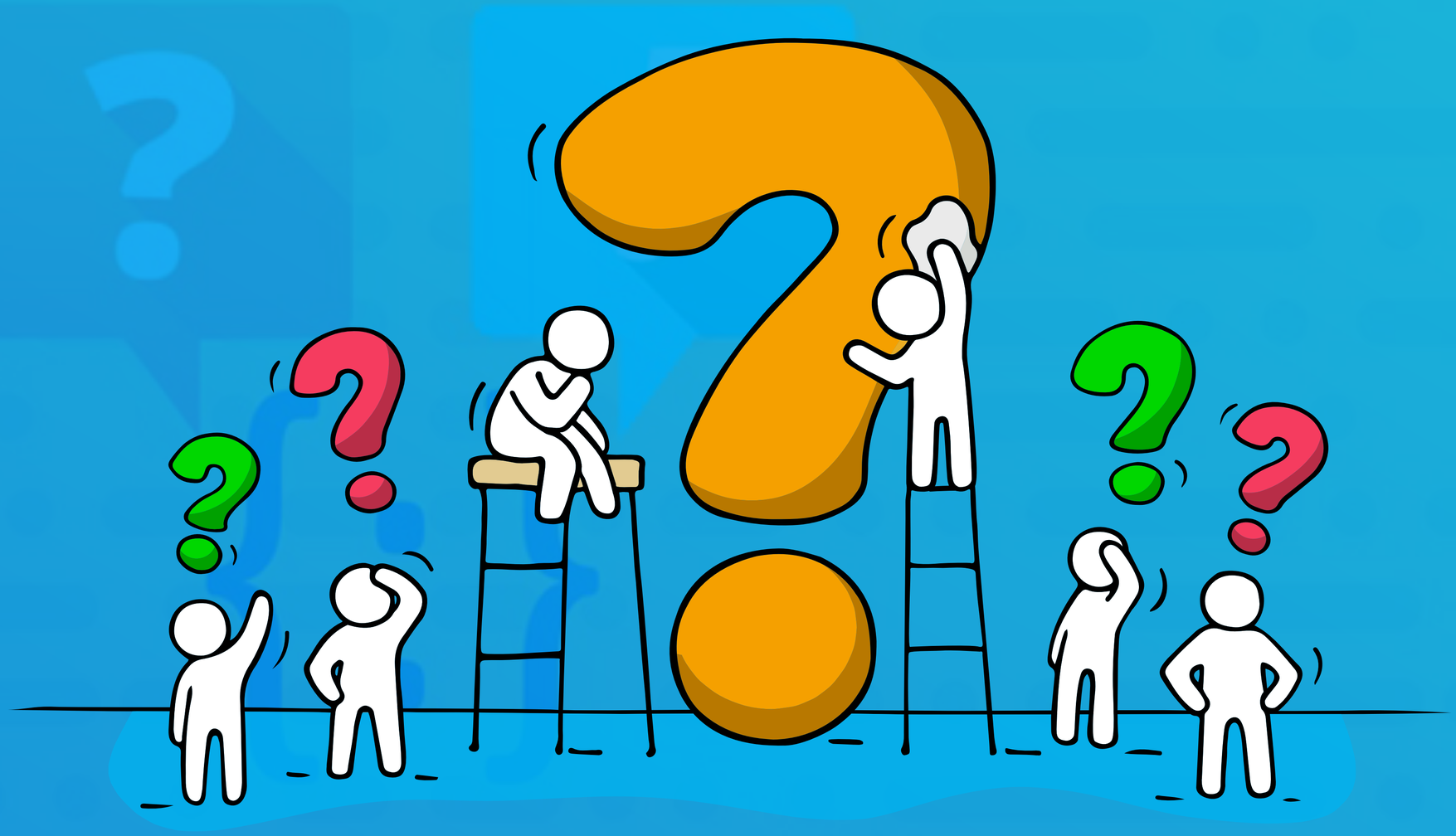
API Design

- In a highly digitized world API becomes a product that a business offers.
- So designing a good and usable API is very important.
- We’ll talk about API design later in more detail.

Resources and HTTP Methods

- In REST HTTP methods are used for CRUD operations of the resources:
  - **GET**: Retrieve a resource or a set of resources
  - **POST**: Create one or more new resources
  - **PUT**: Replace one or more resources
  - **DELETE**: Delete one or more resources
  - **PATCH**: Update only a part of a resource

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Path Parameter

Path Parameters - I

- Resources mostly have plural nouns such as `greetings` or `users`.
- A specific resource probably with an id is requested from a resource URI.
  - Remember resources are always identified with URIs.
- So to choose among a set of resource instances a parameter for id should be passed to the server.
- URI path template allows to define placeholders called **path parameter** (or **template parameter** and **path variable**) for these kinds of needs.

Path Parameters - II

- Path parameter is part of the URI and is specified in curly brackets (or braces) `{}`.
- A path can have many path parameters separated by a `/`.
- Part of the path that declares the parameters is called **path segment**.

```
@Path("greetings")
public class GreetingResource {
    …
    @GET
    @Path("{language}")
    public String getGreeting(…) {…}
}
```

```
GET  /greetings/turkish
```

```
@Path("greetings")
public class GreetingResource {
    …
    @POST
    @Path("{language}/{greeting}")
    public Response createGreeting(…)
}
```

```
POST /greetings/greek/chairete
```

@PathParam - I

- **`javax.ws.rs.PathParam`** is used to bind the value of a URI template parameter or a path segment containing the template parameter to a resource method parameter (and resource class field, or resource class bean property).
- It has only one attribute, **`value`** of type **`String`** which is the name of the parameter.

```
@Path("greetings")
public class GreetingResource {
    …
    @GET
    @Path("{language}")
    public String getGreeting(@PathParam("language") String language) {…}
}
```

```
https://localhost:8080/GreetingRest/resources/greetings/turkish
```

@PathParam - II

- At runtime when the resource is asked the system binds the value passed in the path to the parameter of the resource method.
- If the path has many path variables separated by `/` then many **`@PathParam`** are used.
  - Names of the path and method parameters need not be the same.

```
@Path("greetings")
public class GreetingResource {
    …
  @POST
  @Path("{language}/{greeting}")
  public Response createGreeting(@PathParam("language") String language, @PathParam("greeting") String greeting) {…}
}
```

```
https://localhost:8080/GreetingRest/resources/greetings/turkish/selam
```

@PathParam - III

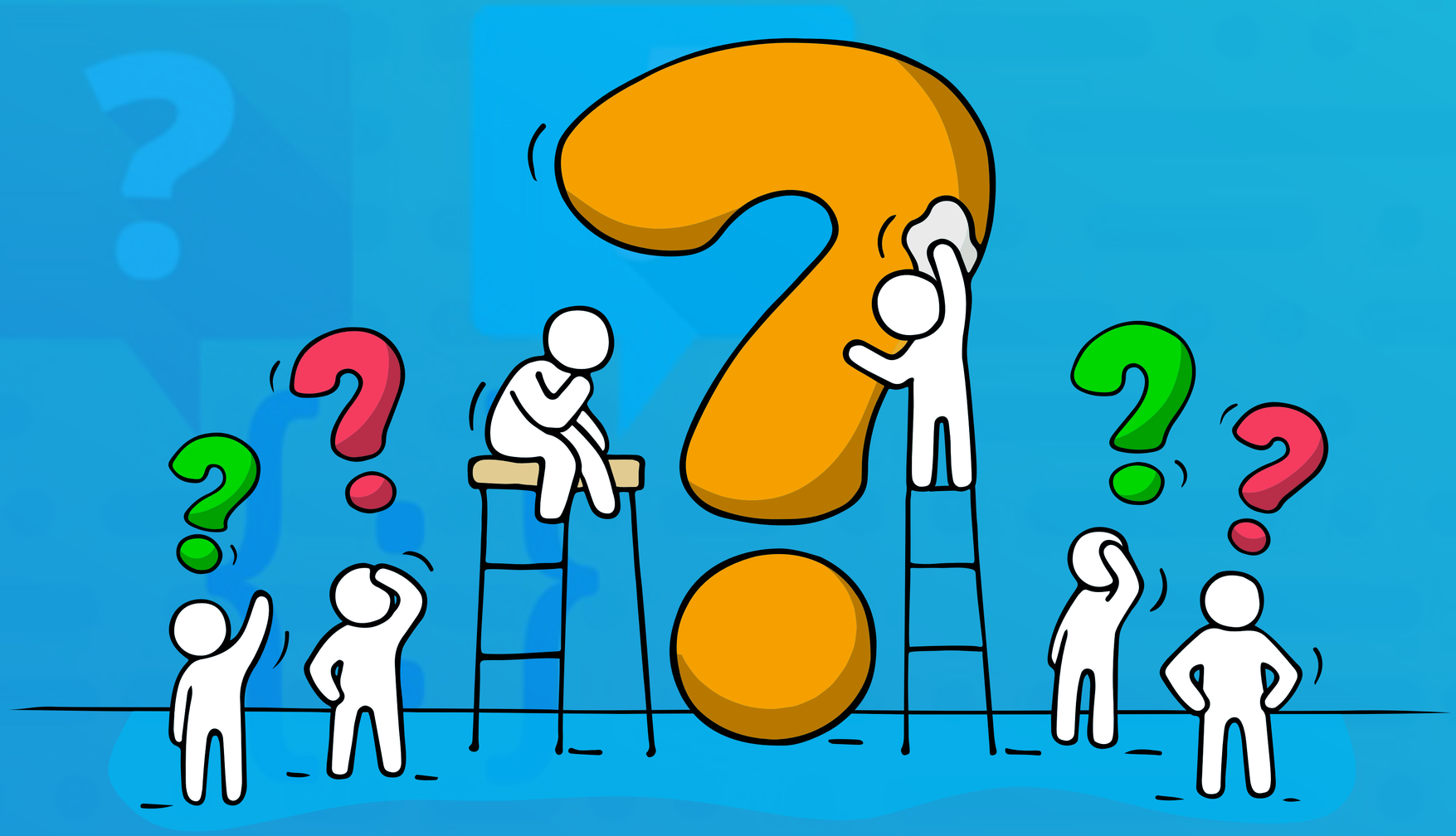
- A regular expression can also be used to restrict the values for path parameters.
- If the path parameter is specified in **`@Path`** of class the all of its resource methods can reach that parameter using **`@PathParam`**.

```
@Path(“greetings/{language}")
public class GreetingResource {
   …
  @POST
  @Path("{greeting}")
  public Response createGreeting(@PathParam("language") String language, @PathParam("greeting") String greeting) {…}
}
```

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.DefaultGreetingResource`
- Have a look at `@Path` and `@PathParam` annotations on resource methods.
- Call paths with path parameters and provide values.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Query Parameter

Query String or Component

- Another way to specify something related to the resources on the server is **query string** (or **query component**).
- Query string is part of URL and was originally intended to clarify the URL.
- But later it became a way to send data to the server.
- A query string helps to narrow down the resource that is searched for.
- URI path template allows to define placeholder for query string.

```
http: // hostname [:port] / path [/parameters] [?query]
https: // hostname [:port] / path [/parameters] [?query]
```

Query Parameter

- A query string starts with a `?` and includes one or more **query parameter**s.
- Query parameters are in the format of `name=value` and separated by `&`.

```
https://google.com/search?q=best+rest+books&oq=best+rest+books&aqs=chrome...
```

```
http://www.javaturk.org/?s=assert
```

@QueryParam - I

- **`javax.ws.rs.QueryParam`** is used to bind the value of a URI query parameter to a resource method parameter (and resource class field, or resource class bean property).
- It has only one attribute, **`value`** of type **`String`** which is the name of the parameter.

```
@Path("greetings")
public class GreetingResource {
    …
    @GET
    public String getGreeting(@QueryParam("language") String language) {…}
}
```

```
https://localhost:8080/GreetingRest/resources/greetings?language=turkish
```

@QueryParam - II

- At runtime when the resource is asked the system binds the value passed in the path to the parameter of the resource method.
- If the path has many query parameters separated by a `&` then many `@QueryParam` annotations are used.
- And of course the names of the query parameter and method parameter need not the be same.

```
@Path(“defaultGreetings")
public class GreetingResource {
    …
  @POST
  public Response createGreeting(@QueryParam("language") String language, @QueryParam("greeting") String greeting){…}
}
```

@QueryParam - III

- Notice that even though HTTP POST is used the query parameters are sent to the server in the URL.
- That makes all query parameters and their values visible.
- So query parameters should not carry sensitive information.
- They should be used mostly for non-sensitive purposes such as searching, filtering and pagination.

@PathParam vs @QueryParam

- Query parameters are not part of the URI i.e. they are not specified in URI template and become visible only when the request is submitted.
- So main difference between a path parameter and a query parameter is that query parameters can’t be specified in `@Path`.

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.DefaultGreetingResource`
- Have a look at `@QueryParam` annotations in resource methods.
- Call paths with query parameters and provide values.

Using Parameters and Limits

Parameters - I

- Path and query parameters are all to send data to the server.
- Although they all can be used to send data to the server, they have different usages:
  - Path parameters are mainly used to identify collections of resources or a single resource and their hierarchies,
  - Query parameters are mainly used to query collections and provide data for filtering, sorting, pagination, etc.

Parameters - II

- There is one other kind of parameter called **matrix parameter**.
  - It will be studied later.
- So, path, query and matrix parameters are all to send data to the server.

Limit for URI Length - I

- Using parameters, path, query or other kinds, increases the length of the URI.
- And there might be a limit enforced by browsers, servers or even intermediaries.
- HTTP specification doesn’t impose any limit on URI length but it warns:
  - *Servers ought to be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations might not properly support these lengths.*

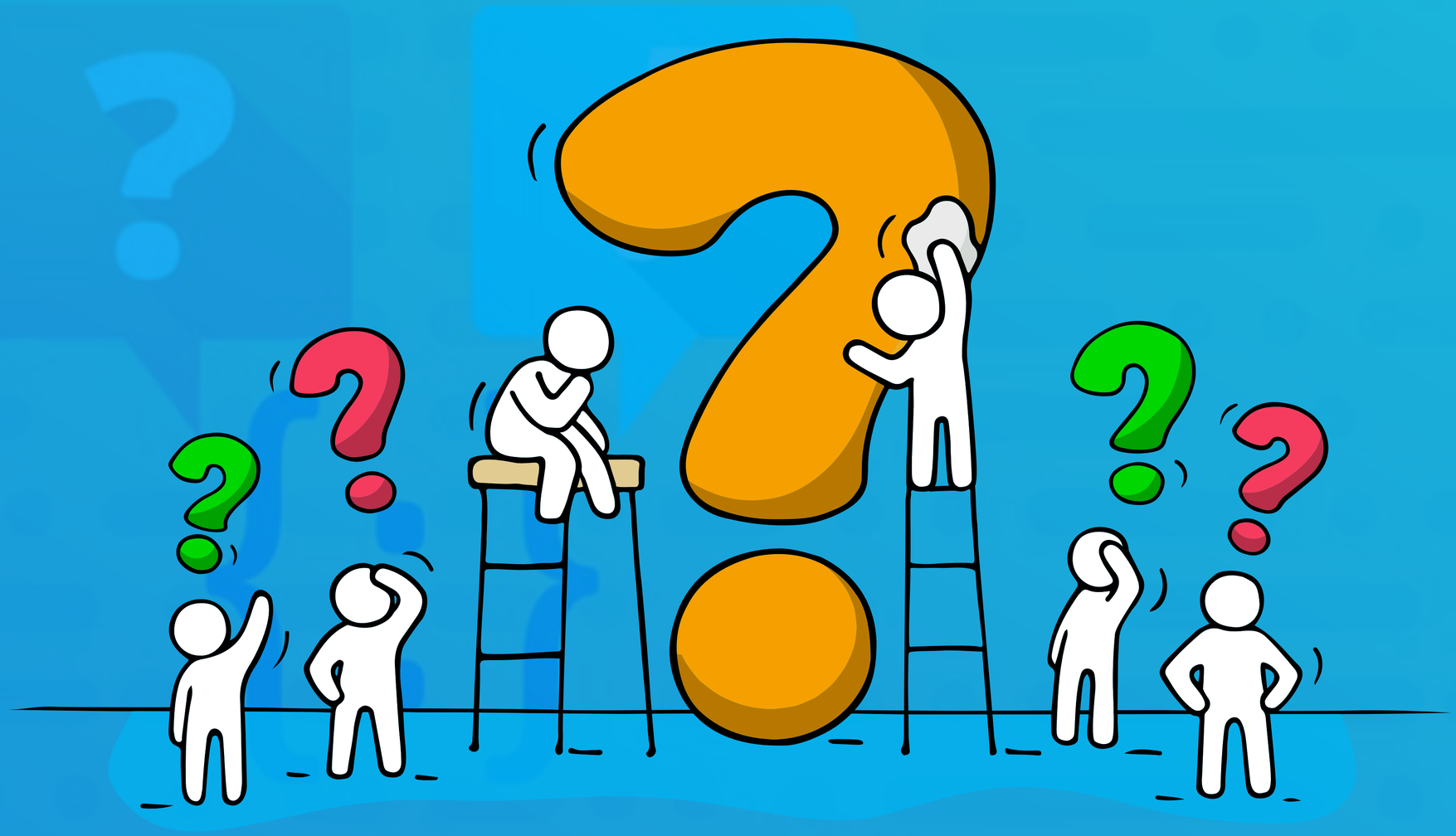
Limit for URI Length - II

- There is HTTP **414 URI Too Long** status code that indicates that the URI requested is longer than the server is willing to interpret so the server refuses to service.

Tomcat's Limits

- For Tomcat there are attributes called **`maxHttpHeaderSize`** and **`maxParameterCount`** for connectors.
  - **`maxParameterCount`** determines the whole max size of headers and request line for request and max size of headers and status line for response and its default value is 8192 bytes (8KB).
  - **`maxParameterCount`** which determines maximum number of parameter and value pairs (GET plus POST).
    - Its default value is 10000.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Content Negotiation

selsoft
build better, deliver faster

Content Negotiation - I

- REST clients and servers need to know the format of the representation of the resources they receive.
- This is called **content negotation** and it happens between servers and clients.
- HTTP has content negotiation capability mainly by **`Accept`** and **`Content-Type`** headers.
- **`Accept`** and **`Content-Type`** headers are expressed in terms of MIME types.

Content Negotiation - II

- **`Accept`** is a request header and specifies what types of media a client prefers for the resource it would receive.
  - The server tries to satisfy this by sending the resource in one of expected media types listed in **`Accept`** header of the request and specifies the format of resource in **`Content-Type`** header.
- **`Content-Type`** is both request and response header and specifies the media of the message.
- **`Accept`** can have one or more values but **`Content-Type`** can only have one.

```
GET /WAP4.0/SelamServlet HTTP/1.1
Host: localhost:7070
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:85.0) Gecko/20100101 Firefox/85.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

```
HTTP/1.0 200 OK
Server: EchoServer
Content-Type: text/html
…
```

```
GET /GreetingRest/resources/greetings/objects HTTP/1.1
User-Agent: PostmanRuntime/7.26.8
Accept: */*
Cache-Control: no-cache
Postman-Token: 14e9b5af-2f70-4039-81d2-194027bc03d0
Host: localhost:4040
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
HTTP/1.0 200 OK
Server: GlassFish Server Open Source Edition  5.1.0
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open
Source Edition  5.1.0  Java/Oracle Corporation/1.8)
Content-Type: application/json
…
```

```
GET /GreetingRest/resources/greetings/objects HTTP/1.1
User-Agent: PostmanRuntime/7.26.8
Accept: application/xml
Cache-Control: no-cache
Postman-Token: 14e9b5af-2f70-4039-81d2-194027bc03d0
Host: localhost:7070
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
HTTP/1.1 406 Not Acceptable
Server: GlassFish Server Open Source Edition  5.1.0
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open
Source Edition  5.1.0  Java/Oracle Corporation/1.8)
Content-Language:
Content-Type: text/html
Connection: close
Content-Length: 1214
```

q-factor Weighting

- In some headers including **`Accept`** there can be a quality value just following the MIME type in the format of **`;q=value`** where value is between 0 and 1 included.
- Quality values, or **q-values** and **q-factors**, are used to describe the order of priority of values in a comma-separated list.
- The highest value denotes the highest priority and when not present, the default value is 1.

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

selsoft

build better, deliver faster

MIME Types

MIME Types - I

- **Multipurpose Internet Mail Extensions** or **MIME** type means media type or format of media.
- It is a standard that indicates the nature and format of a document, file, or assortment of bytes.
- It is defined and standardized in IETF's RFC 6838.
- **The Internet Assigned Numbers Authority** (IANA) manages MIME types:
  - https://www.iana.org/assignments/media-types/media-types.xhtml

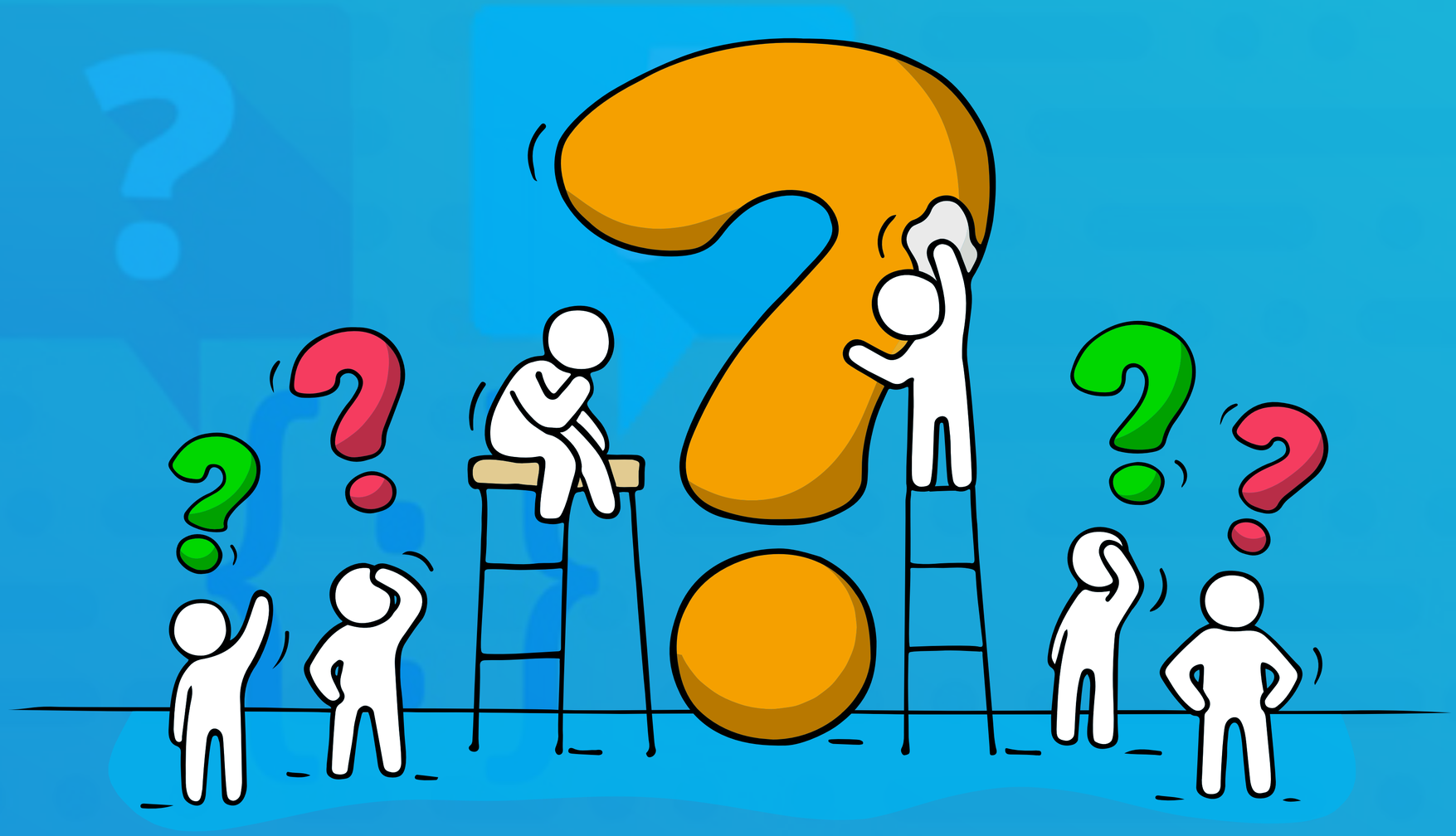
MIME Types - II

- **MIME** types are in the form of **type/sub-type** such as **`application/json`**.
- Type is a main category and can be one of two types: discrete or multipart.
  - Discrete types include **`application`**, **`image`**, **`text`**, **`video`**.
  - Multipart types are types that consist of more than one discrete types such as messages.

MIME Types - III

- Common MIME types are:
  - `text/plain`, `text/html`
  - `application/json` is main MIME type for REST but sometimes `application/xml` is also used.
  - `application/javascript` `application/octet-stream` `application/pdf`
  - `image/gif` `image/jpeg` `image/png`

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Content Negotiation - III

- To meet the content negotiation capability of HTTP, JAX-RS has **`Produces`** and **`Consumes`** annotations in **`javax.ws.rs`** package.
- They let the server specify which media types it can produce and consume.
- While **`Produces`** is used for producing a response, **`Consumes`** is used for consuming a request.
- They are used to create a specific representation of a resource in a specific format.

Content Negotiation - IV

- They can be both used on resource classes and methods.
- When they are used on classes it is also valid for all resource methods.
- And of course resource methods can override it by specifying their own producers and consumers.

```
@Path(“greetings")
@Produces(“application/json”)
public class GreetingResource {
    …
    @GET
    @Path(“{language}")
    @Produces({“application/json”, “application/xml”})
    public String getGreeting(…) {…}
}
```

```
@Path("greetings")
public class GreetingResource {
    …
    @POST
    @Path(“{language}/{greeting}")
    @Consumes(“application/json”)
    public Response createGreeting(…)
}
```

Content Negotiation - V

- If `@Produces` doesn't exist, JAX-RS may render the response in any type.
  - For example in default JAX-RS uses plain text for the representation of primitives and strings which results in `text/plain` in `Content-Type` header of the response.
  - JAX-RS uses JSON for Java objects and collections in default which results in `application/json` in `Content-Type` header of the response.

Content Negotiation - VI

- But plain text is not a good choice for machines so clients and servers should always create more specific representations.
- The same thing is true when server receives a request, it needs to know the format of the representation of the resource it receives.

GreetingRest

- **GreetingRest** project.
- Compare the media type of following URIs on both `DefaultGreetingResource` and `GreetingResource` by calling:
  - `count`
  - `{language}` vs. `{language}/1` and `{language}/2`

Content Negotiation

Produces

@Produces - I

- **`javax.ws.rs.Produces`** is used to produce a representation of a resource in a specific format.
- It has an attribute, **`value`** of type **`String`** that takes one or more **MIME** types separated by comma.
- MIME type of the response is stored in the header **`Content-Type`** of the response so that client can understand the format of the representation of the resource sent by the server.

@Produces - II

- `@Produces` can be used on resource classes and methods.
- If it is used on a class then all resource methods produce response in MIME types specified on the class.
- But methods can override it by specifying their own MIME types in `@Produces`.

@Produces - III

- The same resource may have different representations through different MIME types.
- To achieve this, either different methods with different `@Produces` or one method with `@Produces` listing multiple MIME types can be created.
- In second solution the same resource can be served using the same URI and HTTP method but with different `@Produces` in different methods.
  - That means different resource methods can only differ in terms of MIME types in their `@Produces` in the same class.

@Produces - III

- If `@Produces` has multiple MIME types but the client specifies no `Accept` header then the first MIME type in `@Produces` is applied.

```
@GET
@Path("objects")
@Produces("application/json")
public List<Greeting> getAllGreetingObjectsAsJSON() {
     return repo.getAllGreetingObjects();
}

@GET
@Path("objects")
@Produces("application/xml")
public List<Greeting> getAllGreetingObjectsAsXML() {
     return repo.getAllGreetingObjects();
}

@GET
@Path("objectsJSONorXML")
@Produces({"application/json", "application/xml"})
public List<Greeting> getAllGreetingObjectsAsXMLorJSON() {
     return repo.getAllGreetingObjects();
}
```

GreetingRest

- **GreetingRest** project.
- Call following paths:
  - `objects` on `DefaultGreetingResource`
  - Two `objects` paths and `objectsJSONorXML` path on `GreetingResource`.
  - Notice `@XMLRootElement` on `Greeting` class.
    - Turn in on and off to see its effect.

@Produces - IV

- There are several cases to determine the format of the representation with regard to **`@Produces`** and **`Accept`** header :
  - If the request specifies MIME types in its **`Accept`** header and if the server has a matching representation in **`@Produces`** if serves it in the first preferred format,
  - If the request doesn’t have any **`Accept`** header then the server serves the response in the first MIME type of its **`@Produces`**,

@Produces - V

- If the request specifies the MIME type in its **`Accept`** header and the server doesn’t have any matching type then it lets the client know about this by sending a response with **`406 Not Acceptable`**.

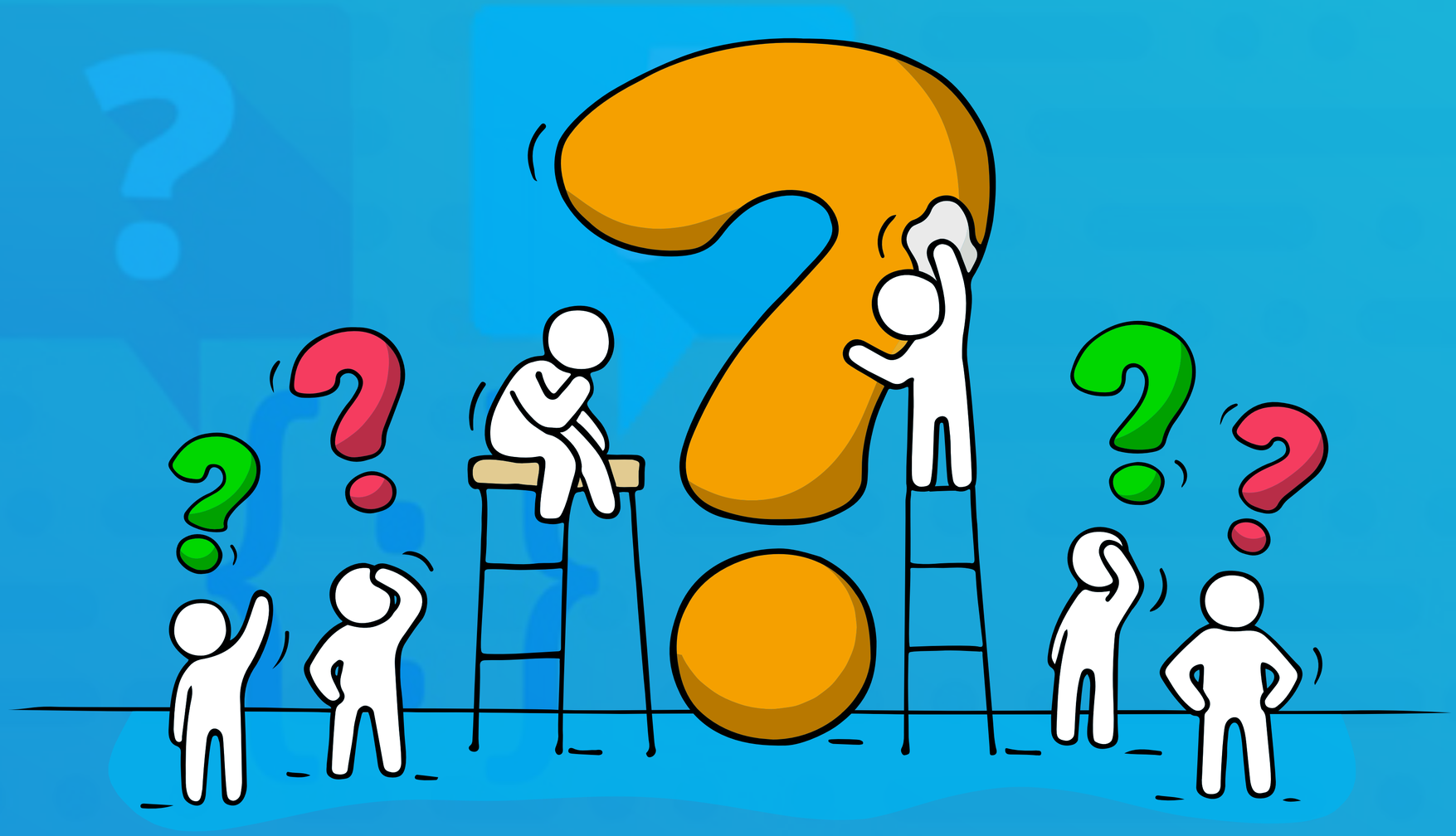
GreetingRest

- **GreetingRest** project.
- Call the path **`objectsJSONorXML`** on **`GreetingResource`** for example with **`application/pdf`** in **`Accept`** header.

JSON vs. XML

- A RESTful web service should support JSON for default representation format of the resources.
- XML can be used optionally as an alternative representation format.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Request Method Designators

selsoft
build better, deliver faster

Request Method Designators

- There are seven annotation to designate resource methods for HTTP requests:
  - `GET`
  - `POST`
  - `PUT`
  - `DELETE`
  - `HEAD`
  - `PATCH`
  - `OPTIONS`

405 Method Not Allowed

- REST developers need to implement resource methods for paths of resources with suitable designator annotations so that clients are served.
- When there is no resource method with corresponding designator for the HTTP request on a path on the server then the server returns **405 Method Not Allowed**.

Properties of Resource Classes

Resource Classes

- Resource classes either
  - must be annotated by **`Path`** or
  - at least one of their methods must be annotated by **`Path`** or one of request method designators such as **`@GET`**, **`@POST`**, etc.
- Resource methods are methods of a resource class annotated with a request method designator.
  - So resource classes have resource methods and non-resource methods as well.

Constructor

- Resource classes must have a **`public`** constructor.
- That constructor may be a non-arg constructor or a constructor that receives parameters in which case JAX-RS would provide all parameter values by injection.
  - Those parameters can be **`@Context`**, **`@Header-Param`**, **`@CookieParam`**, **`@MatrixParam`**, **`@QueryParam`** or **`@PathParam`.**
- There may be more than one constructor and JAX-RS runtime would choose the best fit.

Fields and Properties

- Resource classes can have fields or properties, **`static`** or instance fields and properties.
  - Fields can be declared **`private`** and their getter and setter methods can be provided.
- Fields and properties can be injected by **`@Context`**, **`@Header-Param`**, **`@CookieParam`**, **`@MatrixParam`**, **`@QueryParam`** or **`@PathParam`.**
- Injecting in JAX-RS will be studied in detail.

Life Cycle of Resource Class Instances

- By default a new resource class instance is created for each request to that resource by JAX-RS runtime.
  - First the constructor is called,
  - then any requested dependencies are injected,
  - then the appropriate method is invoked
  - and finally the object is made available for garbage collection.

Properties of Resource Methods

Resource Methods - I

- Resource methods must conform to certain restrictions:
  - A request method designator is a runtime annotation that is annotated with the **`@HttpMethod`** annotation.
    - **`javax.ws.rs.HttpMethod`** annotation has seven fields which corresponds to seven request method designators.
- JAX-RS defines a set of request method designators for the common HTTP methods: **`@GET @POST @PUT @DELETE @PATCH @HEAD @OPTIONS`**.

Resource Methods - II

- Users may define their own custom request method designators including alternate designators for the common HTTP methods.

Visibility

- Resource methods can only be `public`.
- An implementation should warn users if a non-public method carries a method designator or `@Path` annotation.

Properties of the Resource Methods

Parameters

Parameters - I

- When a resource method is invoked, parameters annotated with one of the annotations listed below mapped from the request according to the semantics of the annotation:
- **`@FormParam`**: Extracts the value of a form parameter.
- **`@MatrixParam`**: Extracts the value of a URI matrix parameter.
- **`@QueryParam`**: Extracts the value of a URI query parameter.
- **`@PathParam`**: Extracts the value of a URI template parameter.

Parameters - II

- `@CookieParam`: Extracts the value of a cookie
- `@HeaderParam`: Extracts the value of a header
- `@Context`: Injects an instance of a supported resource.

- And following:
  - **`DefaultValue`** annotation may be used to supply a default value for parameters.
  - **`Encoded`** annotation may be used to disable automatic URI decoding of parameter values.

Parameters - III

- And exceptions thrown during construction of parameter values are treated the same as exceptions thrown during construction of field or bean property values.

EntityParameter

- The parameter not annotated with annotations listed before is called the **entity parameter**.
- Its value is mapped from the request entity body.
- Conversion between an entity body and a Java type is the responsibility of an entity provider.
- Resource methods can have at most one entity parameter.

```
@POST
public Response createGreeting(Greeting greeting, @Context UriInfo uriInfo){
    …
}
```

Properties of the Resource Methods

Return Types

Return Types - I

- JAX-RS specification says that resource methods may return **`void`**, **`javax.ws.rs.core.Response`**, **`javax.ws.rs.core.GenericEntity`** or another Java type.
- These return types are mapped to a response entity body as follows:
  - **`void`**: An empty entity body with a **204 No Content** status code.
  - **`Response`**: An entity body mapped from the entity property of the **`Response`** with the status code specified by the status property of the **`Response`**.

Return Types - II

- **`GenericEntity`**: An entity body mapped from the **`Entity`** property of the **`GenericEntity`**.
  - If the return value is not null a **200 OK** status code is used, a null return value results in a **204 No Content** status code.
- **`Other`**: An entity body mapped from the class of the returned instance or of its type parameter **`T`** if the return type is **`CompletionStage<T>`**; if the class is an anonymous inner class, its superclass is used instead.
  - If the return value is not null a **200 OK** status code is used, a null return value results in a **204 No Content** status code.

Properties of the Resource Methods

Safety, Idempotency, Body and Cache

Safety & Idempotency - I

- There are two important points for HTTP methods:
  - **Safety**: Safe methods don’t cause any change on the state of the server in terms of resources.
  - **Idempotency**: Calling the same method at different times (or many times) should always result in the same effect on the server.
    - Different requests are differet but identical requests.
- Safe methods are supposed to be read-only methods.
- Safe methods are also idempotent but not vice versa.

Safety & Idempotency - II

- **GET** and **HEAD** methods should not have the significance of taking an action other than retrieval so they are both safe and idempotent.
- **POST** is obviously neither safe nor idempotent beause each time it creates a new resource with a new id on the server.
- **PUT** and **DELETE** are not safe but idempotent.
  - They are idempotent just because of the fact that after each time these methods are called either the same resource with the same id exists or doesn’t exist.

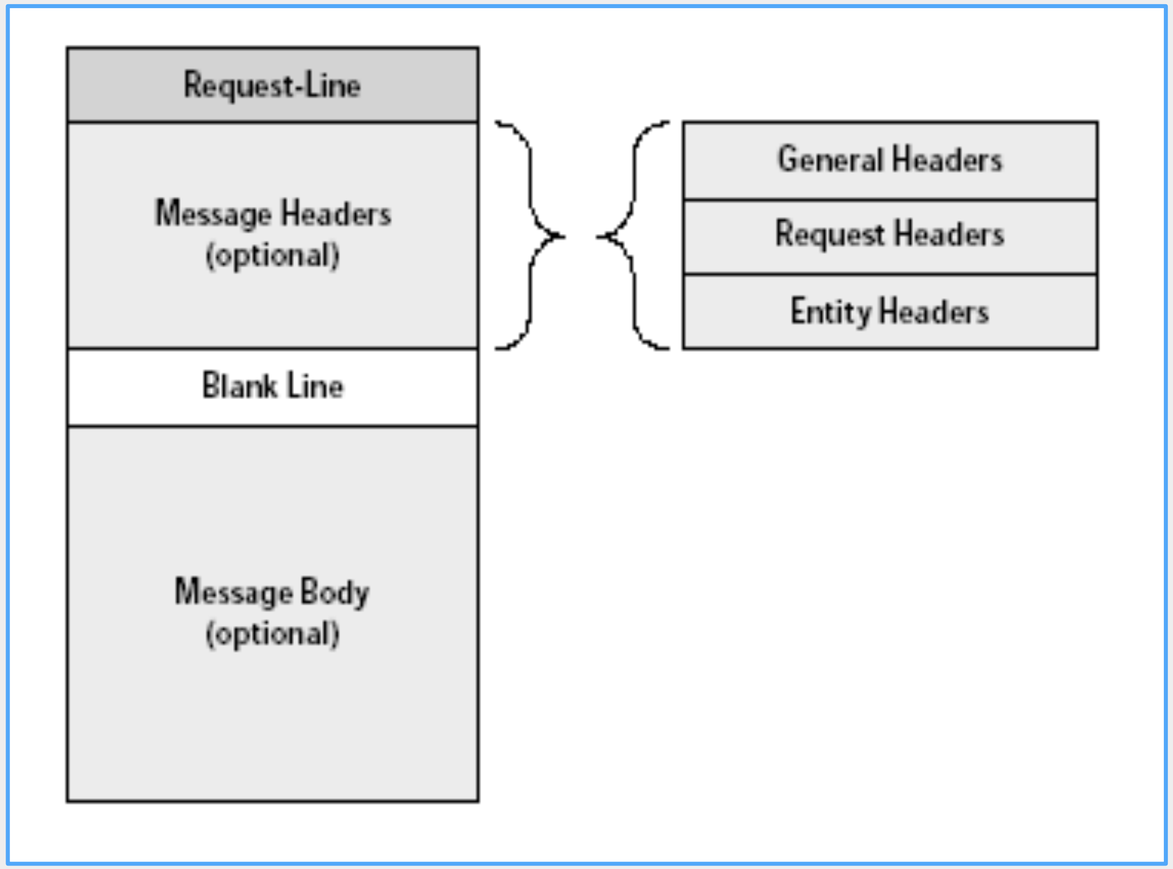
Safety & Idempotency - III

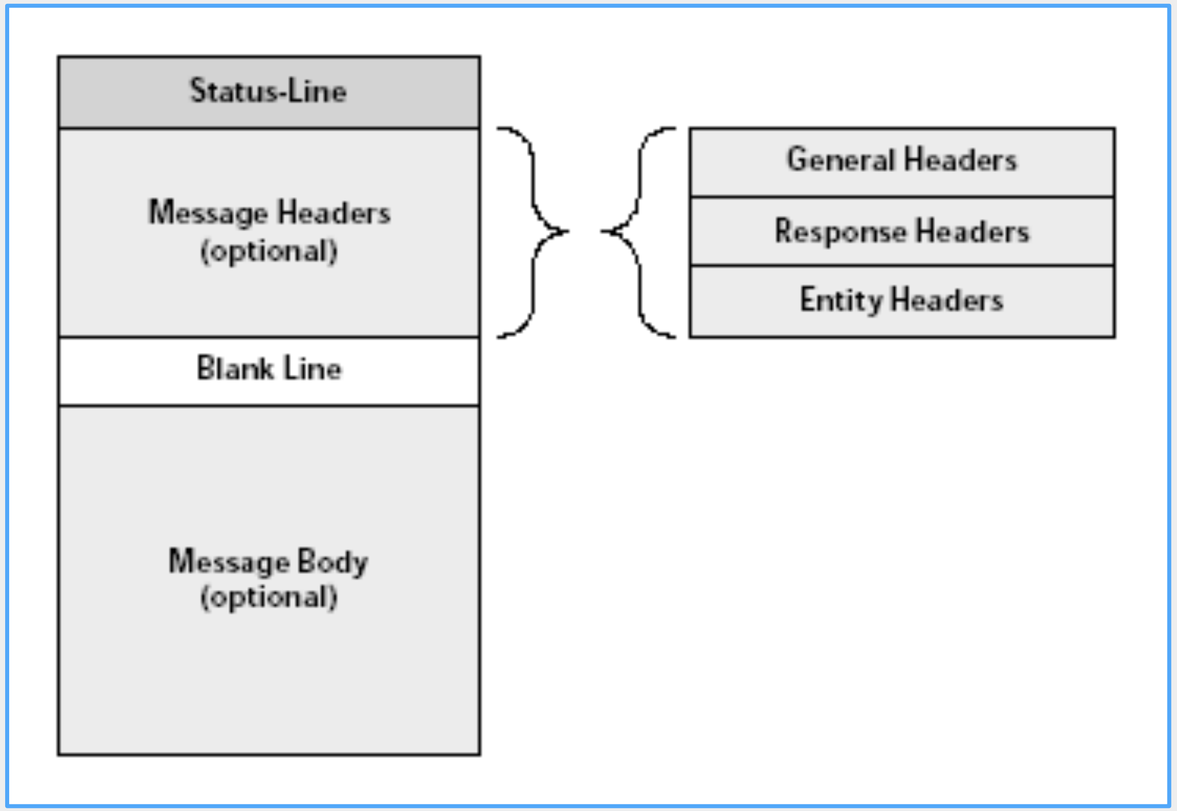
- But **PATCH** is different, it partly updates a resource so it may or may not update the same part of the resource every time it is called.
- So **PATCH** is not idempotent.
- The methods **OPTIONS** and **TRACE** should not have side effects, and so they are inherently idempotent.

Message Body - I

- Message body (or payload body, entity or just body) is part of the HTTP request and response messages.
- It is always after a blank line which comes after the request or status line and headers.
- It is optional and it may contain textual or binary data.
- **Content-Length** header gives the number of bytes in body.
- The presence of a message body in a response depends on both the request method to which it is responding and the response status code.

Body for Request and Response





```
GET /WAP4.0/SelamServlet HTTP/1.0
```

```
HTTP/1.1 200
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 333
Date: Sun, 27 Dec 2020 18:44:00 GMT
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD><TITLE>SelamServlet</TITLE></HEAD>
<BODY>
<H1 align="center">SelamServlet</H1>
<H1>Selam via GET!</H1>
<H2>Sun Dec 27 21:44:00 EET 2020</H2>
…
</BODY></HTML>
```

Message Body - II

- There are rules regarding the message body such as
  - A response to **HEAD** never has a message body.
  - All **1xx** (Informational), **204 No Content**, and **304 Not Modified** responses do not include a message body.
  - All other responses do include a message body, although the body might be of zero length.

Message Body - III

- According to HTTP specification only method that can’t have a message body is **TRACE**.
- But HTTP spec says there is no defined semantics for having a message body for **GET/HEAD/DELETE** methods.
  - It says sending a payload body on a **GET/HEAD/DELETE** request might cause some existing implementations to reject the request.
- So request with **POST**, **PUT** and **PATCH** can have a message body.

Cacheable Response

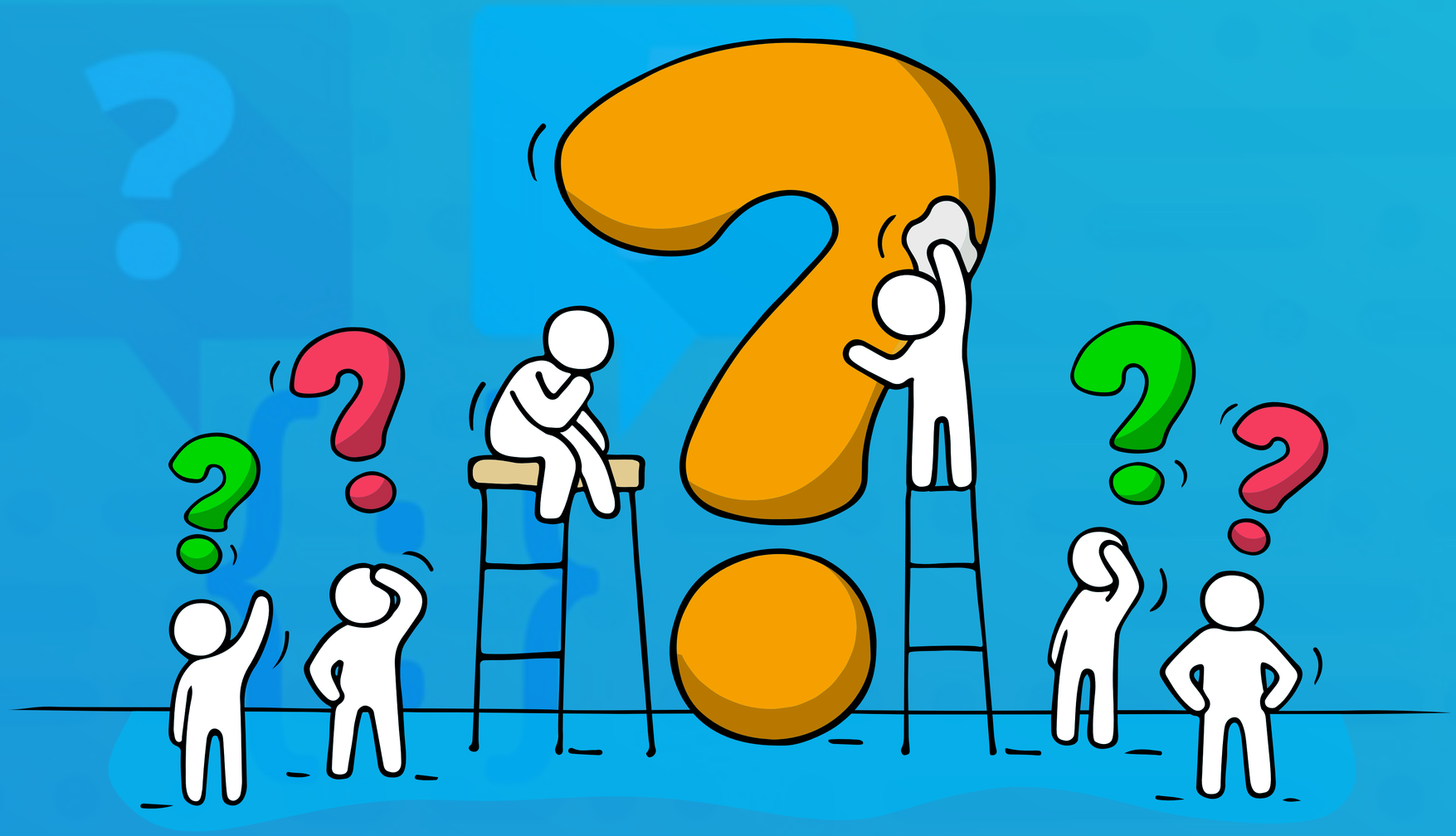
- If an HTTP response can be cached, i.e. is stored to be retrieved and used later it is called **cacheable response**.
- Cacheable response saves a new request to the server.
- Not all HTTP responses can be cached, methods whose response can be cached are **GET** and **HEAD**.
- Caching the response of other methods might be useless.
- The following status codes are cacheable: 200, 203, 204, 206, 300, 301, 404, 405, 410, 414, and 501.

Method Properties

Method	Safe	Idempotent	Message Body	Cachable
GET	yes	yes	no	yes
HEAD	yes	yes	no	yes
PUT	no	yes	yes	no
POST	no	no	yes	no
DELETE	no	yes	no	no
TRACE	yes	yes	no	no
PATCH	no	no	yes	no
OPTION	yes	yes	no	no

- So when designing resource methods always be predictable.
- We'll go into this in API design in more detail.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

GET

GET

- HTTP **GET** is to retrieve whatever information (in the form of an entity) is identified by the Request-URI.
- The semantics of the **GET** method change to a *conditional GET* if the request message includes an **`If-Modified-Since`**, **`If-Unmodified-Since`**, **`If-Match`**, **`If-None-Match`**, or **`If-Range`** header field.
- It is both safe and idempotent, it is not to modify a resource, other methods are for this.
- The response to a **GET** request is cacheable under some circumstances.

@GET - I

- **`javax.ws.rs.GET`** is a request method designator for HTTP **GET** requests.
- It is used to retrieve a resource or a set of resources depending on the path.
- The name of the resource method might be something like **`get`**, **`retrieve`**, **`fetch`**, etc.

@GET - II

- Path may refer to a collection of resources or only a resource.
- Resources are identified by paths and they are used in plural forms.
- A single resource is defined using its id on resource of URI.

```
@Path(“greetings")
public class GreetingResource {

   @GET
   public Map<String, String> getAllGreetings() { … }
}
```

```
@Path(“greetings")
public class GreetingResource {

   @GET
   @Path("languages")
   public Set<String> getAllLanguages() { … }
}
```

```
@Path(“greetings")
public class GreetingResource {

   @GET
   @Path("{language}/1")
   public String getGreeting1(@PathParam("language") String language) { … }
}
```

Path for @GET - I

- So for greeting resource the path would be **`greetings`**.
  - If the path of the resource method with **`@GET`** does not have a path parameter then all instances of the resource are returned in a collection.
  - So a **GET** request to **`greetings`** path would return all greeting resources as a collection.
- **GET** can also retrieve sub-resources such as **`greetings/languages`**.

```
@Path(“greetings")
public class GreetingResource {
  @GET
  public Map<String, String> getAllGreetings() { … }
}
```

```
@Path(“greetings")
public class GreetingResource {
   @GET
   @Path("languages")
   public Set<String> getAllLanguages() { … }
}
```

Path for @GET - II

- There might be several ways to select only one or several resources:
  - **Path parameter**: The path of the resource method with **`@GET`** can have a path parameter or parameters: `greetings/{language}`
  - **Query parameter**: If the path of the resource method with **`@GET`** can have a query parameter or parameters: `greetings?language=XXX`
- In both case the convention is that only matching instance or instances of the resource are returned.

Path for @GET - III

- If path parameter is used it is mostly the id of a resource so only one matching resource is returned.
- But if one or more query parameters are used then one or more matching resources are returned.

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.GreetingResource`
- Run `getGreeting1()`, `getGreeting2()`, `getGreeting3()`, and `getGreeting4()` methods.
- Notice `getGreeting4()` method uses a query string.

Returning from @GET - I

- If the requested resource(s) is found on the server then **200 OK** status code is returned along with the resource(s).
- JAX-RS normally returns **200 OK** status code automatically if everything goes fine and the resource method returns the resource(s).
- If the requested resource(s) is not found on the server then **404 NOT FOUND** status code may be returned.
  - JAX-RS does not return it automatically, it needs to be coded.

Returning from @GET - II

- Or alternatively **200 OK** status code along with a string that states that the resource is not found may be returned.

- If the requested resource(s) is not found on the server and `null` is returned then the server automatically returns **204 NO CONTENT** status with no message body.

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.GreetingResource`
- Run paths with `@GET` annotations.
- Notice methods return different types including `Response` object.
- They all return in JSON except one case where it is designated to return XML.
  - JAX-RS always set the status of the response **200 Ok** automatically if it is not set in `Response` object.

MIME Type for Return - I

- Remember that if `@Produces` doesn't exist, JAX-RS may render the response in any type.
- In default i.e. when `@Produces` doesn't exist JAX-RS uses
  - plain text for the representation of primitives and strings which results in `text/plain` in `Content-Type` header of the response.
  - JSON for Java objects, collections, enum types, temporal types such as `java.util.Date` and types in `java.time` which results in `application/json` in `Content-Type` header of the response.

MIME Type for Return - II

- If the `Content-Type` header of the response is `text/plain` then all returned values are in plain text.
  - No value with a specific type is available in plain text.
  - Even Java's `String` types would be represented as a text without "".
- But if `Content-Type` is `application/json` then return values retain their own types in JSON.

EmployeeResource

- **REST Examples Ch06** project.
- `org.javaturk.rest.ch06.resource.EmployeeResource`
  - Run methods with `@GET` and observe the `Content-Type` and the message body of the response.
  - Put `@Produces("application/json")` on the class and run methods with `@GET` and observe the `Content-Type` and the message body of the response.

How To Return Resource - I

- Returning resources from the resource methods with `@GET` there are mainly two alternatives:
  - Returning the bare resource itself,
  - Returning the resource inside a wrapper of `Response`.

How To Return Resource - II

```
@GET
@Path("{language}/1")
public String getGreeting1(@PathParam("language") String language){
   String greeting = repo.getGreeting(language);
   if(greeting ≠ null)
     return "\"" + greeting + "\"";
   else
     return "No such language found: " + language;
}
```

```
@GET
@Path("{language}/2")
public String getGreeting2(@PathParam("language") String language){
   String greeting = repo.getGreeting(language);
   if(greeting ≠ null)
     return jsonb.toJson(greeting);
   else
     return jsonb.toJson("No such language found: " + language);
}
```

```
@GET
@Path("{language}/3")
public Response getGreeting3(@PathParam("language") String language) {
   String greeting = repo.getGreeting(language);
   if(greeting ≠ null)
     return Response.status(200).entity(jsonb.toJson(greeting)).build();
   else
     return Response.status(404).entity(jsonb.toJson("No such language found: " + language)).build();
}
```

How To Return Resource - III

- There are several trade offs in this.
  - Returning the bare resource itself might seem better because of the fact thet REST is resource-oriented and returning resources is a natural choice.
  - Returning the resource inside a wrapper of **`Response`** is better because it gives more control on what is returned.
    - Status code and headers can be set in **`Response`**.
  - We'll talk about this later in API design.

How To Return Resource - IV

- Best practice would be keeping the response simple.
- So if there is no need to control response code and add header information then there is no need to use `Response` object.
  - Let the JAX-RS arrange status code for your response automatically.
- We'll talk about this later in API design.

selsoft

build better, deliver faster

Exercise

Exercise

- Twitter provides an extensive set of REST services.
  - https://developer.twitter.com/en/docs/twitter-api
- **Twitter Standard API** provides many endpoints for its resources such as users, tweets, followers, messages, etc.
  - https://developer.twitter.com/en/docs/twitter-api/v1
- **Twitter API v2** is in early access and getting improved.
  - https://developer.twitter.com/en/docs/twitter-api/early-access

Exercise

- To be able to make a request to Twitter API you need to have a **Twitter developer account**.
- When you have a Twitter developer account you will be able to create projects in **Twitter Developer Dashboard** and **API keys** necessary for authorization of REST requests to Twitter API.
- Twitter REST API uses **OAuth 1.0** and **OAuth 2.0 Bearer Token** authentication for access.
- Using Twitter Developer Dashboard you can import all REST API for v.2 into Postman and enter API keys for authorization so that you can make requests using Postman.

Exercise

- Twitter Standard API allows to post tweets but Twitter API v2 doesn’t yet.
  - So to post tweets use Standard API.
- Twitter Developer Dashboard allows you to import all REST API for v.2 into Postman.
  - There is no such utility for Standard API .

Exercise

https://api.twitter.com/2/users/by/username/kaldiroglu

```
{
    "data": {
        "id": "95688001",
        "name": "Akin Kaldiroglu",
        "username": "kaldiroglu"
    }
}
```

https://api.twitter.com/2/users/95688001/following

```
…
{
     "id": "6108292",
      "name": "Sam Newman",
      "username": "samnewman"
},
{
      "id": "1302596158859153408",
      "name": "Türkiye Felsefe Kurumu",
      "username": "TRFelsefeKurumu"
}
…
```

https://api.twitter.com/2/tweets/1352516947284553729

```
{
    "data": {
        "id": "1352516947284553729",
        "text": "Bu sene salgının sağladığı zamanda tamamen yenilediğim Java ile Nesne-Merkezli Programlamaya Giriş isimli
programlamaya Java ile giriş yapan eğitimin slideları ve kodları şuradadır:  https://t.co/0ll0FkbPp9"
    }
}
```

Exercise

- So do following for this and coming exercises:
  - Apply for a Twitter developer account
  - Create a project and API keys and save them somewhere
  - Walk around the REST API
    - For REST API v.2 you can import calls into Postman

Exercise

- Twitter REST API utilizes many **GET** endpoints.
  - `GET statuses/show/:id`
  - `GET statuses/oembed`
  - `GET statuses/lookup`
  - `GET statuses/retweets/:id`
  - `GET statuses/retweets_of_me`
  - `GET statuses/retweeters/ids`
  - `GET favorites/list`
- And then make **GET** requests such as getting your user information, your followers, your tweets, etc.

selsoft

build better, deliver faster

Exercise

Exercise

- Google has tons of REST APIs for its different services.
  - https://developers.google.com/apis-explorer
- You can register to use its APIs.
- Explore APIs and play with them.

selsoft

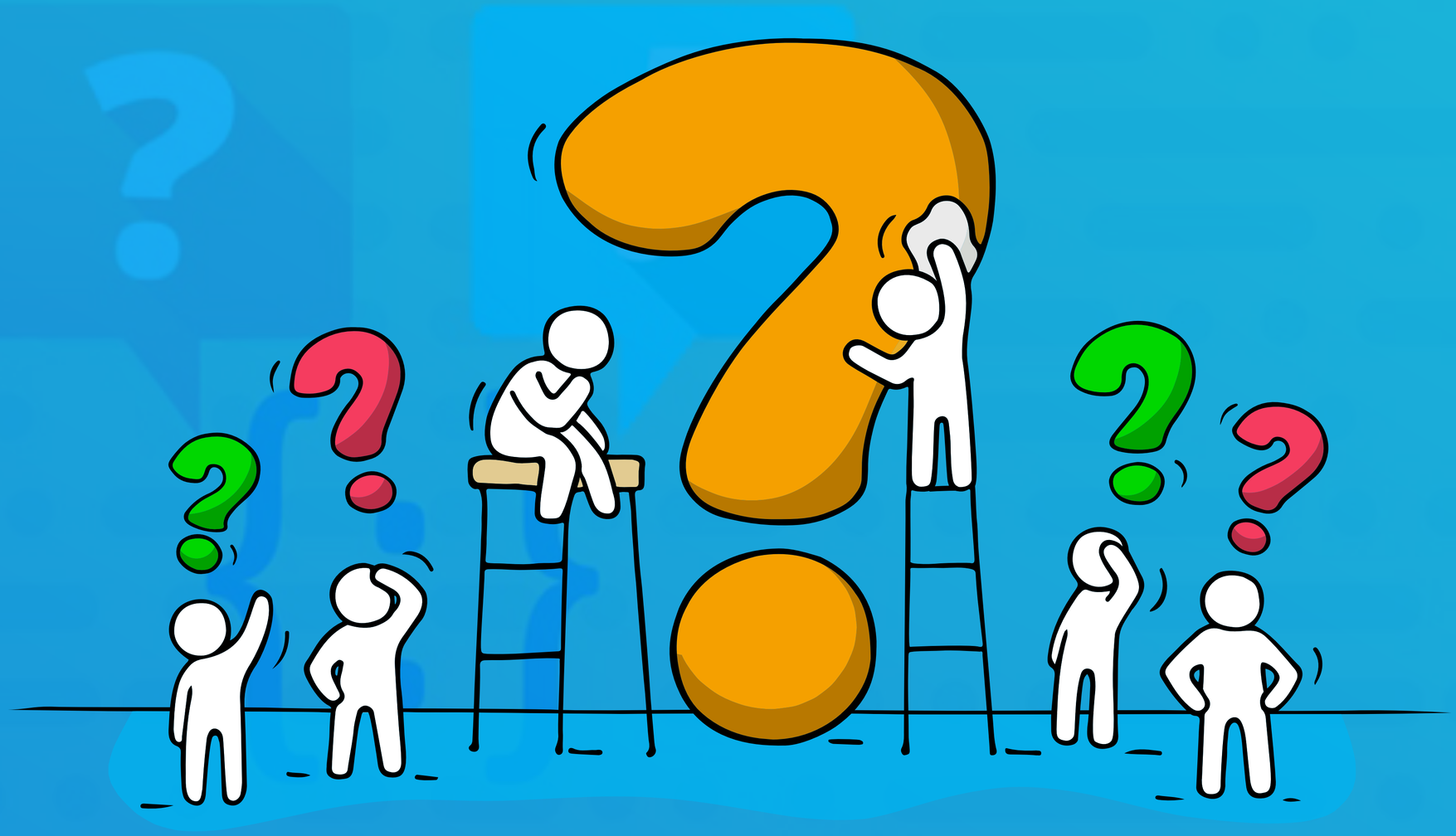
build better, deliver faster

Exercise

Exercise

- In Internet there are many open and free REST services such as:
  - https://openlibrary.org/developers/api
  - https://openweathermap.org/api
- But most of them are read only i.e allow only **GET**.
- There are many premium REST services too.
- You can explore their APIs and play with them.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

HEAD

HEAD - I

- HTTP **HEAD** method is the same as **GET** method except that it only retrieves status line and headers without any message-body.
- The metainformation contained in the HTTP headers in response to a **HEAD** request should be identical to the information sent in response to a GET request.
- This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself.
- This method is often used for testing hypertext links for validity, accessibility, size of the resource and recent modification.

HEAD - II

- **HEAD** requests are safe, idompotent and its responses are cacheable.

@HEAD - I

- `@HEAD` is used to deginate resource methods that produce response for **HEAD** requests.
- JAX-RS provides a default implementation for **HEAD** requests that dispatches to the resource method annotated with `@GET` that only returns status line and headers, discards the returning entity.
- If there is no corresponding method with `@GET` then the server returns **405 Method Not Allowed**.

@HEAD - II

- Response to a **HEAD** request should not have a body.
  - If it does it is simply ignored by clients.

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.GreetingResource`
- Call paths with `@GET` annotations using **HEAD**.

selsoft

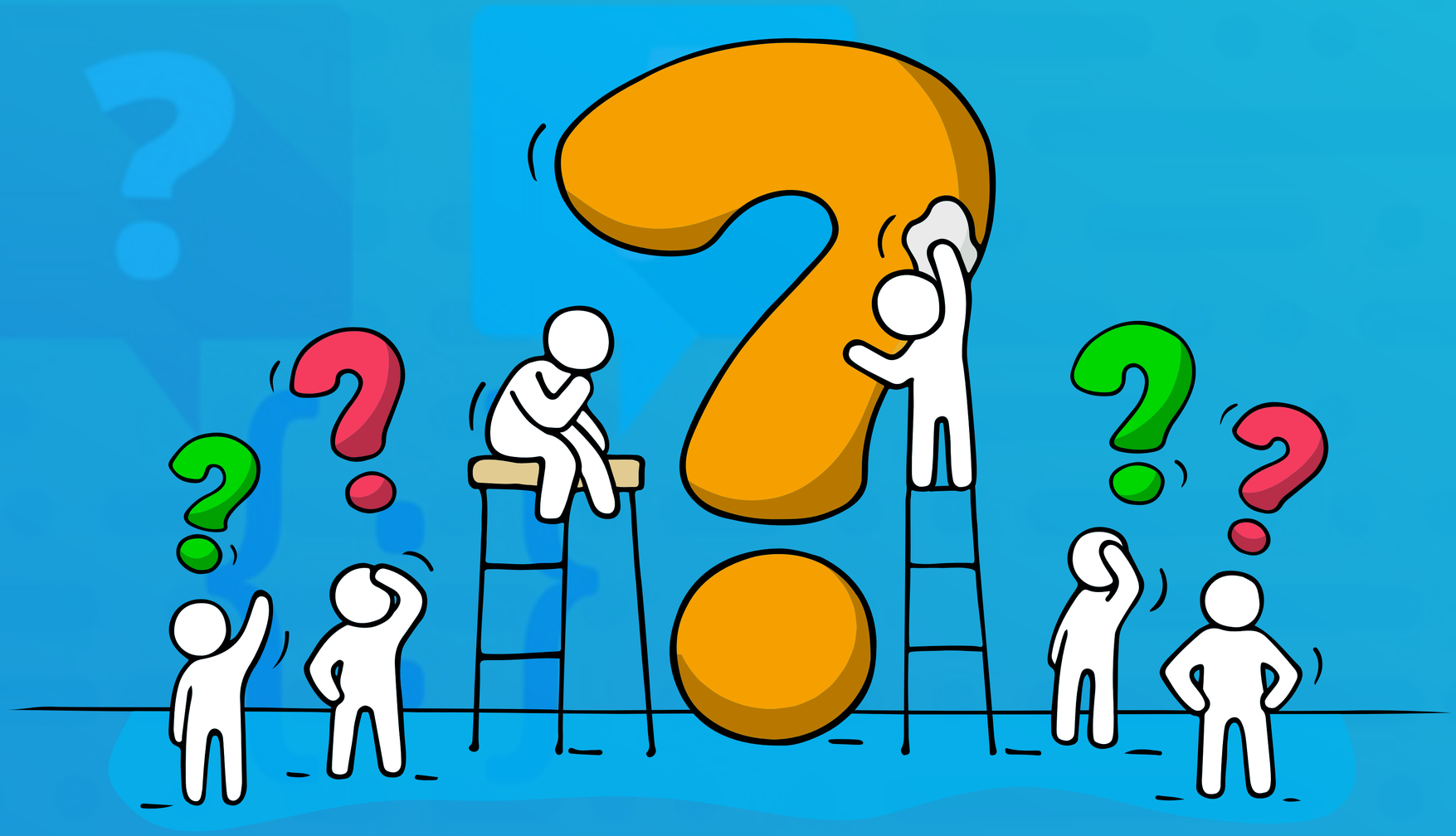
build better, deliver faster

Exercise

Exercise

- Make **HEAD** requests to the URIs for **GET** such as getting your user information, your followers, your tweets, etc.
- Does Twitter API allow **HEAD** requests?

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

POST

POST

- HTTP **POST** method is used to post data to the server.
- For REST it is used to create a new resource on the server.
- **POST** is neither safe nor idempotent.
- So the **POST** requests always add a new resource to a collection of resources.
- The name of the resource method might be something like `create`.

@POST - I

- `@POST` deginates resource methods that create a new resource.
- The URI of the `@POST` methods would generally be the resource name which is a collection and request carries the entity in its body as JSON.
- In this case it is declared as an entity parameter in create methods.

```
@Path("greetings")
public class GreetingResource {

  @POST
  public Response createGreetingByEntity(Greeting greeting, @Context UriInfo uriInfo) { … }
}
```

```
POST /GreetingRest/resources/greetings HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:8888
…
Content-Length: 112

{"greeting":"Χαίρετε","language":"greek","typeName":"Greeting [language=greek, greeting=Χαίρετε]"}
```

@POST - II

- Other alternatives to post data to the server for the new resource might be using parameters:
  - Path parameters can be passed to the server for the new resource.
  - URI may include a template if path variables are used.

```
@Path("greetings")
public class GreetingResource {

   @POST
   @Path("{language}/{greeting}")
   public Response createGreetingByParameter(@PathParam("language") String language,
                                             @PathParam(“greeting") String greeting, @Context UriInfo uriInfo){ … }
}
```

```
POST /GreetingRest/resources/greetings/arabic/as-salamu%20alaikum HTTP/1.1
Accept: application/json
User-Agent: Jersey/2.27 (HttpUrlConnection 11.0.7)
Host: localhost:8888
Connection: keep-alive
```

@POST - III

- Using path parameters can be suitable only for simple objects.

- We don’t consider query string as a good alternative!

@POST - IV

- It is better to use entity in **POST** requests for many reasons:
  - HTTP specification always talks about “enclosed entity” in requests,
  - Entity is an encapsulated content that represents the resource but parameters are just separate values,
  - Sending data as an entity in payload provides security in case of HTTPS while using parameters makes values totally visible and thus unprotected,

@POST - V

- In case of parameters, path, query or other kinds, there might be a limit enforced by browsers, servers or even intermediaries.
  - This concern is studied before.

Posting Data

- When posting large amount of data to the server, there might be a concern for limits that the servers enforce.
- For example Tomcat has an attribute `maxPostSize` for connectors.
  - Its default value is 2 MB.

Id For New Resource

- For **POST**, most of the time it is the server that creates the unique id for the newly created resource.
- But sometimes the client may need to decide about id too,
  - So the client sets the id on the entity before posting it to the server.

Returning from @POST - I

- After creating the resource, the server may send the created resource back to the client with **201 Created** status code.
- In the response it is better to include the newly created resource and URI to that resource.
  - For the location of the newly created resource **Location** header is used.
  - For this purpose `created(String URI)` method on `Response` is called.
    - Passed URI is set as the value of **Location** header.

Returning from @POST - II

- If creating a new resource fails for example due to a unique id conflict **409 Conflict** should be returned.

GreetingRest

- **GreetingRest** project.
- Call the path `greetings` on `GreetingResource` with a **POST** using **GreetingRestClient** project.

selsoft

build better, deliver faster

Exercise

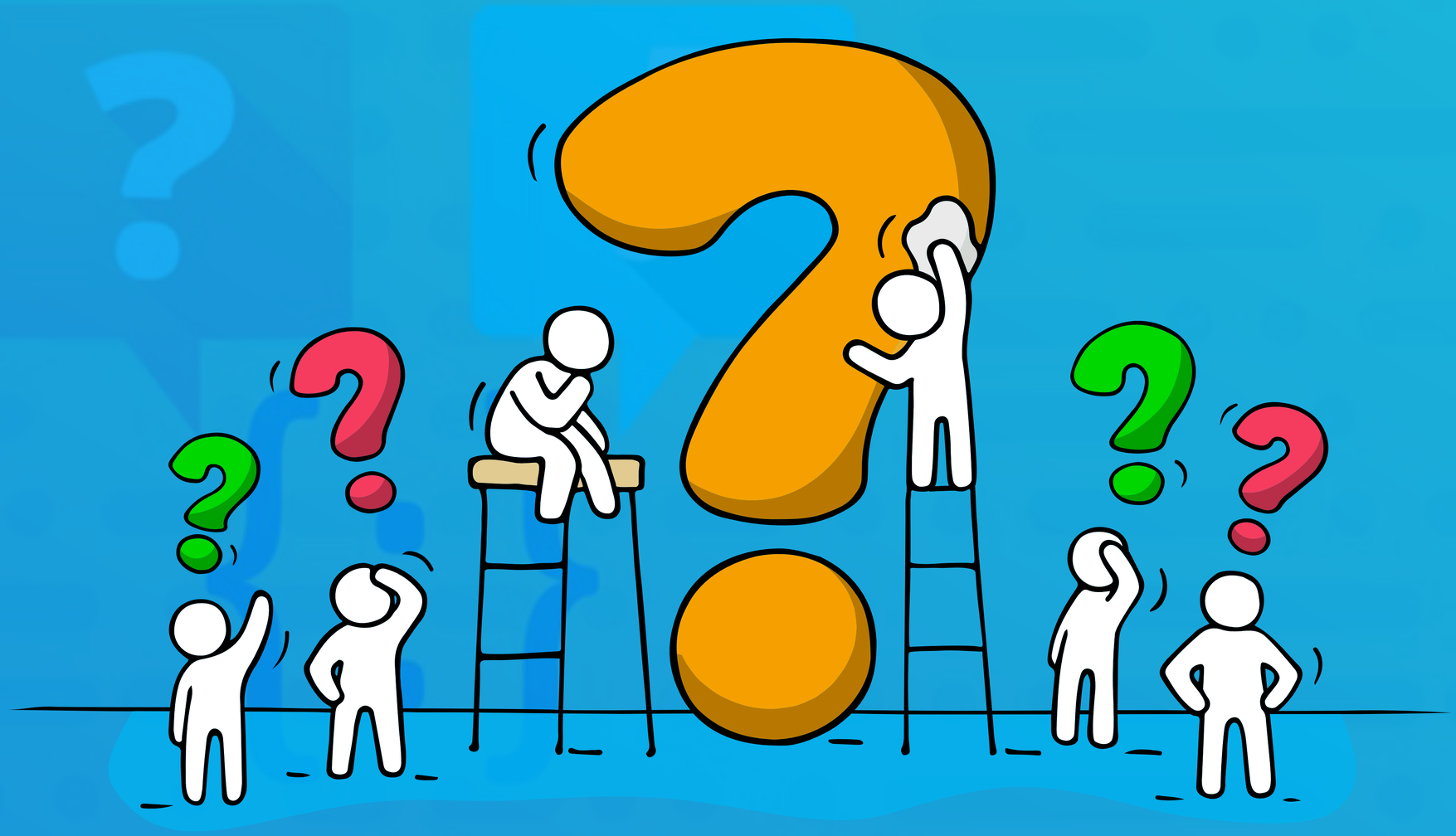
Exercise

- Twitter Standard API has following endpoints:
  - `POST statuses/update`
  - `POST statuses/destroy/:id`
  - `POST statuses/retweet/:id`
  - `POST statuses/unretweet/:id`
  - `POST favorites/create/:id`
  - `POST favorites/destroy/:id`
- It uses **POST** to tweet a new status, to retweet or favorite (like) an existing tweet.
- How do you interpret the use of **POST** for deleting a tweet, retweet or a favorite?

Exercise

- Make a **POST** request to tweet using Twitter Standard API.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

PUT

PUT - I

- HTTP specification says:

  The **PUT** method requests that the enclosed entity be stored under the supplied Request-URI. If the Request- URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server.

- In REST since **POST** is used to create a new resource, **PUT** is mainly used to replace an existing resource.

PUT - II

- If the server doesn’t have the resource that is intented to be replaced, the server may choose to create a new one by using the provided resource.
- PUT is not safe but idempotent.
  - It is because after each time it is called always a completely new version of the resource exists on the server.

PUT vs. POST - I

- Main difference between **PUT** and **POST** is visible in their URIs:
  - While in **POST** URI identifies the collection of the resource, the URI in a **PUT** request identifies the entity enclosed with the request.

```
@Path("greetings")
public class GreetingResource {

  @POST
  public Response createGreetingByEntity(Greeting greeting, @Context UriInfo uriInfo) { … }
}
```

```
@Path("greetings")
public class GreetingResource {

   @PUT
   @Path("{language}")
   public Response replaceGreetingByEntity(@PathParam("language") String language, Greeting greeting,
                                           @Context UriInfo uriInfo) { … }
}
```

PUT vs. POST - II

- So always provide **POST** resource methods to create a totally new resource on the server.
- If replacing an existing resource is part of your business then provide **PUT** resource methods to replace an existing resource.
  - That’s why **PUT** resource methods are not that common.

Irreplacable Resources

- For irreplacable resources, **PUT** method should not be provided.
- For some business, either a new resource is created or an existing one is deleted, other than retrieving them.
- In this case **PUT** is not provided so no resource can be replaced.

@PUT - I

- `@PUT` deginates resource methods that replace a resource on the server.
- The name of the resource method might be something like `replace`.
- In **PUT** id of the resource is sent via path parameter while the new version of the resource is sent to the server as an entiy in request body.
- So the entity is declared in replace methods.

@PUT - II

- The URI of a **PUT** request should refer to the id of the resource to be replaced under the collection.

```
@Path("greetings")
public class GreetingResource {

   @PUT
   @Path("{language}")
   public Response replaceGreetingByEntity(@PathParam("language") String language, Greeting greeting,
                                           @Context UriInfo uriInfo) { … }
}
```

```
PUT /GreetingRest/resources/greetings/turkish HTTP/1.1
Accept: application/json
Content-Type: application/json
User-Agent: Jersey/2.27 (HttpUrlConnection 11.0.7)
Host: localhost:8888
Connection: keep-alive
Content-Length: 102

{"greeting":"merhaba","language":"turkish","typeName":"Greeting [language=turkish, greeting=merhaba]"}
```

@PUT - III

- It is also possible to send the new version of the resource using path parameters.
- In this case the resource method does not receive any entity.

```
@Path("greetings")
public class GreetingResource {

   @PUT
   @Path("{language}/{greeting}")
   public Response replaceGreetingByParameter(@PathParam("language") String language,
                                              @PathParam(“greeting") String greeting, …) { … }
}
```

```
PUT /GreetingRest/resources/greetings/turkish/merhaba HTTP/1.1
Accept: application/json
User-Agent: Jersey/2.27 (HttpUrlConnection 11.0.7)
Host: localhost:8888
Connection: keep-alive
```

@PUT - IV

- But this form of **PUT** is against the rules of the HTTP specification.
  - When the client does not send an entity in **PUT** request, JAX-RS gives an error: **`java.lang.IllegalStateException`**: Entity must not be null for http method **PUT**.
  - To suppress this error JAX-RS provides a client property, **`SUPPRESS_HTTP_COMPLIANCE_VALIDATION`**.

```
…
client = ClientBuilder.newClient();
client.property(ClientProperties.SUPPRESS_HTTP_COMPLIANCE_VALIDATION, true);
```

- So don’t use **PUT** this way.

Returning from @PUT - I

- For returning from `@PUT`:
  - If the resource is replaced successfully with the new version then the server returns **200 Ok** status code.
  - If the server creates a new resource then as with **POST** it sends back to the client new resource as an entity, its URI along with **201 Created** status code.
    - For this purpose `created(String URI)` method on `Response` is called.
    - Passed URI is set as the value of **Location** header.

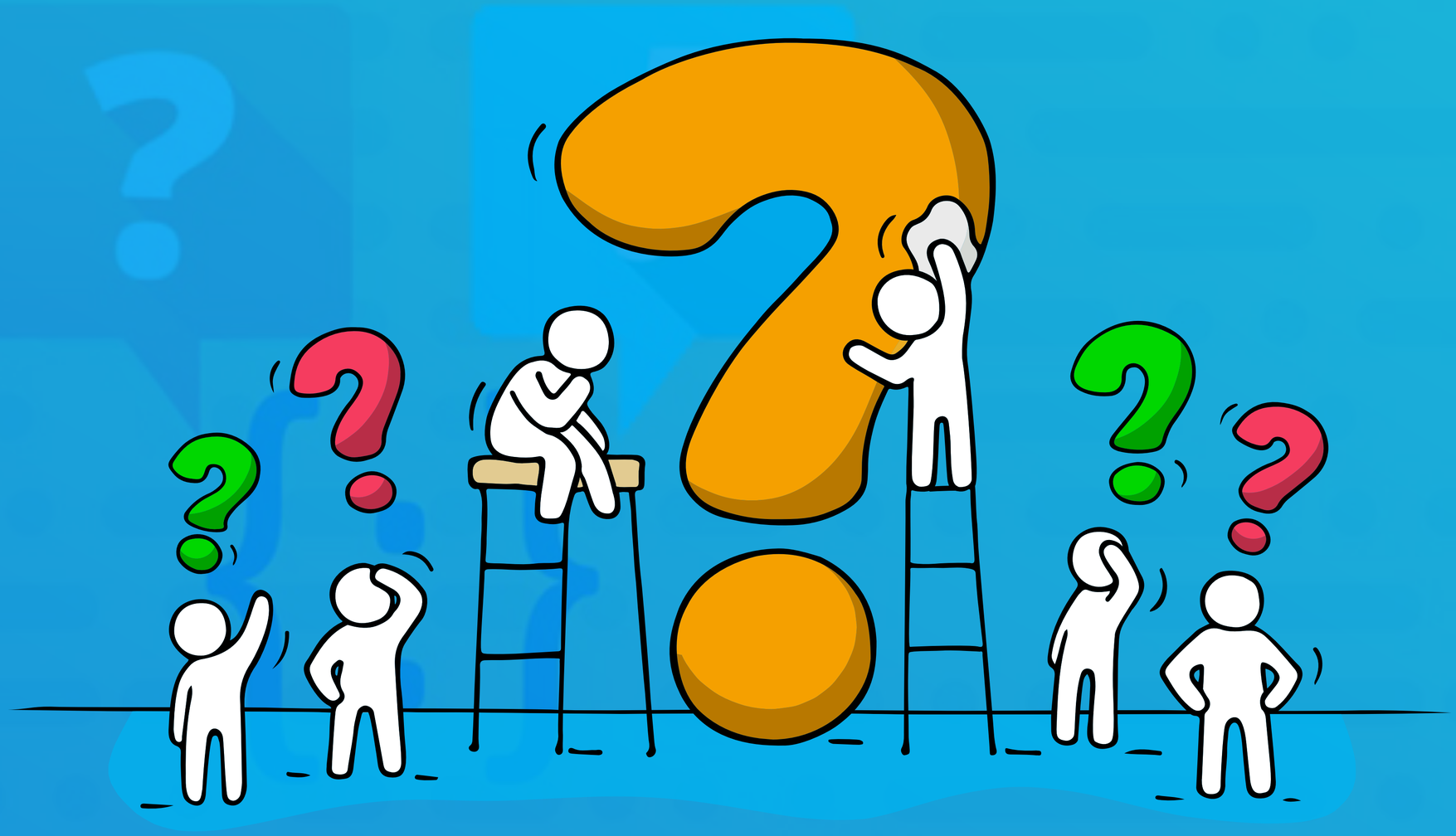
Returning from @PUT - II

- If replacing the resource fails **409 Conflict** should be returned.

GreetingRest

- **GreetingRest** project.
- Call the path `greetings` on `GreetingResource` with a **PUT** method using **GreetingRestClient** project.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

PATCH

PATCH - I

- HTTP specification says:

  The PATCH method is similar to PUT except that the entity contains a list of differences between the original version of the resource identified by the Request-URI and the desired content of the resource after the PATCH action has been applied. The list of differences is in a format defined by the media type of the entity (e.g., "application/diff") and MUST include sufficient information to allow the server to recreate the changes necessary to convert the original version of the resource to the desired version.

PATCH vs. PUT - I

- HTTP specification says:

  The PUT method is already defined to overwrite a resource with a complete new body, and cannot be reused to do partial changes. Otherwise, proxies and caches, and even clients and servers, may get confused as to the result of the operation.

PATCH vs. PUT - II

The difference between the PUT and PATCH requests is reflected in the way the server processes the enclosed entity to modify the resource identified by the Request-URI. In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced. With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version. The PATCH method affects the resource identified by the Request-URI, and it also MAY have side effects on other resources; i.e., new resources may be created, or existing ones modified, by the application of a PATCH.

PATCH - II

- In HTTP specification **PATCH** method is described similar to **PUT**.
- **PATCH** is used to update the resource i.e. modify only some parts of the resource while **PUT** replaces the whole resource.
  - **PATCH** is partial modification while **PUT** is a whole replacement.
- **PATCH** method uses unique identifier of the resource to update it.
- **PATCH** is neither safe nor idempotent while **PUT** is idempotent.
  - It is not idempotent because of the fact that it may modify different fields of the resource each time it is called.

PATCH - II

- In HTTP specification **PATCH** method is described similar to **PUT**.
- **PATCH** is used to update the resource i.e. modify only some parts of the resource while **PUT** replaces the whole resource.
  - **PATCH** is partial modification while **PUT** is a whole replacement.
- That's why **PATCH** method is the hardest one among HTTP methods due to the nature of partial update.
  - How do you express which parts need to be updated?

@PATCH

- **`@PATCH`** deginates resource methods that update a resource on the server.
- So the URI of a **PATCH** request always refers to a specific resource through its unique id.

```
@Path("employees")
public class EmployeeResource {
    …
    @PATCH
    @Path("{id}")
    public Response updateEmployee(@PathParam("id") long id, …) { … }
}
```

- The name of the resource method might be something like **`update`**.

Patch Information - I

- How to send patch information to the server?
- Although the spec is talking about the list of differences and a format that includes sufficient information to allow the server to recreate the desired version there is no standard way of doing it.
- There are several alternatives:
  - Sending a whole, updated entity to the server,
  - Sending only new values for the fields to be updated with field information,
  - Other ways?

Patch Information - II

- If **PATCH** methods receive a path parameter for the id of the resource to be updated and the updated entity, then **PATCH** would be just like **PUT**.
  - In this case how to know which fields to be udated?
    - Overriding all fields is not a good choice if we keep track of the versions of the resources where changes are traced.
  - Moreover it requires first a **GET** then modify the entity and **PATCH** it to the server.

```
@PATCH
@Path("{id}")
public Response updateEmailByEntity(@PathParam("id") long id, Employee employee, …) { … }
```

Patch Information - III

- On the other hand to update only specific fields, the client needs to know which fields exist and which of them are updatable.
- What happens if a new updatable field is added to the resource on the server?
- How do all these need to be documented and how do the clients know about them?

```
public class Employee {
     private long id;
     private String name;
     private char sex;
     private String email;
     private LocalDateTime dob;
     private boolean fullTime;
     private ZonedDateTime startingDate;
     private EmployeeRank rank;
     private double salary;
     private Department department;
   …
}
```

Patch Information - IV

- For **PATCH** it might seem easier for the resource methods to receive path parameters for the id of the resource and the value for the field to be updated.

```
@Path("employees")
public class EmployeeUpdateResource {

   @PATCH
   @Path("{id}/{email}")
   public Response updateEmailByParameter(@PathParam("id") long id, @PathParam("email") String email, …) { … }
}
```

```
PATCH /REST_Examples_Ch06/resources/employees/1/new_email@company.com HTTP/1.1
Accept: application/json
User-Agent: Jersey/2.27 (HttpUrlConnection 11.0.7)
Host: localhost:8888
Connection: keep-alive
```

Patch Information - V

- There are problems with this approach:
  - First problem with this is that it is against the specification of HTTP.
  - Another problem is that there need to be lots of update methods to update different fields.
- How about if more than one field need to be updated together for the consistency of the entity?

Patch Information - VI

- Google created its own solution called **field mask**.
- It is a notation to point to fields in resources or nested resources for the purpose of partial retrieval or update.
  - https://developers.google.com/protocol-buffers/docs/reference/java/com/google/protobuf/FieldMask

Returning from @PATCH

- For returning from `@PATCH` it is best to return back to the client the new respresentation of the resource with **200 OK** status code.
  - Or only **204 No Content** can be returned.
- If updating the resource fails such as the entity does not exists, **409 Conflict** should be returned.

Unpopular PATCH - I

- In practice **PATCH** is not a common method to use in REST world due to several reasons:
- First, partial update may not be a natural choice for the business.
  - Most of the time replacing a resource with the new one with the same id would be more suitable for business needs.
    - Instead of updating an order, replacing it completely might be a more natural and simpler solution.
    - Sometimes even **PUT** is not applicable for resources of most business.

Unpopular PATCH - II

- Second reason is the fact that most REST APIs use **PUT** for all kinds of updates i.e. for both whole replacement and partial update and avoid **PATCH**.
- This is mostly due to the natural difficulties and lack of a standard in the notation of partial update.
- But using **PATCH** for update and sparing **PUT** only to a total replacement is what conforms to the standard and thus is the best practice.

Unmodifiable Resources

- For unmodifiable resource, **PATCH** method should not be provided.

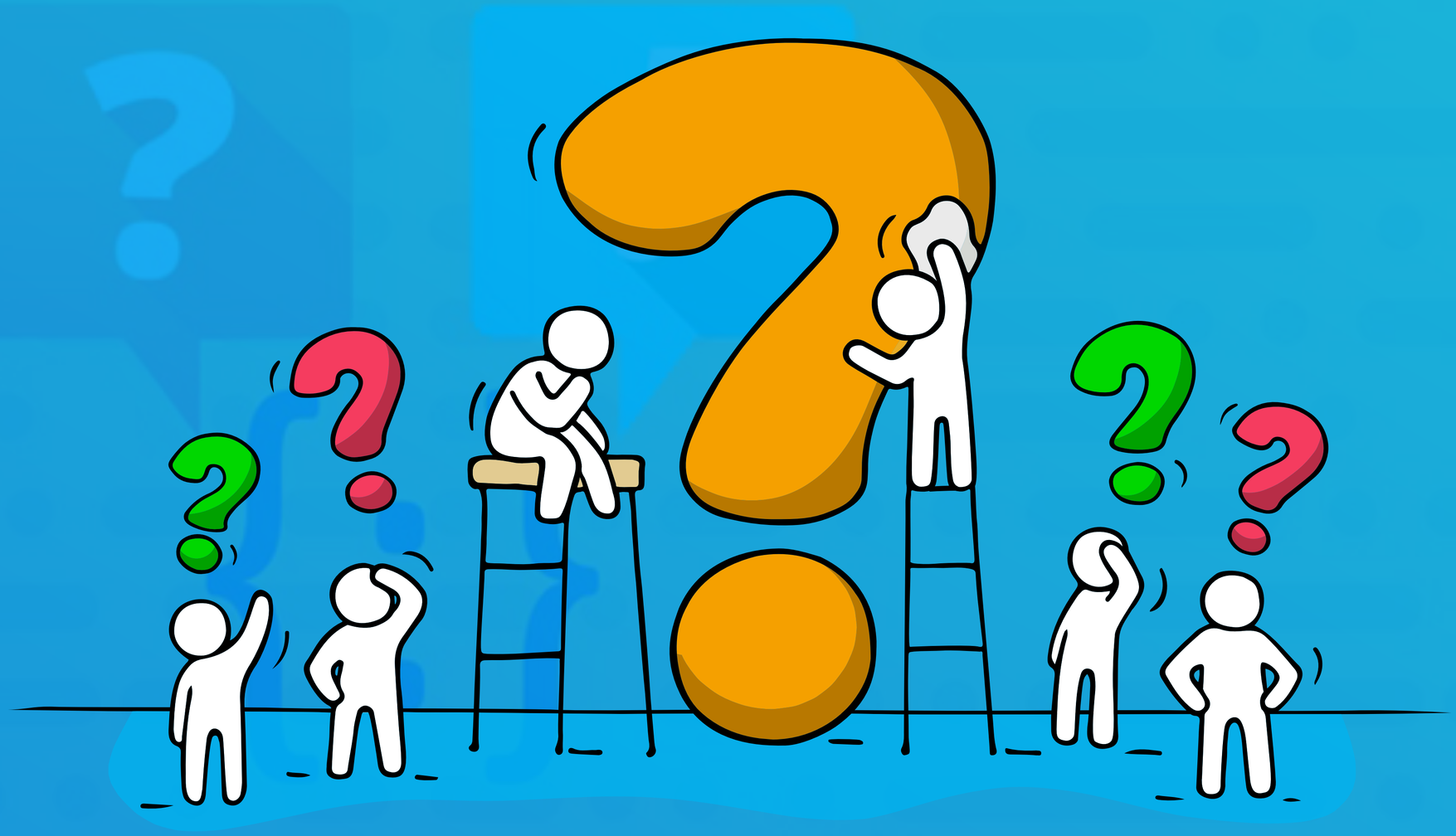
GreetingRest

- **GreetingRest** project.
- Call the path `greetings` on `GreetingResource` with a **PATCH** method using **GreetingRestClient** project.

Rest Example Ch06

- **REST Example Ch06** project.
- Call the paths of **PATCH** resource methods in **`EmployeeUpdateResource`** with using **REST Example Ch06 Client** project.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

DELETE

DELETE

- **DELETE** method requests that the server delete the resource identified by the URI.
- It can be provided for the resource that is deletable.
- The server may delete the resource or make it inaccessible for the clients.

@DELETE - I

- **`@DELETE`** deginates resource methods that delete a resource on the server.
- So the URI of a **DELETE** request always refers to a specific resource through its unique id.

```
@Path("employees")
public class EmployeeResource {
    …
    @DELETE
    @Path("{id}")
    public Response deleteEmployee(@PathParam("id") long id, …) { … }
}
```

```
DELETE /REST_Examples_Ch06/resources/employees/1 HTTP/1.1
Accept: application/json
User-Agent: Jersey/2.27 (HttpUrlConnection 11.0.7)
Host: localhost:8888
Connection: keep-alive
```

@DELETE - II

- The name of the resource method might be something like **`delete`** or **`remove`**.

Returning from @DELETE

- For returning from `@DELETE` it is best to return back to the client with **200 OK** status code maybe with an entity explaining the status if the method is successfull.
  - Or only **204 No Content** can be returned.
- **202 Accepted** can also be returned if the action has not yet been enacted.
- If deleting the resource fails **409 Conflict** should be returned.

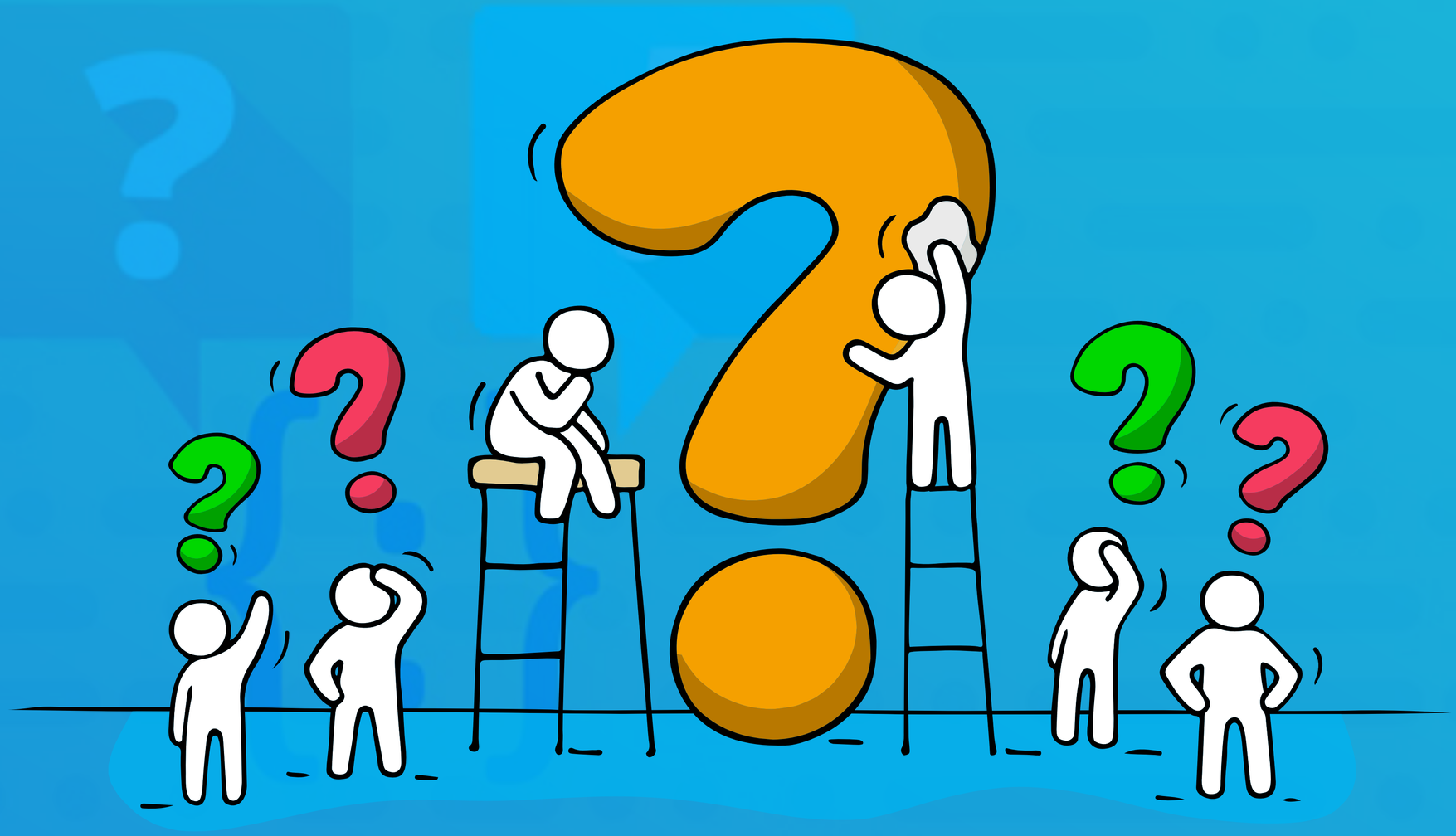
GreetingRest

- **GreetingRest** project.
- Call the path `greetings` on `GreetingResource` with a **DELETE** method using **GreetingRestClient** project.

Rest Example Ch06

- **REST Example Ch06** project.
- Call the paths of **DELETE** resource methods in **`EmployeeUpdateResource`** with using **REST Example Ch06 Client** project.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

OPTIONS

OPTIONS

- HTTP **OPTIONS** method represents a request for information about the communication options available on the request/response chain identified by the Request-URI.
- It allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.
- **OPTIONS** requests are safe, idompotent but its responses are not cacheable.

@OPTIONS - I

- `@OPTIONS` is used to deginate resource methods that produce response for **OPTION** requests.
- JAX-RS provides a default implementation for **OPTIONS** requests if the resource doesn’t have an explicit implementation.
- The default implementation sets the **Allow** response header to all the HTTP method types supported by the resource.
-

@OPTIONS - I

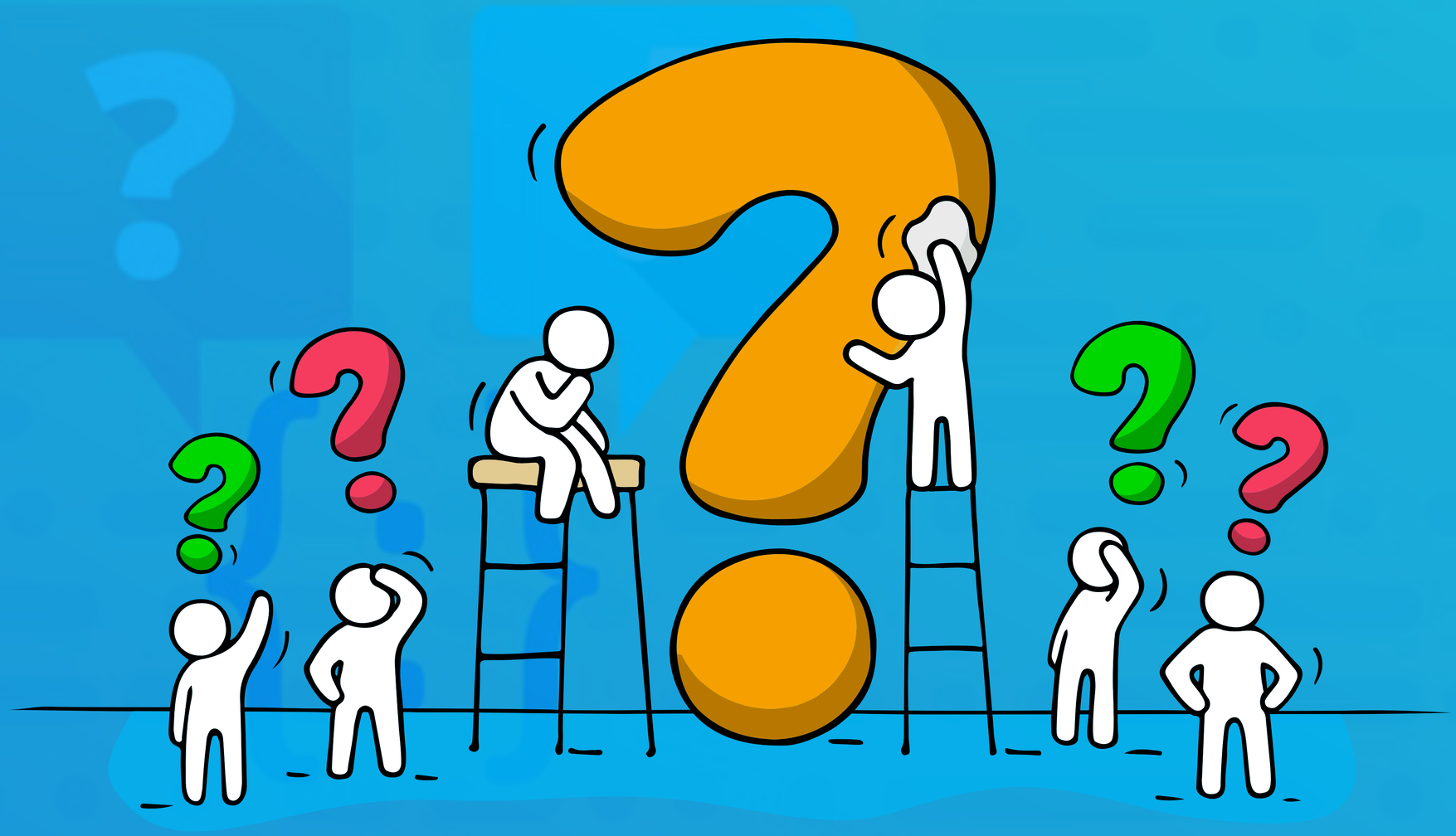
- A custom implementation might set **Allow** response header by calling one of two `allow()` methods on `Response` object.

```
@OPTIONS
public Response getOptions() {
   return Response.status(200).allow("GET", "HEAD", "POST", "PUT", "PATCH", "OPTIONS").build();
}
```

GreetingRest

- **GreetingRest** project.
- `org.javaturk.rest.greet.GreetingResource`
- Call paths with `@OPTIONS` annotations.

Time for questions!

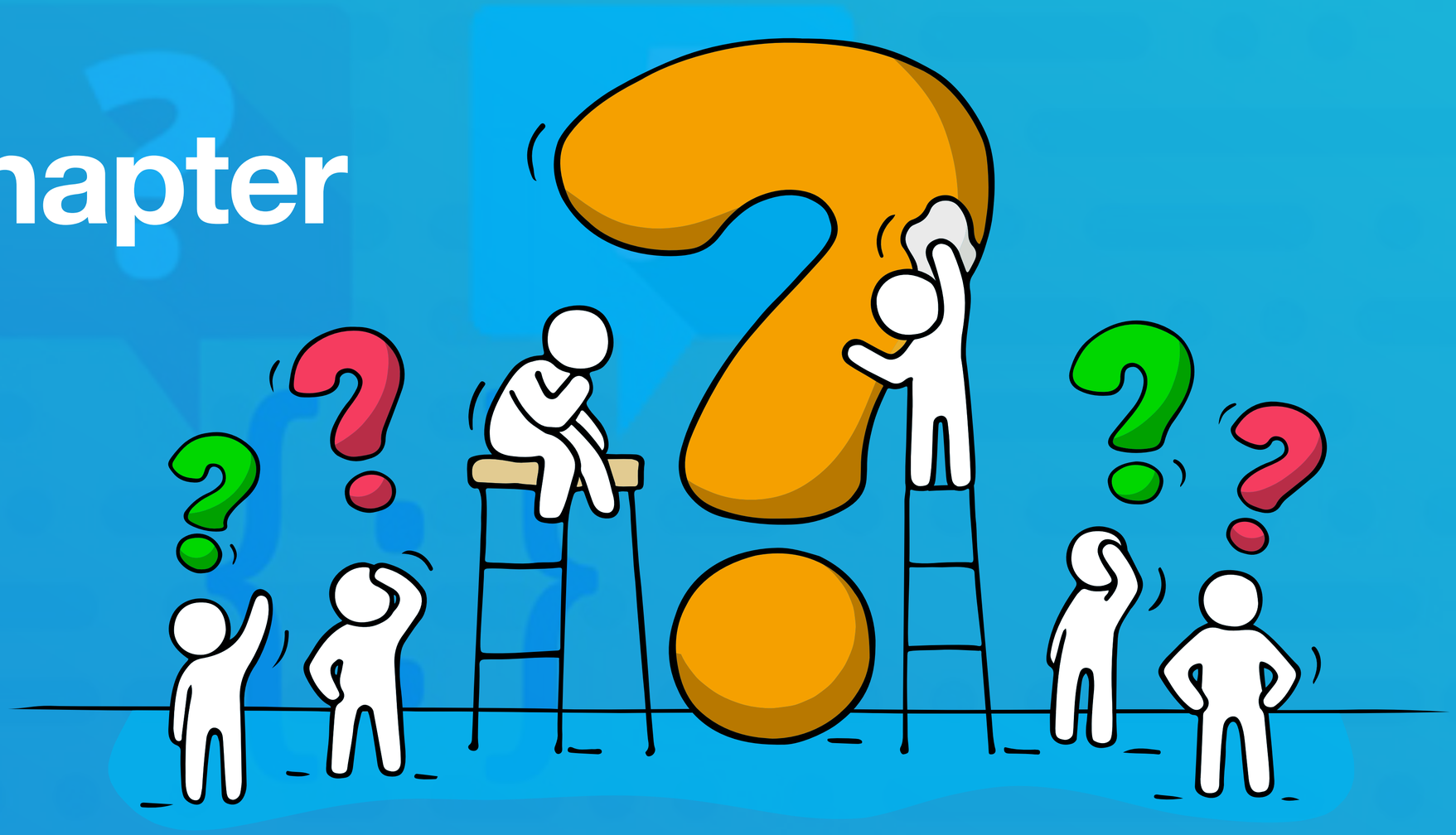


selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

End of Chapter

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr