WEBSERVICES

JAX-RS is an specification (just a definition) and Jersey is a JAX-RS implementation. Imagine that there is a benevolent organization that creates (and gives away for free) designs for cars. Manufacturers around the world recognize that those designs are good and implement real cars based on those designs. Well, JAX-RS is just like a plan or design on how to build RESTful APIs, and software manufacturers recognize it and implement those ideas with real RESTful APIs, one of this is called Jersey, but there are others (i.e. RESTEasy). Just as many other manufacturers besides Toyota may use the designs to create their own version of cars.

Jackson is a Json Processor. It helps you in converting your objects to json and vice versa.

Web services are services that are exposed on the internet for programmatic access. They are online APIs that you can call from your code. RESTful web services are a type of web services that are modern, light-weight, and use a lot of the concepts behind HTTP, the technology that drives the web. they are web services, so the exchange of data happens over the web. Over HTTP. A client sends an HTTP request, and the server returns back an HTTP response. Similar to web sites. But like we discussed, instead of the response being complete web pages, only the data is returned, because the client is just a program, and not a human. The client could then have it’s own logic to present the data to the users in a presentable format, but the exchange between the web service client and web service server is usually just bare-bones data. When a web service client makes a request to a web service endpoint, they are usually messages transmitted from one machine to another. These messages need to be in a format, a language, that both the client and the server can understand. This is protocol. We saw that since RESTful web services are web services, the request and response messages are almost always exchanged over HTTP. But in HTTP, there are different methods available. You would have heard of GET, POST PUT and so on. Messages can be exchanged in any (or all) HTTP methods. There are guidelines and best practices that tells you what methods need to be used when designing the service, depending on what the request is, but there is no rule as such.

Representational state transfer is actually an architecture style. REST make good use of the ideas and concepts behind HTTP. HTTP stands for Hyper Text Transfer Protocol. Like we’ve already seen, you can think of a protocol as a language or mechanism for communication. So, HTTP is a way to exchange and communicate information online.

Rest have resource based URI.  A GET method lets you get information from the server. And POST is used when you want to submit information to the server. They work well with resource based URIs that we just saw. HTTP also defines status codes and response headers which lets the server send back extra information or metadata that might be useful to the client. HTTP also defines status codes and response headers which lets the server send back extra information or metadata that might be useful to the client. in the case of RESTful web services, you cannot send readable messages because the client is a piece of code! This is why sending the right status code is very important. ow can the server even identify what kind of data is sent? Similarly, how does the client know what data format is returned by the server? The answer is again a header value called Content-Type

This was a broad overview of some of the important points about RESTful web services and how they’ve been influenced by HTTP. When you design a RESTful API:

1. You need to have resource-based URIs. Every resource or entity should be identifiable by a single URI.
2. You need to choose the right HTTP methods for different actions and operations for the API.
3. The response needs to return the right HTTP status codes
4. All requests and responses need to have the right Content-Type header set so that the format of the messages are well understood by everyone.

 One set of methods, including GET, PUT and DELETE, are safe for make repeated calls without worrying about the impact. They may not all be read-only. But they do not cause side-effects if called multiple times. And the other category, consisting of POST which you have to be very careful with, and make only as many calls as you need. The methods in the first set are called **idempotent** methods. GET, PUT and DELETE are idempotent. POST is **non-idempotent**. Take the example of a browser refresh button. Every browser has a refresh or reload button that does a very simple function: resend the last HTTP request that was made by the browser. If the last request happens to be an idempotent request like a GET, the browser just goes ahead and resends the request when you hit refresh. But if it was a POST, like after you've submitted a form, if you hit refresh, the browser warns you with a message that says something like "You've already submitted this data before. Are you sure you wish to resubmit?". This is simply the browser protecting you from making a duplicate non-idempotent request. So, it pays to generally use the right HTTP method for the right operation.

The HTTP protocol has a concept of request and response headers. Every HTTP request or response has a body, which is the message itself, and certain header values that contain metadata about the message. The header data could be stuff like the content length and date. One such possible header is Content-Type.

1XX Codes - Informational

The codes starting with 1XX are informational, like acknowledgement responses. We'll not be using this set of codes in this course.

2XX Codes - Success

The codes starting with 2XX are success codes. This indicates that the server received the request from the client and processed it successfully. Some examples:

3XX Codes - Redirection

## The server sends these codes to ask the client to do further action to complete the request. For example, it could be a redirect, asking the client to send the request s

## 4XX Codes - Client error

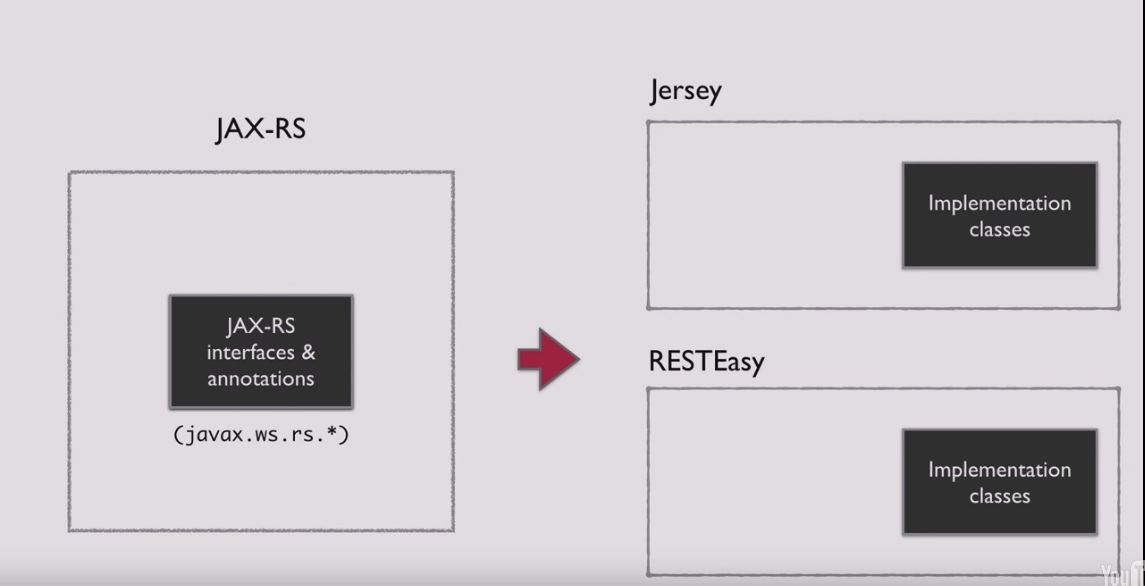
These error codes are returned if the client makes an error in the request. The request syntax could have been incorrect, or the client is requesting something that it's not supposed to see. somewhere else.

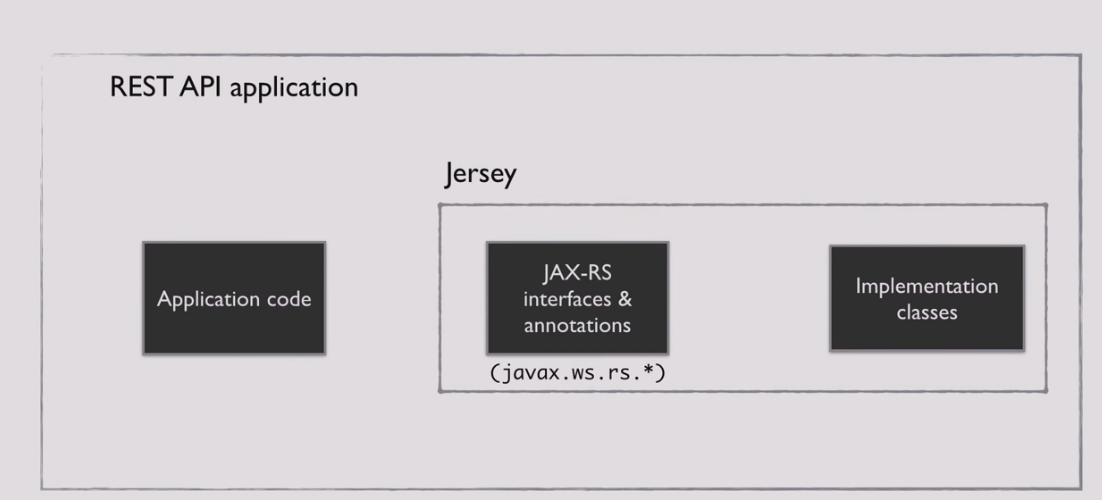
5XX Codes - Server error

The 4XX codes are when the client screws up when sending the request. The 5XX codes are when the server screws up when sending the response. It's basically the server saying, Ok, I got your request, and it looked like a valid one, but something went wrong when I tried to process it.

HATOS stands for Hypermedia as the Engine of Application State. web service is being super-helpful to the client by providing all these links in the response. Similar to hyperlinks in web sites. Whether the client wants to use it or not doesn't matter. But if they want it, it's there. And just like that, you've eliminated the need for documentation for all these APIs. The client developer just picks up the value of the right URIs from a previous response and makes subsequent calls to those URIs.

If you do this, you don't let the client programmer have to know and hard-code the URIs in order to interact with the resources and the application state. You basically let the hypertext you send in the response drive the client's interaction with the application state. So, you could say that hypertext, or hypermedia as it is sometimes called, is being the driver or engine of application state. Hypermedia as the Engine of Application State. HATEOAS.





Any resource request which comes in is handled by Jersey.

**JAXB** stands for Java architecture for XML binding. It is used to convert XML to java object and java object to XML.**JAXB** defines an API for reading and writing Java objects to and from XML documents.