WEB TECHNOLOGIES USING JAVA

⮡ COURSE 5 – REST WEBSERVICES.

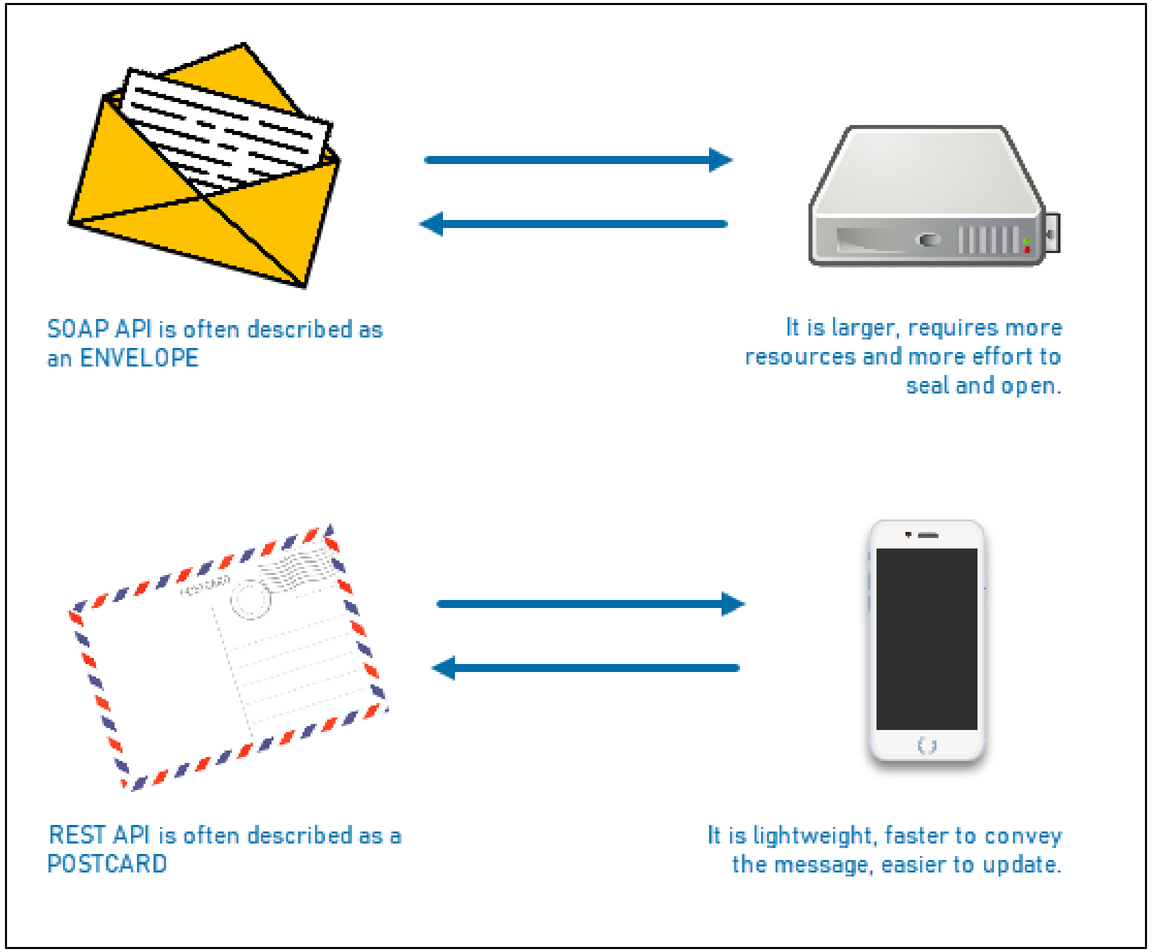
AGENDA

- WEBSERVICES
- REST WEBSERVICES
- REST APIS
- RESOURCES
- REPRESENTATIONS
- HTTP METHODS AND RESPONSE CODES
- REST API DESIGN

WEBSERVICES

- Web service: a web server programmed with specific, often reusable, logic
- Types of web services:
  - REST (Representational State Transfer)
  - SOAP (Simple Object Access Protocol)

WEBSERVICES



SOAP API is often described as an ENVELOPE

It is larger, requires more resources and more effort to seal and open.

REST API is often described as a POSTCARD

It is lightweight, faster to convey the message, easier to update.

WEBSERVICES

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API

REST WEBSERVICES

- REST - Representational State Transfer:
  - the architecture of transferring the state of resources, for designing distributed systems
  - a set of rules/standards/guidelines for how to build a web API
  - when application A communicates with application B, application A supplies a representation of its relevant state with each request to application B

REST WEBSERVICES

- REST design principles:
  - **Client-Server**: concerns should be separated between clients and servers. This enables client and server components to evolve independently and in turn allows the system to scale.
  - **Stateless**: the server shouldn’t remember the state of the client. Instead, clients must include all the necessary information in the request.
  - **Layered System**: multiple layers such as gateways, firewalls, and proxies can exist between client and server. Layers can be added, modified, reordered, or removed transparently to improve scalability.
  - **Cache**: the client can cache responses and reuse them for later requests. This reduces the load on the server and helps improve the performance.
  - **Uniform Interface**: interfaces uniformity. The components can evolve independently if they implement the agreed-on contract.

REST APIS

- a Web API conforming to the REST architectural style is a REST API
- having a REST API makes a web service “RESTful”
- REST doesn’t dictate the actual technology to be used for developing APIs
- Uniform Interface is achieved through these abstractions:
  - resources
  - representations
  - URIs
  - HTTP methods

RESOURCES

- a resource is anything that can be accessed or manipulated.
- resources must have an identifier – the URI (Uniform Resource Identifier)
- the URI represents the actual location of a resource on the Web
- URI templates provide a standardized mechanism for describing URI structure (http://blog.example.com/{year}/posts, where year is a variable)

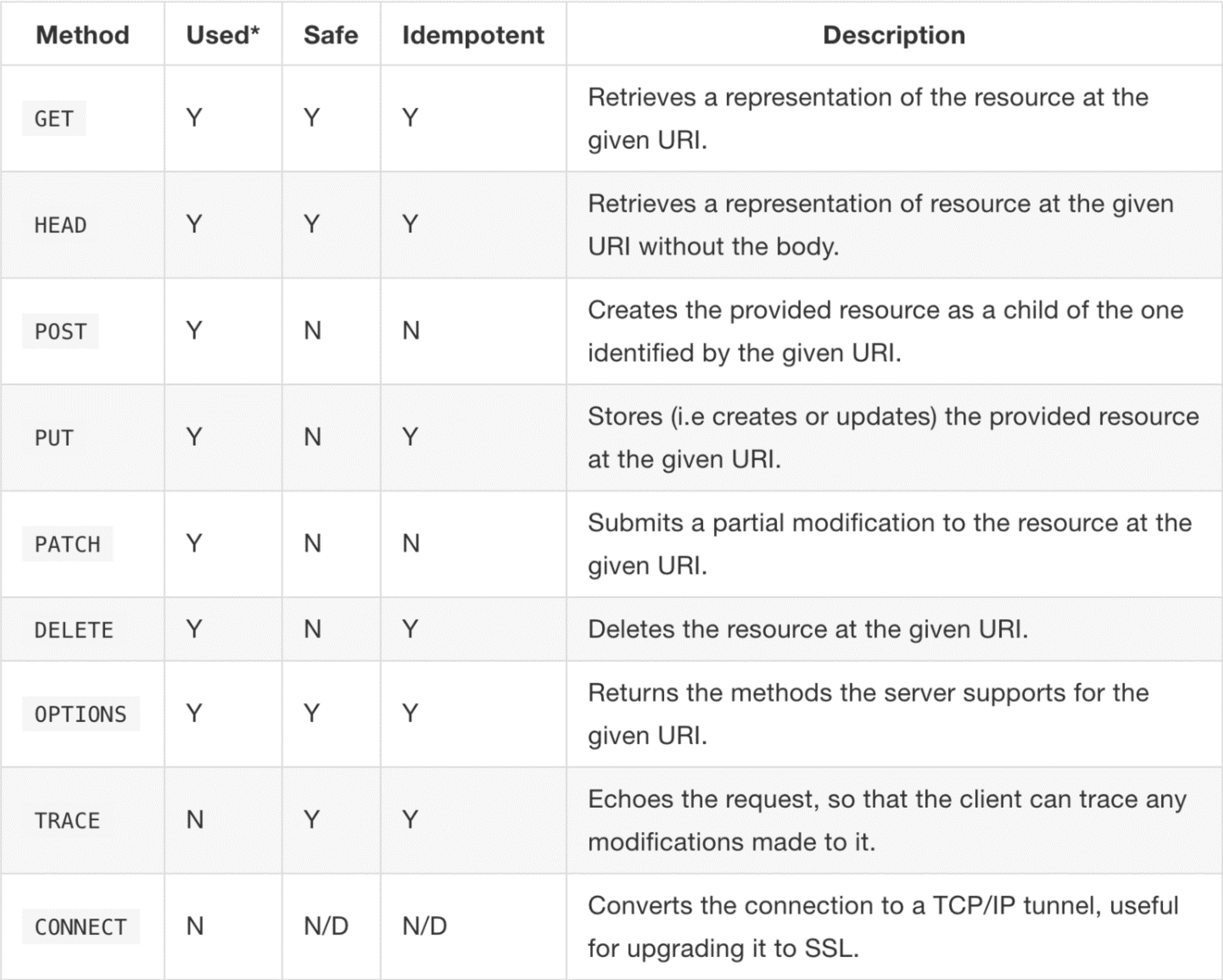
REPRESENTATIONS

- Serialized data of a resource is a representation. It can be viewed as a snapshot of a resource’s state at a given point in time
- REST components interact with a resource by transferring its representations back and forth. They never directly interact with the resource
- one resource can have several representations, such as HTML, XML, JSON, PDFs, JPEGs, MP4s etc
- **JSON** is the de facto standard for REST services

HTTP REQUEST METHODS

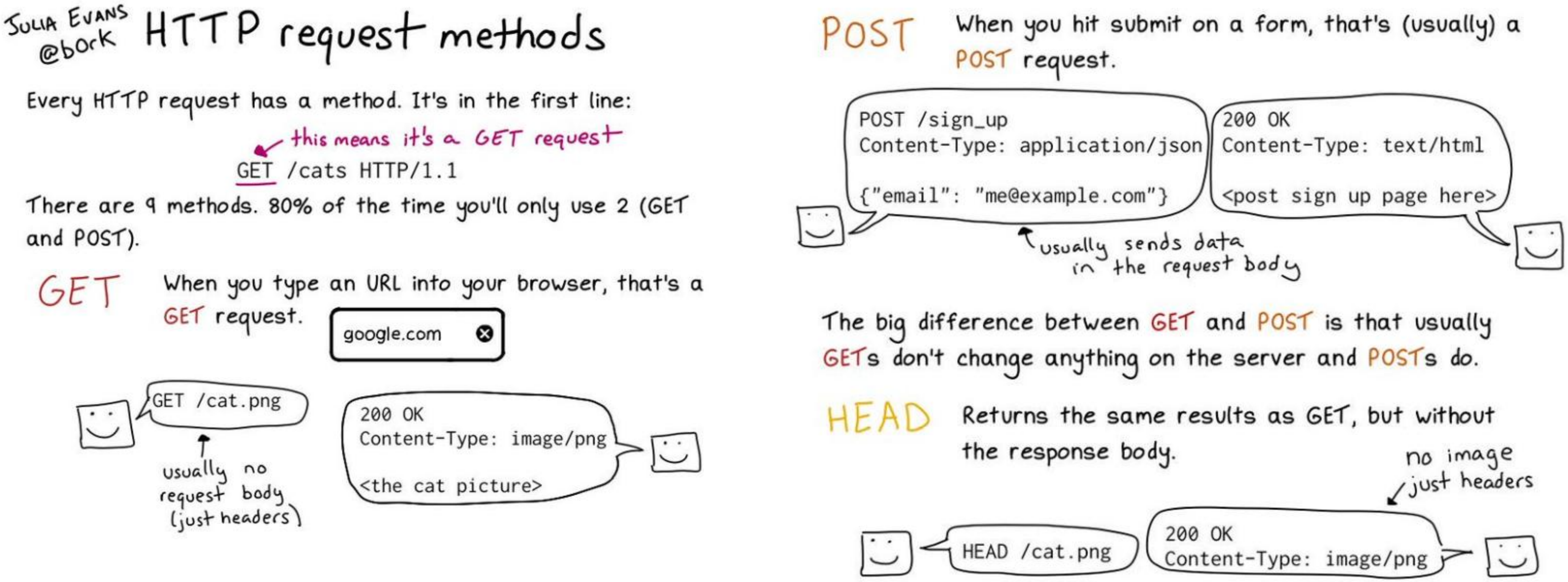
- **CRUD**: four basic persistence functions - Create, Read, Update, and Delete
- combination of an URI and a HTTP method generates a CRUD operation for a resource

HTTP REQUEST METHODS



Method	Used*	Safe	Idempotent	Description
`GET`	Y	Y	Y	Retrieves a representation of the resource at the given URI.
`HEAD`	Y	Y	Y	Retrieves a representation of resource at the given URI without the body.
`POST`	Y	N	N	Creates the provided resource as a child of the one identified by the given URI.
`PUT`	Y	N	Y	Stores (i.e creates or updates) the provided resource at the given URI.
`PATCH`	Y	N	N	Submits a partial modification to the resource at the given URI.
`DELETE`	Y	N	Y	Deletes the resource at the given URI.
`OPTIONS`	Y	Y	Y	Returns the methods the server supports for the given URI.
`TRACE`	N	Y	Y	Echoes the request, so that the client can trace any modifications made to it.
`CONNECT`	N	N/D	N/D	Converts the connection to a TCP/IP tunnel, useful for upgrading it to SSL.

HTTP REQUEST METHODS



JULIA EVANS
@bork

HTTP request methods

Every HTTP request has a method. It's in the first line:

this means it's a GET request

```
GET /cats HTTP/1.1
```

There are 9 methods. 80% of the time you'll only use 2 (GET and POST).

GET When you type an URL into your browser, that's a GET request.

google.com ⊗

```
GET /cat.png
```

usually no request body (just headers)

```
200 OK
Content-Type: image/png

<the cat picture>
```

POST When you hit submit on a form, that's (usually) a POST request.

```
POST /sign_up
Content-Type: application/json

{"email": "me@example.com"}
```

usually sends data in the request body

```
200 OK
Content-Type: text/html

<post sign up page here>
```

The big difference between GET and POST is that usually GETs don't change anything on the server and POSTs do.

HEAD Returns the same results as GET, but without the response body.

```
HEAD /cat.png
```

no image just headers

```
200 OK
Content-Type: image/png
```

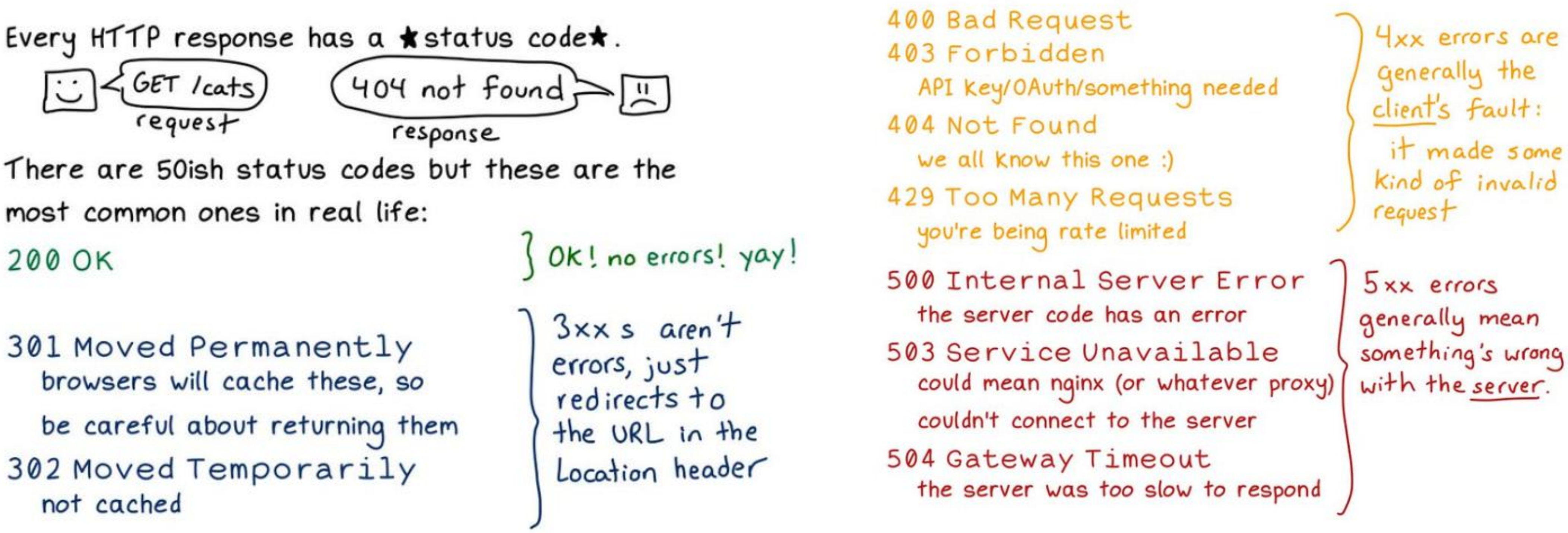
HTTP RESPONSE STATUS CODES

- the status codes are grouped into the following categories:
  - **Informational Codes**: the server has received the request but hasn’t completed processing it – 100 series.
  - **Success Codes**: the request has been successfully received and processed – 200 series.
  - **Redirection Codes**: the request has been processed, but the client must perform an additional action to complete the request – 300 series.
  - **Client Error Codes**: there was an error or a problem with client’s request – 400 series.
  - **Server Error Codes**: there was an error on the server while processing the client’s request – 500 series.

The worst code to see

Source: Http Toolkit

HTTP RESPONSE STATUS CODES



Every HTTP response has a ★status code★.

GET /cats
request
404 not found
response

There are 50ish status codes but these are the most common ones in real life:

200 OK

} OK! no errors! yay!

301 Moved Permanently
browsers will cache these, so be careful about returning them

302 Moved Temporarily
not cached

} 3xx s aren't errors, just redirects to the URL in the Location header

400 Bad Request

403 Forbidden
API key/OAuth/something needed

404 Not Found
we all know this one :)

429 Too Many Requests
you're being rate limited

} 4xx errors are generally the client's fault: it made some kind of invalid request

500 Internal Server Error
the server code has an error

503 Service Unavailable
could mean nginx (or whatever proxy) couldn't connect to the server

504 Gateway Timeout
the server was too slow to respond

} 5xx errors generally mean something's wrong with the server.

REST API DESIGN

- the end users should consume the API easily
- high level steps:
  - identify resources
  - identify endpoints: design URIs that map resources to endpoints
  - identify actions: identify the right HTTP methods
  - identify responses: identify the supported resource representation for the request and response along with the right status codes to be returned

REST API DESIGN

- Best practices to model URIs in REST
  - URI should be simple, intuitive, easy to read, and consistent.
  - avoid having your API start the root domain (http://example.com/api instead of http://api.example.com)
  - resource endpoints should be a plural and not a singular name (http://example.com/api/products)
  - access a specific resource by its identifier in HTTP GET call (http://example.com/api/products/prod123)
  - do not use privileged user information as unencrypted parameters (http://example.com/api/user?password=039456337-87)
  - allow multiple results representations:
    - http://example.com/api/products?format=json
    - http://example.com/api/products?format=pdf

BIBLIOGRAPHY

- Spring in Action, by Craig Walls
- Spring REST, by Balaji Varanasi, Sudha Belida
- REST API Design Rulebook, by Mark Masse
- https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

THANK YOU

DANIELA SPILCĂ