# Spring REST

"Code with Passion!"

# Topics

- CRUD operations via REST calls
- REST client tools
- @ResponseBody
- Representations (Formats) - produces and consumes
- XML binding
- Create/Update/Delete
- @RestController

# CRUD Operations via REST calls

# CRUD Operations are Performed through HTTP method + URI

| CRUD Operations | 4 main HTTP methods | |
|---|---|---|
| | Verb | Noun |
| **Create (Single)** | **POST** | **Collection URI** |
| **Read (Multiple)** | **GET** | **Collection URI** |
| **Read (Single)** | **GET** | **Entry URI** |
| **Update (Single)** | **PUT** | **Entry URI** |
| **Delete (Single)** | **DELETE** | **Entry URI** |

# HTTP Methods:

- /orders
  - GET - list all orders
  - POST - submit a new order

  /orders/{order-id}
  - GET - get an order representation
  - PUT - update an order
  - DELETE - cancel an order

  /orders/average-sale
  - GET - calculate average sale
- /customers
  - GET - list all customers
  - POST - create a new customer

  /customers/{cust-id}
  - GET - get a customer representation
  - DELETE- remove a customer

  /customers/{cust-id}/orders
  - GET - get all orders of a customer

http://www.infoq.com/articles/rest-introduction

# REST Tools

# REST Client Tools

- curl
  - Command line tool for sending/receiving all REST requests/responses (GET, POST, PUT, DELETE, HEAD)
- Postman
  - Chrome plug-in for sending/receiving all REST requests/responses (GET, POST, PUT, DELETE, HEAD)
- RESTClient
  - GUI tool (Java-based) for sending/receiving all REST requests/responses (GET, POST, PUT, DELETE, HEAD)
- Browser
  - Useful for sending/receiving GET requests/responses

# Lab:

Exercise #1: Install REST client tools
4949_spring4_mvc_rest.zip

# @ResponseBody

# @ResponseBody

- Annotation that indicates a method return value should be bound to the web response body (rather than returning a logical view)

```
// Retrieve all in JSON
@RequestMapping(method = RequestMethod.GET,
                produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public List<Person> getAll() {
    return peopleList;
}

// Retrieve one item in JSON
@RequestMapping(value = "{id}", method = RequestMethod.GET,
                produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public Person getPerson(@PathVariable("id") int id) {
    Person person = peopleList.get(id-1);
    return person;
}
```

# Formats: Produces & Consumes

# Formats in HTTP

## Request

```
GET /music/artists/beatles/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

Format

## Response

State transfer

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

Representation

# Multiple Formats

- Client and server can send and receive data in multiple formats
- Type of formats
  - text/html – regular web page
  - Application/json – in JSON
  - application/xhtml+xml – in XML
  - application/rss+xml – as a RSS feed
  - application/octet-stream – an octet stream
  - application/rdf+xml – RDF format

# HTTP Headers for Formats

- How does a client specify acceptable response formats?
  - Through 'Accept' HTTP request header
- How does a client specify its request format?
  - Through 'Content-Type' HTTP request header
- How does a server specify its response format?
  - Through 'Content-Type' HTTP response header

# Produces

- Used to specify the MIME media types of representations (formats) a resource can produce and send back to the client

```
@GetMapping(value="/resources/{id}",
                    produces={MediaType.APPLICATION_XML_VALUE,
                                        MediaType.APPLICATION_JSON_VALUE})
public Account getAccount(@PathVariable int id) {
        Account account = this.accounts.get(id);
         if (account == null) {
              throw new ResourceNotFoundException(id);
         }

         return account;
}
```

# Consumes

- Used to specify the MIME media types of representations (formats) a resource can consume

```
// Can consume either XML or JSON in the request
@RequestMapping(value = "{id}",
                        method = RequestMethod.GET,
                        consumes={MediaType.APPLICATION_XML_VALUE,
                                  MediaType.APPLICATION_JSON_VALUE})
public Account getAccount(@PathVariable int id) {
       Account account = this.accounts.get(id);
        if (account == null) {
            throw new ResourceNotFoundException(id);
        }

        return account;
}
```

# Create/Update/Delete

# Create a new Item

- Controller method that handles HTTP POST create request gets selected
- HTTP request POST body contains new item to be created
  - Could be in either JSON or XML based on values of “consumes”
- You want to return newly created item along with HTTP 201

```
// Create a new Person item
@RequestMapping(method = RequestMethod.POST,
                        consumes = {
                              MediaType.APPLICATION_JSON_VALUE,
                              MediaType.APPLICATION_XML_VALUE })
@ResponseBody
@ResponseStatus(HttpStatus.CREATED)
public Person create(@RequestBody @Valid Person person) {
   peopleList.add(person);
   return person;
}
```

# Update an existing Item

- Controller method that handles HTTP PUT request selected
- Request PUT body contains update item
  - Could be in either JSON or XML based on values of “consumes”

```
// Update one
@RequestMapping(value = "{id}", method = RequestMethod.PUT,
                        consumes = {
                              MediaType.APPLICATION_JSON_VALUE,
                              MediaType.APPLICATION_XML_VALUE })
@ResponseBody
public Person update(@PathVariable("id") int id,
                          @RequestBody @Valid Person newPerson) {
        Person person = peopleList.get(id - 1);
        if (person == null) {
            throw new ResourceNotFoundException(id);
        }
        person.setFirstName(newPerson.getFirstName());
        person.setLastName(newPerson.getLastName());
        peopleList.add(person);
        return person;
}
```

# Delete an Item

- Controller method that handles HTTP DELETE request selected

```
// Delete one
@RequestMapping(value = "{id}", method = RequestMethod.DELETE)
@ResponseBody
public List<Person> delete(@PathVariable("id") int id) {
    peopleList.remove(id-1);
    return peopleList;
}
```

# Lab:

Exercise 3: Create/Update/Delete
4949_spring4_mvc_rest.zip

# @RestController

# @RestController: definition

- Introduced in Spring 4.0
- Convenience annotation of combining @Controller and @ResponseBody annotations

```
// Definition of @RestController annotation
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
}
```

# @RestController: usage

```
@RestController
// @Controller
public class AccountController {
   private List<Account> accounts = new ArrayList<Account>();

   @RequestMapping(value = "{id}", method = RequestMethod.GET)
   // @ResponseBody
   public Account getAccount(@PathVariable int id) {
       Account account = this.accounts.get(id);
       // ...
       return account;
    }
```

# Lab:

Exercise 4: @RestController
Exercise 5: Spring Data
4949_spring4_mvc_rest.zip

# REST client using RestTemplate

# Why RestTemplate?

- Invoking RESTful services in Java is typically done using a helper class such as Jakarta Commons *HttpClient*. For common REST operations this approach is too low level as shown below

```
String uri = "http://example.com/hotels/1/bookings";

PostMethod post = new PostMethod(uri);
String request = // create booking request content
post.setRequestEntity(new StringRequestEntity(request));

httpClient.executeMethod(post);

if (HttpStatus.SC_CREATED == post.getStatusCode()) {
  Header location = post.getRequestHeader("Location");
  if (location != null) {
    System.out.println("Created new booking at :" + location.getValue());
  }
}
```

# RestTemplate

- *RestTemplate* provides higher level methods that correspond to each of the six main HTTP methods that make invoking many RESTful services a one-liner and enforce REST best practices
- RestTemplate methods
  - getForObject(), getForEntity() for HTTP GET
  - postForLocation (String url, ...) and postForObject (String url, ...) for HTTP POST
  - delete() for HTTP DELETE
  - put(String url, ...) for HTTP PUT
  - headForHeaders(String url, ..) for HTTP HEAD
  - optionsForAllow(String url, ..) for HTTP OPTIONS

# Method Naming Convention

- *getForObject()* will perform a GET, convert the HTTP response into an object type of your choice and return that object
- *postForLocation()* will do a POST, converting the given object into a HTTP request and return the response HTTP Location header where the newly created object can be found

# URI Template Arguments

- Each method takes URI template arguments in two forms
  - String variable length argument

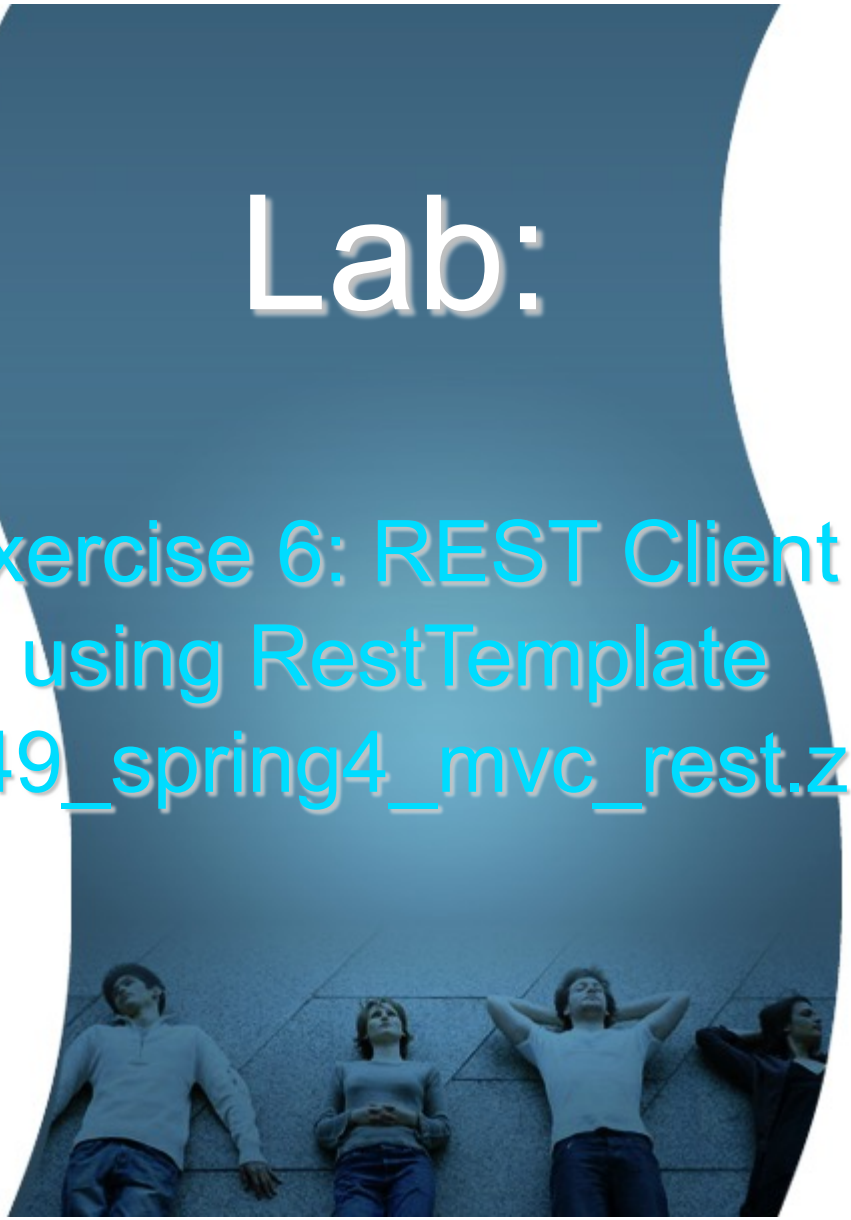
```
String result = restTemplate.getForObject(
        "http://example.com/hotels/{hotel}/bookings/{booking}",
        String.class,"42", "21");
```

  - Map<String, String>

```
Map<String, String> variables = new HashMap<String, String>(2);
variables.put("hotel", "42");
variables.put("booking", "21");
String result = restTemplate.getForObject(
        "http://example.com/hotels/{hotel}/rooms/{booking}",
         String.class,
         variables);
```

# Lab:

## Exercise 6: REST Client using RestTemplate
## 4949_spring4_mvc_rest.zip



# Code with Passion!