

RESTful Web Services with Spring

Explained Using Airport Symbols



Arjen Poutsma
SpringSource

About me

- Fifteen years of experience in Enterprise Software Development
- Six years of Web service experience
- Development lead of Spring Web Services
- Now working on Spring 3.0
- Contributor to various Open Source frameworks: (XFire, Axis2, NEO, ...)

Agenda

- What is REST?
- RESTful architecture & design
- REST in Java
- REST in Spring



What is REST?

What is REST?

- Representational State Transfer
- Architectural style
- Architectural basis for HTTP

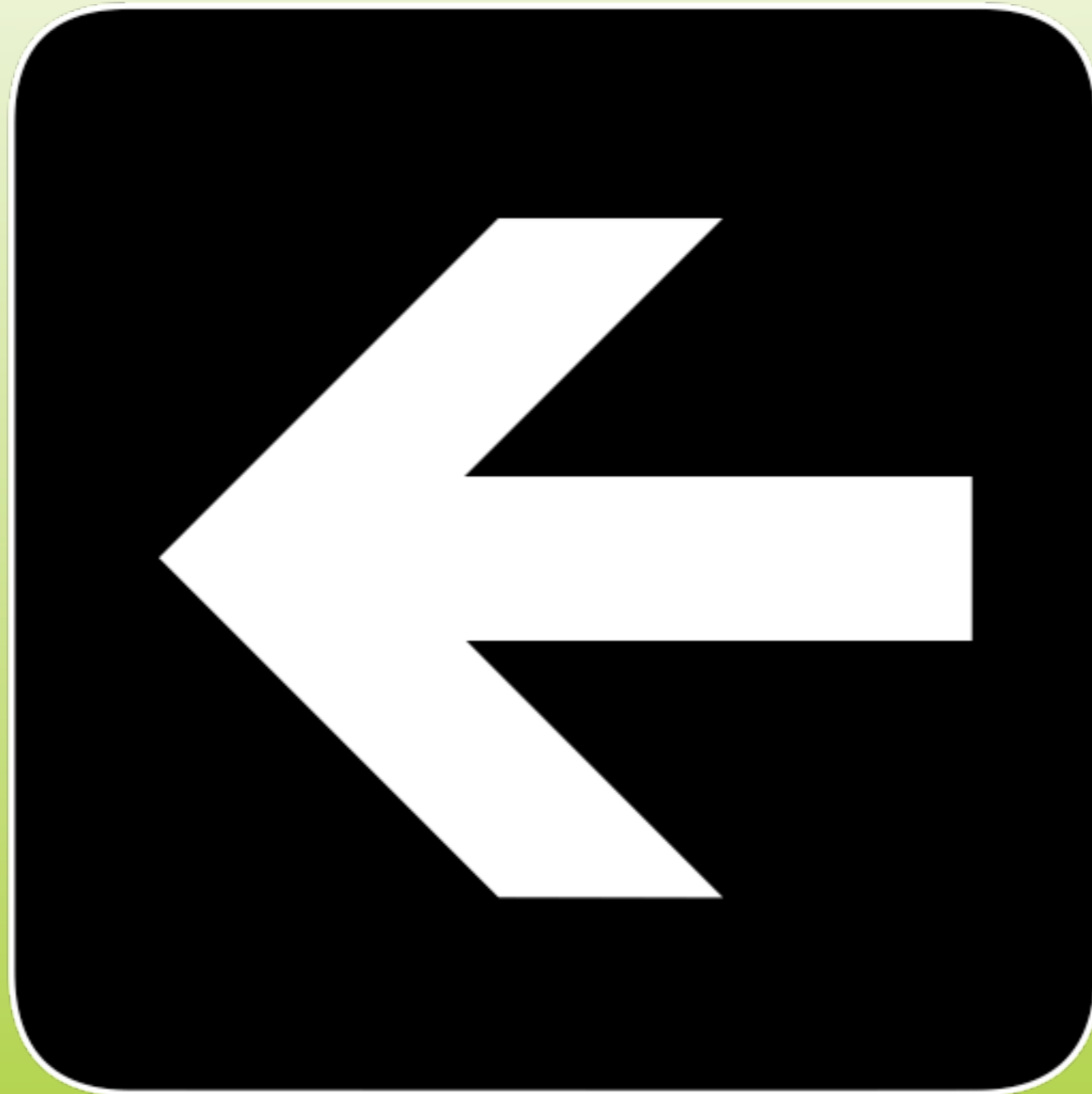


Resources

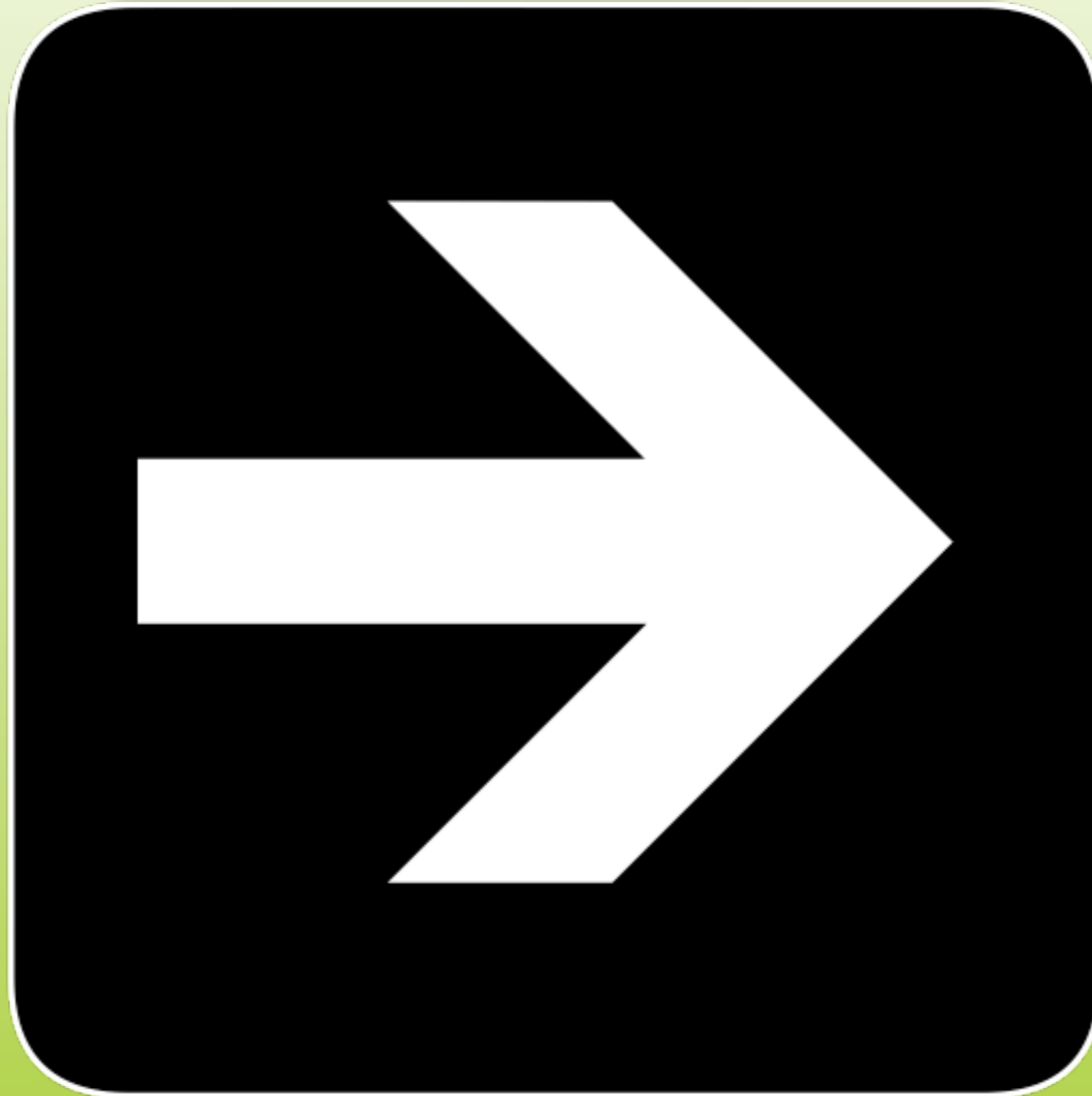
- Typically nouns
 - Customer
 - Orders
 - Shopping cart
- Expressed by URIs



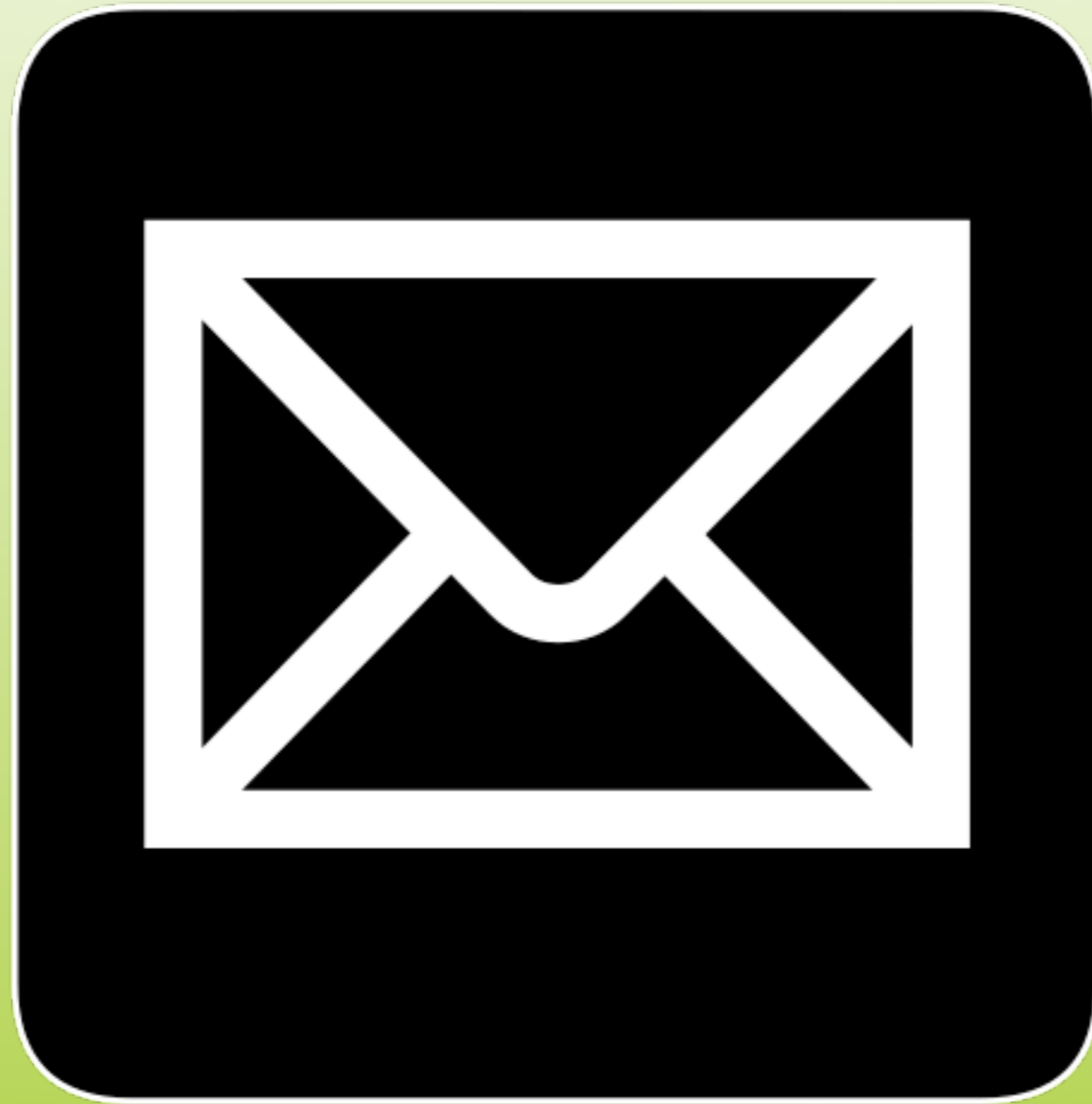
Uniform Interface



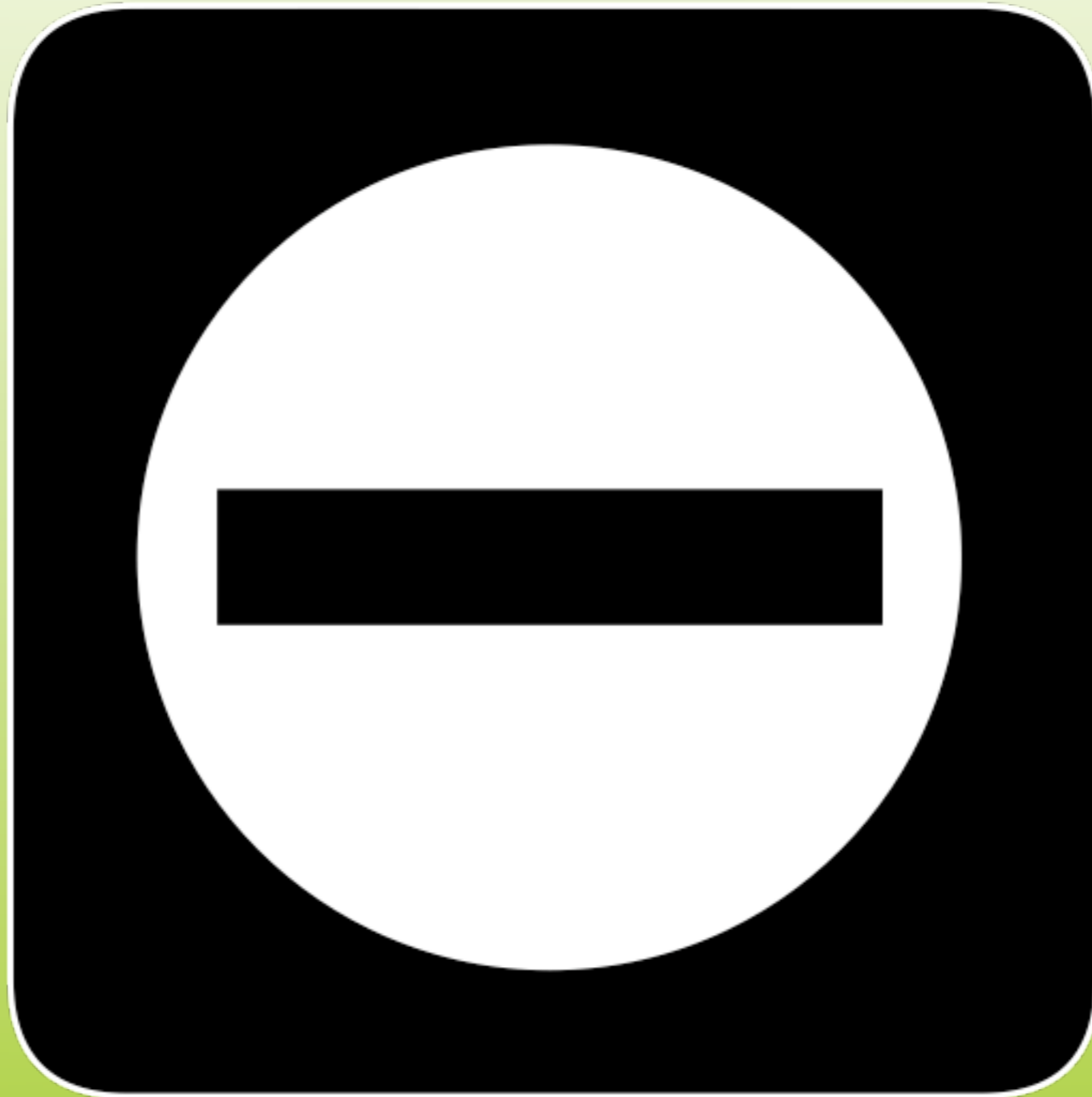
Uniform Interface



Uniform Interface

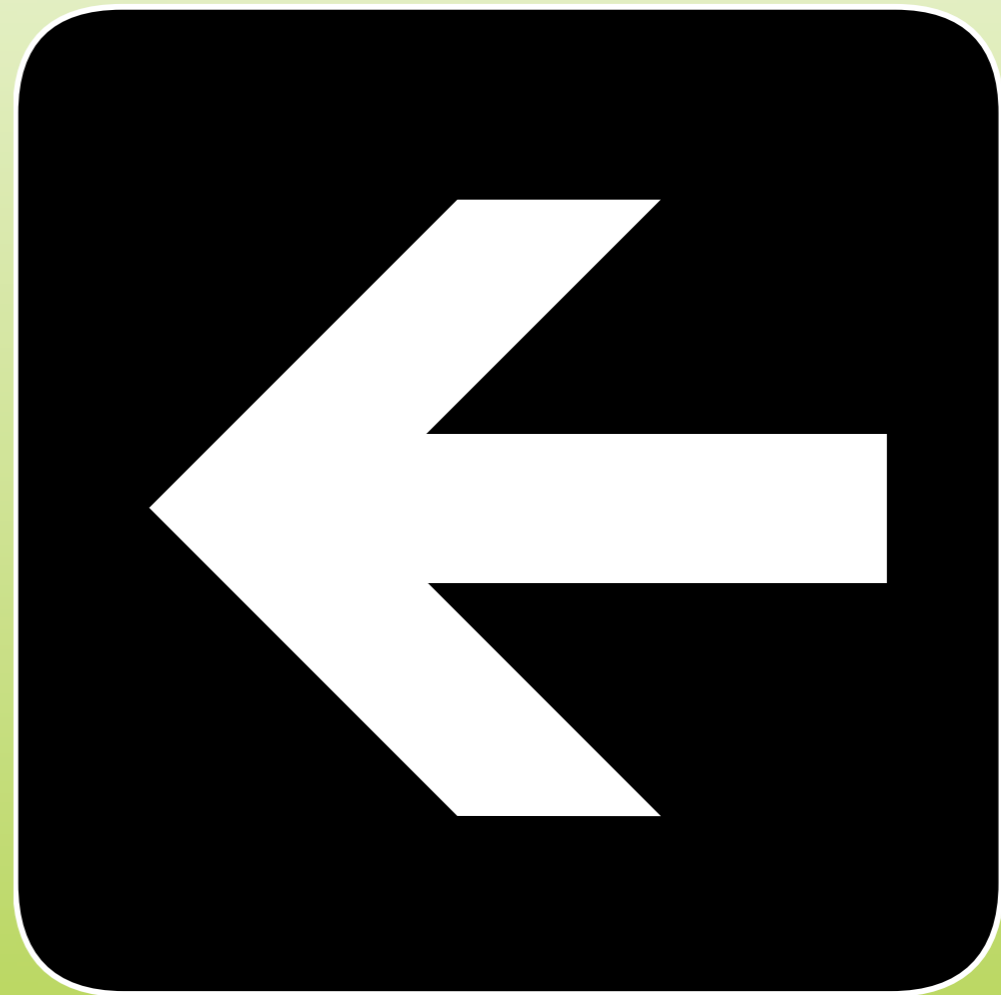


Uniform Interface



GET

- Retrieves Representation of Resource
- Safe operation



GET is Cacheable

- Servers returns ETag header
- Send on subsequent retrieval
- If not changed, 304 (Not Modified) is returned



GET Example

```
GET /hotels  
Host: example.com  
...
```



GET Example

```
GET /hotels
Host: example.com
...
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: 1456
Content-Type:
  application/xml

<hotels>
...
</hotels>
```

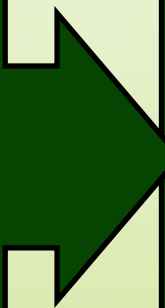
Conditional GET

```
GET /hotels  
Host: example.com  
...
```



Conditional GET

```
GET /hotels  
Host: example.com  
...
```



```
HTTP/1.1 200 OK  
Date: ...  
ETag: "b4bdb3"  
Content-Length: 1456  
...
```


Conditional GET

```
GET /hotels  
Host: example.com  
...
```

```
HTTP/1.1 200 OK  
Date: ...  
ETag: "b4bdb3"  
Content-Length: 1456  
...
```

```
GET /hotels  
If-None-Match: "b4bdb3"  
Host: example.com  
...
```

Conditional GET

```
GET /hotels  
Host: example.com  
...
```

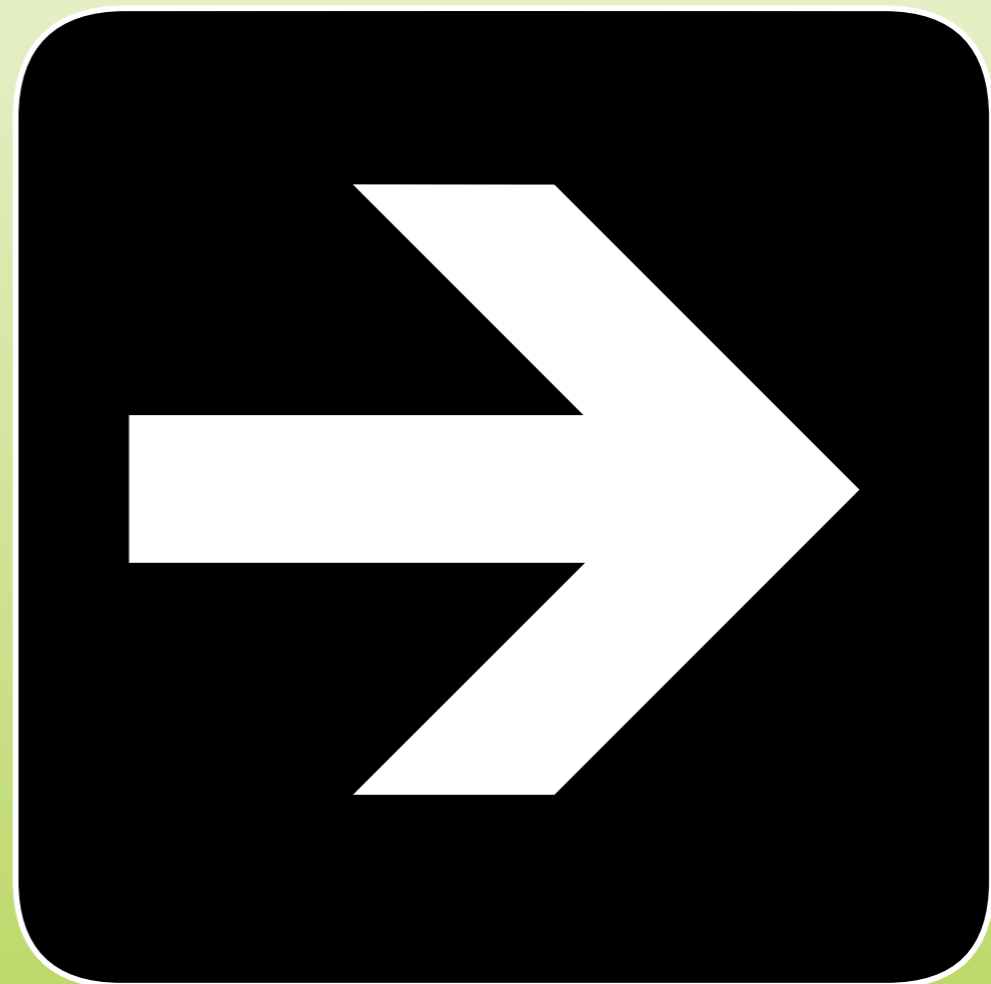
```
HTTP/1.1 200 OK  
Date: ...  
ETag: "b4bdb3"  
Content-Length: 1456  
...
```

```
GET /hotels  
If-None-Match: "b4bdb3"  
Host: example.com  
...
```

```
HTTP/1.1 304 Not  
Modified  
Date: ...  
ETag: "b4bdb3"  
Content-Length: 0
```

PUT

- Updates resource
- Creates resource, when the destination URI is known
- Idempotent



PUT Example

```
PUT /hotels/2  
Host: example.com
```

```
<hotel>  
...  
</hotel>
```



PUT Example

```
PUT /hotels/2  
Host: example.com
```

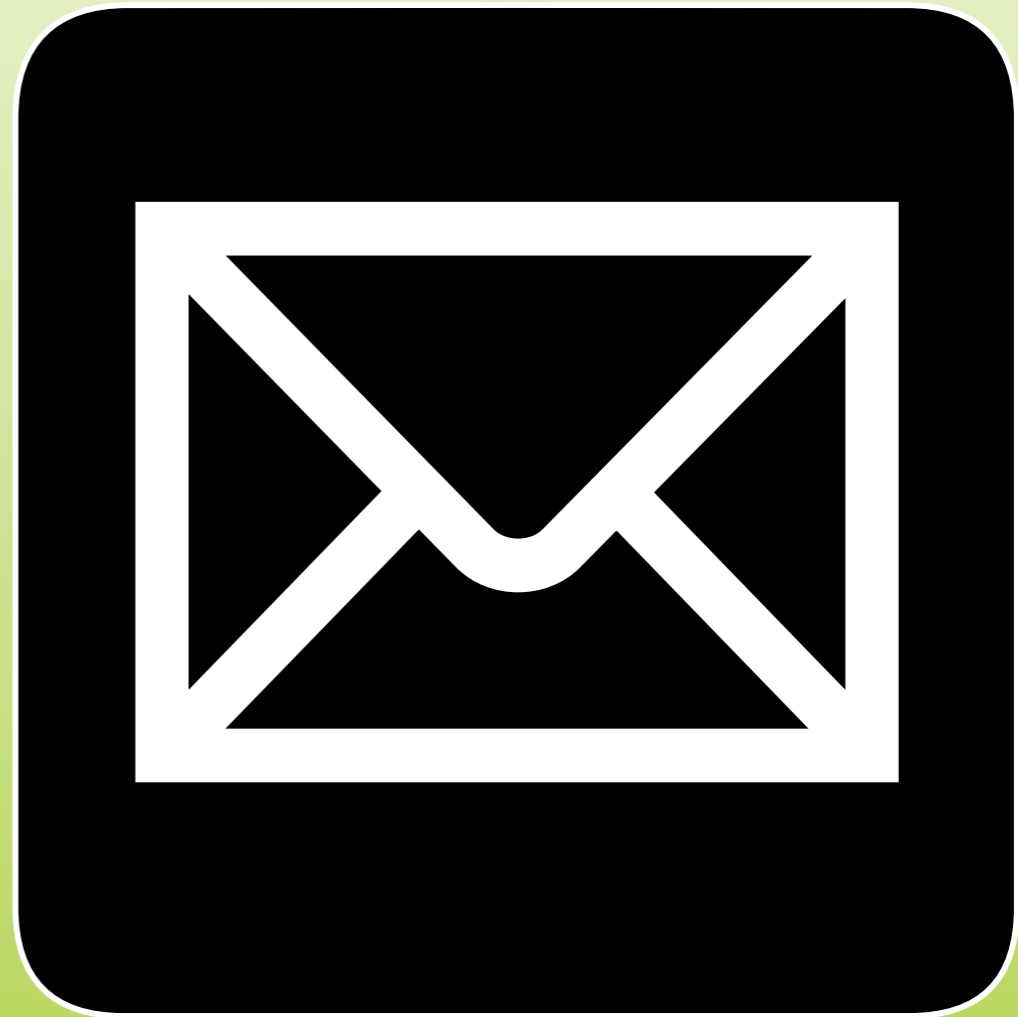
```
<hotel>  
...  
</hotel>
```

```
HTTP/1.1 201 Created  
Date: ...  
Content-Length: 0
```

```
...
```

POST

- Creates new Resource
- Child of other Resource
- Response Location header is used to indicate URI of child



POST Example

```
POST /hotels/1/  
  bookings  
Host: example.com
```

```
<booking>  
...  
</booking>
```



POST Example

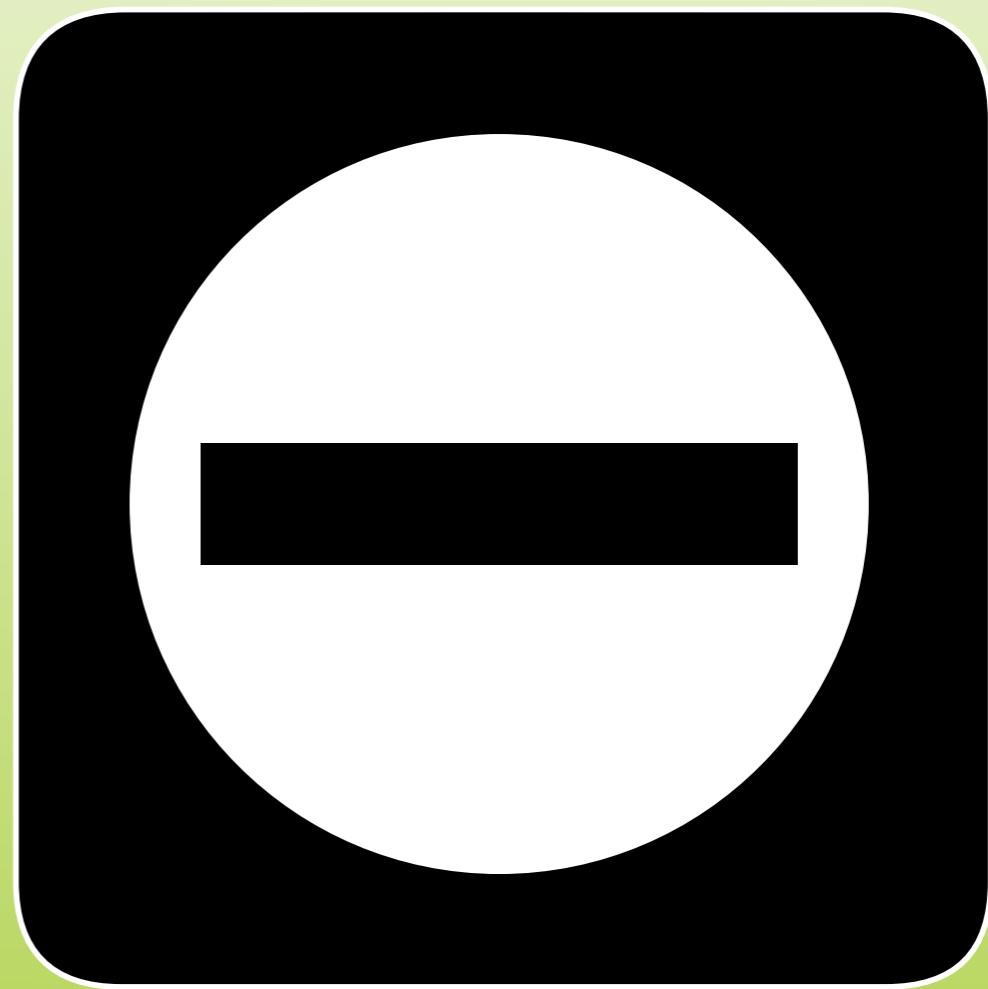
```
POST /hotels/1/  
  bookings  
Host: example.com
```

```
<booking>  
...  
</booking>
```

```
HTTP/1.1 201 Created  
Date: ...  
Content-Length: 0  
Location:  
http://example.com/  
hotels/1/bookings/50  
...
```



DELETE

- Deletes a resource
- Idempotent



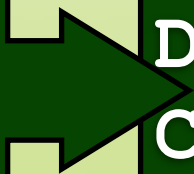
DELETE Example

```
DELETE /hotels/3  
Host: example.com  
...
```



DELETE Example

```
DELETE /hotels/3  
Host: example.com  
...
```



```
HTTP/1.1 204 No Content  
Date: ...  
Content-Length: 0  
...
```

Representations

- Access resource through representations
- More than one representation possible
- Desired representation in Accept header
 - Or file extension
- Delivered representation show in Content-Type



Stateless conversation

- Server does not maintain state
 - No HTTP Session!
- Client maintains state through links
- Very scalable
- Loose coupling



Hypermedia

- Resources contain links
- Client state transitions are made through these links
- Links are provided by server
- XLink
- Seamless evolution



RESTful architecture

RESTful Architecture

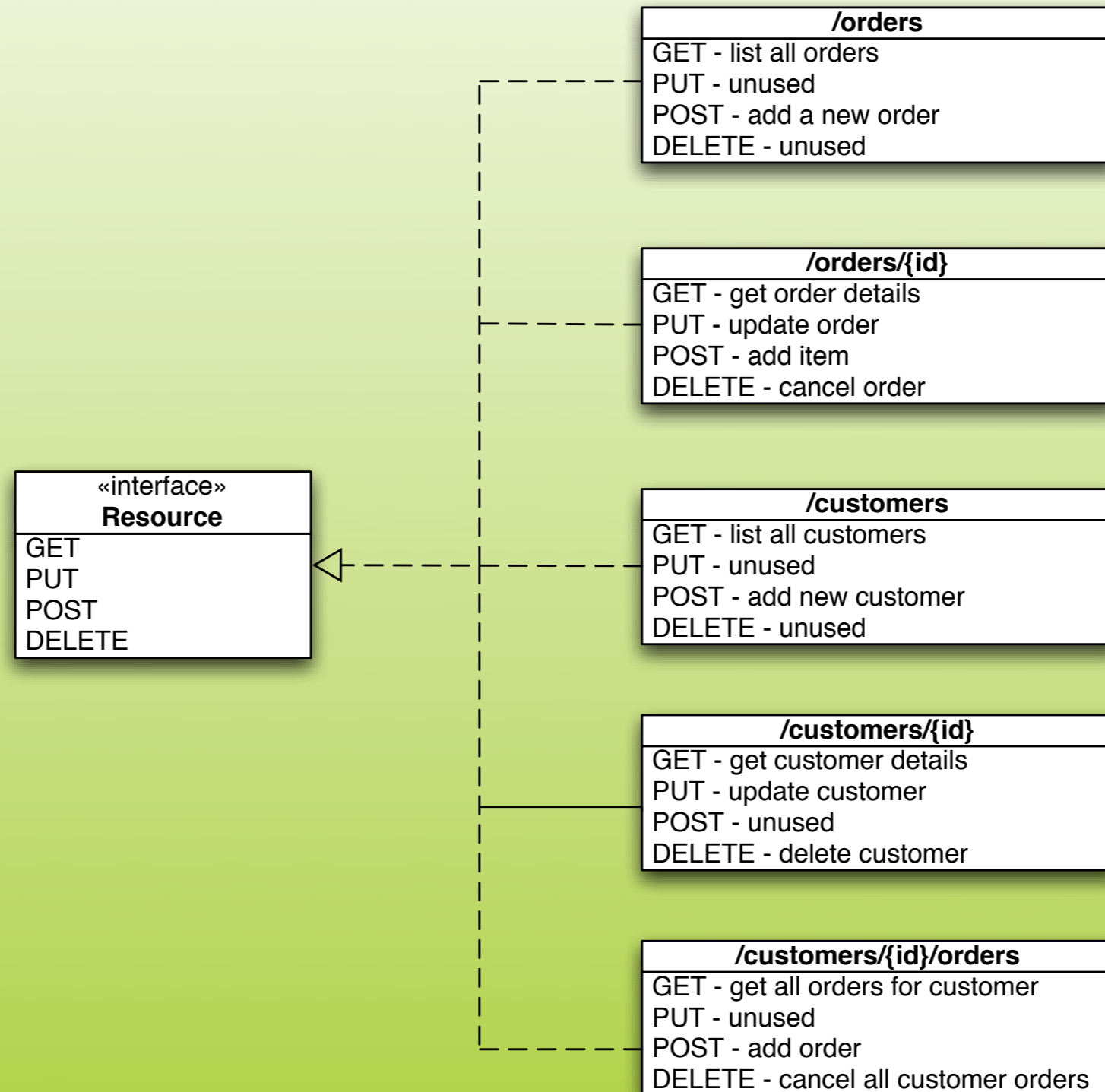
OrderManagementService

- + getOrders()
- + submitOrders()
- + getOrderDetails()
- + getOrdersForCustomers()
- + updateOrder()
- + addOrderItem()
- + cancelOrder()

CustomerManagementService

- + getCustomers()
- + addCustomer()
- + getCustomerDetails()
- + updateCustomer()
- + deleteCustomer()

RESTful Architecture



RESTful application design

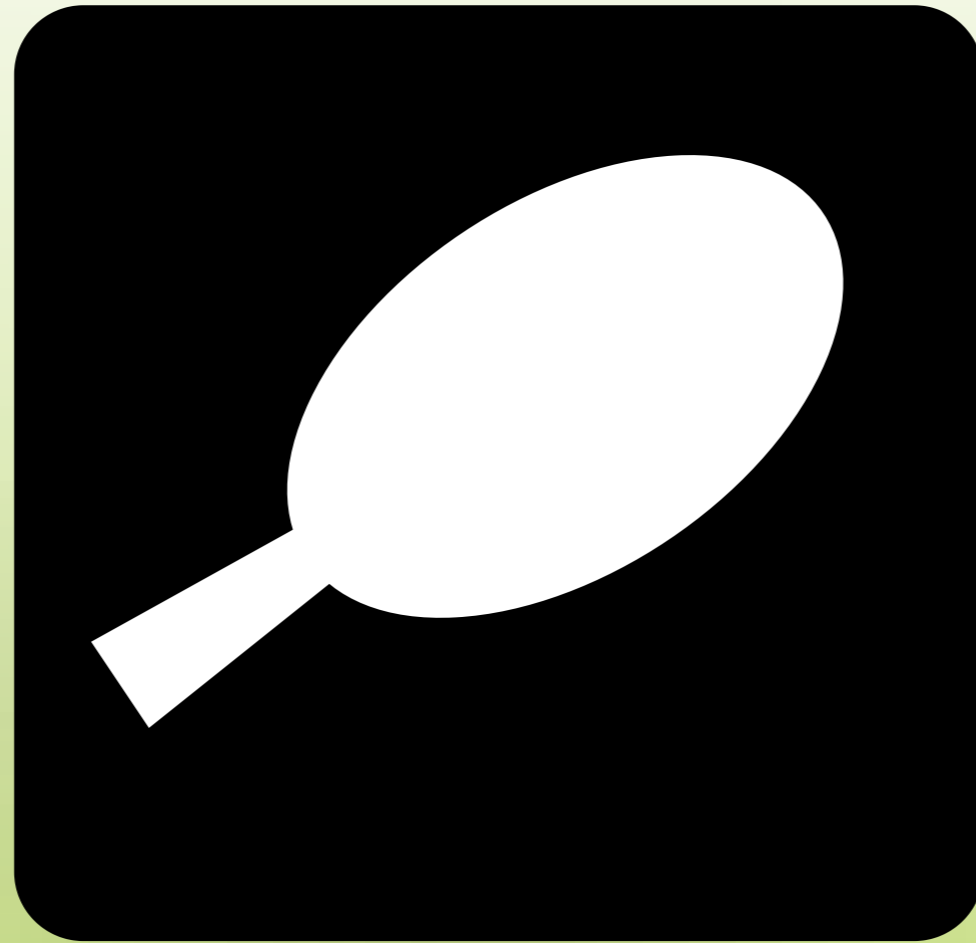
1. Design URIs
2. Select representations
3. Identify method semantics
4. Select response codes

Advantages

- Widely supported
 - Languages
 - Scripts
 - Browsers
- It works!

Advantages

- Scalability
- Redirects (versioning)
- Caching
- Different representations (no more attachment hell!)
- Identification



REST in Spring 3

Uri Templates

```
@Controller
public class HotelController {

    @RequestMapping(value="/hotels/{id}", method=RequestMethod.GET)
    public String getHotel(@PathParam String id) {
        // use id

        return "someView";
    }

    @RequestMapping(value="/hotels/{id}/bookings", method=RequestMethod.POST)
    public void addBooking(@PathParam String id, Booking booking) {
        // store booking
    }
}
```



Representations

- View resolution
- XML
 - Spring-WS OXM
- JSON
- Atom, RSS
- Flex

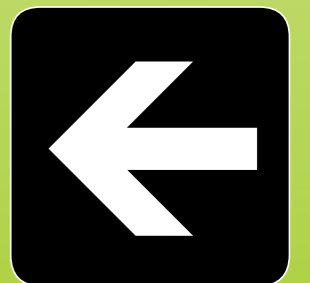


RestTemplate

- RestTemplate as core client-side component
- Similar to other templates in Spring
 - JdbcTemplate
 - JmsTemplate
 - WebServiceTemplate

RestTemplate methods

- `getForObject`
 - Performs GET and converts
- `put`
 - Performs PUT
- `postForLocation`
 - Performs POST, and retrieves Location header
- `delete`



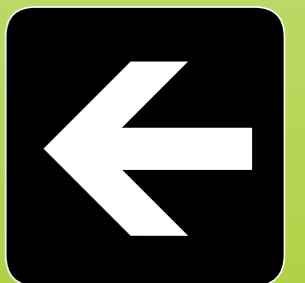
RestTemplate

```
String uri = "http://example.com/hotels/{id}"  
template = new RestTemplate();  
HotelList result = template.getForObject(uri,  
    HotelList.class, "1");
```

```
Booking booking = // create booking object  
uri = "http://example.com/hotels/{id}/bookings";  
Map<String, String> vars = Collections.singletonMap("id", "1");  
URI location = template.postForLocation(uri, booking, vars);
```

```
template.delete(location.toString());
```

```
template.execute(uri, HttpMethod.GET,  
    myRequestCallback,  
    myResponseCallback);
```



Time Line

- Part of Spring 3.0
- PathParam in 3.0 MI (October)
- Rest follows later
- 3.0 Final planned for early 2009



Q & A