JAX-RS Done Right

How to use JAX-RS correctly (and how not)

Markus KARG (Head Crashing Informatics, JSR 339, JSR 370)

JUG Switzerland, St Gallen, 2015-10-29

Legal Disclaimer

This presentation expresses solely my personal opinion and is not necessarily aligned with the official statement of any of my customers or employers, the JSR 339 and 370 Expert Group, Oracle Corp., or any other named company.

All trademarks belong to their particular owners, even if not declared explicitly.

The Cheeky™ comic character is used by courtesy of *inviticon*℠.

Bio (condensed)

Markus

Born 1973
ZX Spectrum (~1985)
State-Qualified Information Scientist (1997)
Java Addict (1997)
JAX-RS EG Member (JSR 339, 370)
JUG Switzerland – Zurich (April 24)
JUG Switzerland – St Gallen (now)

https://headcrashing.wordpress.com
markus@headcrashing.eu

How to write *ugly* JAX-RS code

```java
@Path("notebook")
public class BadNotebookResource {
	// Cannot replace for testing :-(
	NotebookApplication notebookApplication = new NotebookApplication();


	// Strings are pretty simple and straightforward to handle – but not type safe!
	@GET @Produces("text/plain")
	public List<String> getNotesAsStringList(@MatrixParam("from") String startDate, @MatrixParam("to") String endDate) {
		List<Note> notes = notesAsList(startDate, endDate);
		List<String> notesAsStrings = notes.stream().map(Note::toString).collect(Collectors.toList());
		// TODO Implement Note::toString!
		return notesAsStrings;
	}


	private List<Note> notesAsList(String startDate, String endDate) {
		return notebookApplication.getNotes(new TimeSpan(Instant.parse(startDate), Instant.parse(endDate))).asList();
	}


	// Hopefully JAXB support will not get deprecated sometimes... ;-)
	@GET @Produces("application/xml")
	public List<Note> getNotesAsXML(@MatrixParam("from") String startDate, @MatrixParam("to") String endDate) {
		List<Note> notes = notesAsList(startDate, endDate);
		// TODO Add @XmlRootElement to Note class
		return notes;
	}


	// Now we're rather screwed! :-(
	@GET @Produces("application/pdf")
	public PDF /* TBD */getNotesAsPDF(@MatrixParam("from") String startDate, @MatrixParam("to") String endDate) {
		List<Note> notes = notesAsList(startDate, endDate);
		PDF pdf = useFOP(notes); // TODO fix this!
		return pdf;
	}
}
```

Dude, **that's** really ugly!

Lesson #1: Choose Right API

Servlet API

- **You want to do *something* with HTTP.**
- Virtualizes web-servers
  - (Tomcat, Jetty, etc.)
- Layer 7 (HTTP)
- Request-oriented
- Slim and fast

JAX-RS

- **You want to write *RESTful applications*.**
- Virtualizes frameworks
  - (Jersey, RESTeasy, etc.)
- „Layer 8“ (Business)
- Domain-oriented
- Comes at a cost

Lesson #2: Use JAX-RS 2.0

JAX-RS 1.x

- **Nice idea.**
- Providers
- Auto-discovery
- JAXB
- Conditional Requests
- REST Level 2

http://martinfowler.com/articles/richardsonMaturityModel.html

JAX-RS 2.x

- **Now we're talking!**
- Features
- Configuration
- Filters, Interceptors
- Converter Providers
- Validation
- Basic Hypermedia
- Asynchronous Processing
- Client API
- REST Level 2.5

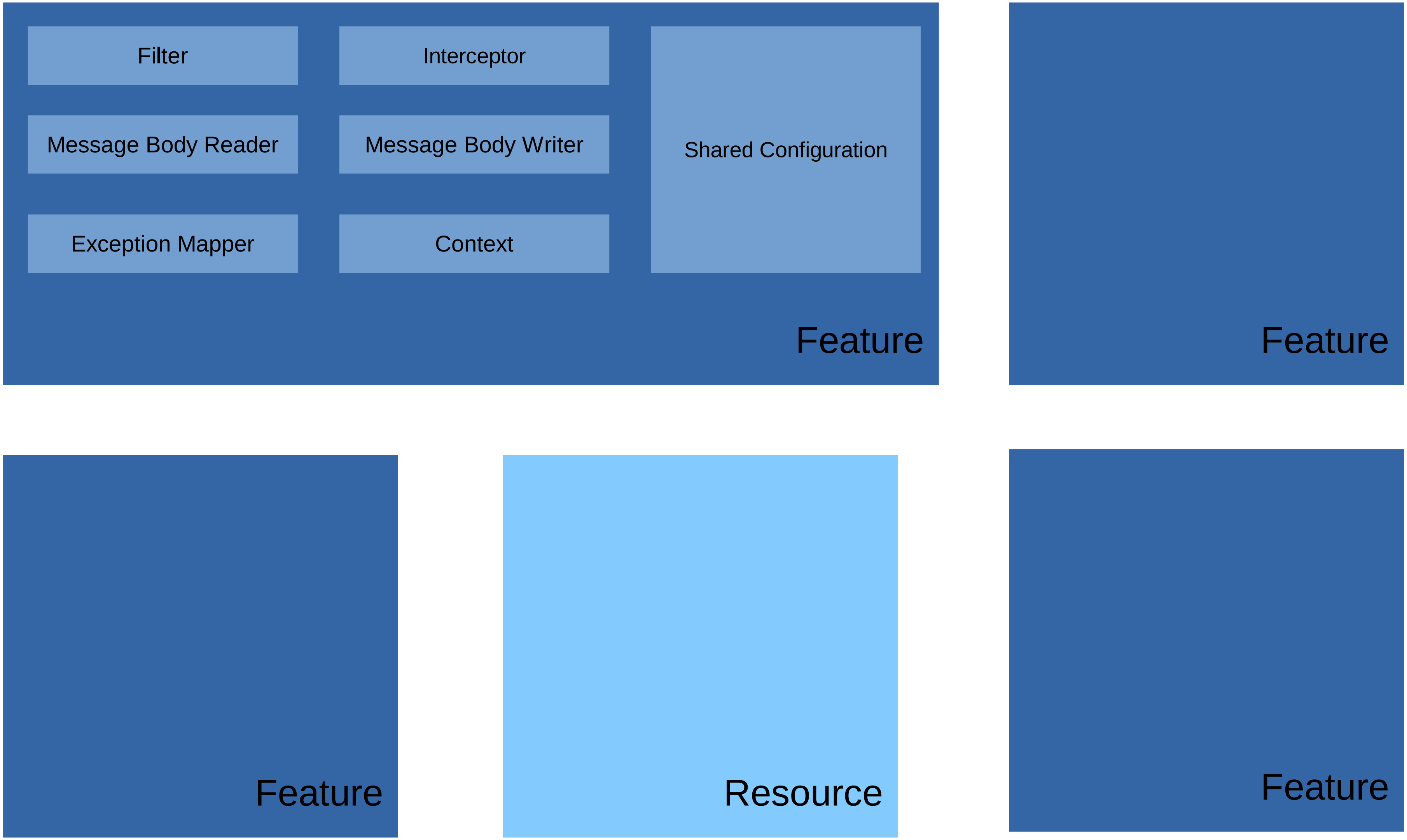
Lesson #3: Clean Business Object

- Apply CoC (Convention over Configuration)
  - Use annotations sparingly. You might actually not need them at all!
- Don't mix technology into business logic
  - Resource – „Business Service“
  - Entity – „Business Item“
  - Header – „Business State“
  - Exception – „Business Problem“
- Don't reference providers
- Let framework do the rest
  - Parsing and Rendering of State Representation (HTTP Entity)
  - Encoding / Decoding of additional information (HTTP Headers)
  - Result Code and Exceptions
  - Transfer Encoding, Compression, Caching
  - Dealing with URIs, Parameters, etc.

Lesson #4: Separate Aspects

- Apply SoC (Separation of Concerns)
  - Disintegrate monolithic application
  - Compose standalone components
- Think in features
  - „PDF Support“, „JSON Support“
  - „Compression Support“
  - „OData Support“
- **Know Your API**
  - Filters, Providers, Configuration, ...

Disintegrated Application



Filter

Interceptor

Message Body Reader

Message Body Writer

Shared Configuration

Exception Mapper

Context

Feature

Feature

Feature

Resource

Feature

How *nice* JAX-RS code looks like

```
@Path("notebook")
public class GoodNotebookResource {
    @Inject
    NotebookApplication notebookApplication;

    @GET
    public Notes getNotes(@BeanParam TimeSpan timeSpan) {
        return this.notebookApplication.getNotes(timeSpan);
    }
}
```

Ain't *that* cool?

The Magic behind JAX-RS

- Auto-detects features and global providers
- Auto-selects stuitable provider alternatives
- Manages component lifecycle
- Features configure providers
- Features are dynamic, optional and configurable
- Integrates with CDI, Bean Validation API, and EJB

Ingredients

- Providers
  - Message Body Readers and Writers
  - Parameter Converters
  - Context Resolvers
  - Exception Mappers
- Filters and Interceptors
  - The JAX-RS Swiss Army Knife
  - Can completely re-route, modify or even suppress requests and commits!
- Features
  - Dynamic Features are asked to register for each method AT DEPLOYMENT; can also bind globally
- Configuration
  - Shared among all components, application scoped
- Request and Response Properties
  - Forward information tags from one component to the next

Conclusion

- Application := ∑*Features*
- Marketplace with replaceable off-the-shelf *Features*
  - PDF Support
  - Encryption
  - Compression
  - Data Type Conversion (`Instant`, `Image`, `URL`, ...)
- Less `*.java`, more `pom.xml`

Got It?

The Bonus Slide: JAX-RS 2.1 Status

- *Oracle has better things to do than doing open source.*
  - *Oracle was rather inactive for many months.*
- *Reactivated Expert Group recently with* ***massively*** *reduced charter:*
  - *RX (Support for reactive programming using `CompletableFuture<T>`)*
  - *NIO (Improving scalability by decoupling thread cound from client count)*
  - *SSE (Pushing events to clients)*
  - *Alignment with MVC specification (JAX-RS based MVC controllers)*
  - *Support for JSON-B*