Advanced Software Engineering FOSS 2/2

Mark Struberg

Kontakt: teaching@inso.tuwien.ac.at

About me

- Mark Struberg
- ~25 years in the industry
- TU-Wien / INSO researcher
- Apache Software Foundation member
- struberg [at] apache.org
- Committer / PMC for Apache OpenWebBeans, MyFaces, TomEE, Maven, OpenJPA, BVal, Isis, DeltaSpike, JBoss Arquillian, ...
- Java JCP Expert Group member
- Twitter: @struberg

What interests me in FOSS?

- **FOSS: Free and Open Source Software**
- **Looking behind the scenes!**
  - Open Source is an easy way to look at how things *really* work
  - It's important to know the fundamental mechanism used in a tool to be able to judge side effects and solve problems the right way.
- **E.g.: How do virtual methods work?**
- **subroutine address modes in CPUs**
  - absolute
  - relative
  - indirect
- **Method overriding via Virtual Function Pointer Table**

Agenda

- General Considerations
- OSS Licenses
- The ASF in numbers
- How the ASF works
- The Apache License 2.0 (ALv2)
- How to Contribute?
- in between: Removing the shadows

General Considerations

The Weapon of Choice

- **"If you have a hammer, every problem seems to be a nail"**
- **"Use the right tool for the right job"**
- **Every design decision has pros and cons!**
  - There is no solution which perfcectly fits all your problems
  - Example: centralised vs de-centralised systems,
    App evolution in waves: HOST -> server/client PCs -> HTML webapps -> AJAX -> native phone apps -> ?

FOSS vs Closed Source

- **Pros and cons of commercial software tools**
- **Pros and cons of Open Source Software tools**
- **Big benefit of OSS if you hit a bug in a FOSS framework**
  - You don't need workarounds in your own code
  - Instead you can fix bugs directly in the OSS libraries you use
- **"Standard Software"**
- **Custom Development**
- **Project Development vs Product Development**
  - Developing a generic Product costs much more

FOSS vs Closed Source

- **Pros and cons of commercial software tools**
- **Pros and cons of Open Source Software tools**
- **Big benefit of OSS if you hit a bug in a FOSS framework**
  - You don't need workarounds in your own code
  - Instead you can fix bugs directly in the OSS libraries you use
- **"Standard Software"**
- **Custom Development**
- **Project Development vs Product Development**
  - Developing a generic Product costs much more

The License Situation

- **Commercial Licenses**
- **GPL**
- **MIT / BSD**
- **ALv2**
- **Examples for not really open OSS licenses (not OSI approved):**
  - do-no-evil-license
  - beer-license
- **Do you have a right on the sources at all?**
- **Always add some License! (re github)**
  - If you don't add any license to your published sources then no one can use them because they have no rights.

Patents

- **Some licenses contain a 'patent grant'**
  - Apache License v 2.0 (ALv2)
  - GNU Public License v3.0 (GPLv3)
  - Mozilla Public License v1.1 (MPL)
- **Software Patents are allowed in the US**
  - at least it is handled that way right now
- **Software Patents are <u>not</u> allowed in the EU**
  - but they give a damn about that...

Copyright

- **The right to do whatever I want with a certain piece of work.**
- **Difference between 'Urheberrecht' (non-dispositive, ius cogens) and 'Verwertungsrechte' in AT**
- **in EU -> implicit**
- **in US -> rather explicit**

Code Provenance

- **Where does the code come from?**
- **This is <u>utter</u> important for big companies!**
  - e.g. if they get sued or to show prior art
- **Reason why ASF doesn't accept unverified pull requests from github**
- **ASF requires signed iCLA for non-trivial contributions**

Trademarks

- **Choose your mark wisely!**
- **Name must be unique (in your field)**
  - google
  - trademarkia.org
  - EU trademark registration office
- **Actively defend your marks**
  - Marks vanish if they frequently get used by others without proper attribution
  - require attribution
    "Apache Foo is a trademark of the apache Software Foundation"
  - http://www.apache.org/foundation/marks/
- **Establish a mark without restricting the economy around your project**
  - e.g. the Apache Hadoop ecosystem
  - allow 'powered by...', '... for Apache Foo'

Ways to monetise OSS

- **Adding commercial available value on top of a base OSS offering**
  - RedHat
  - JBoss AS, IBM WebSphere
- **Professional Training**
- **Embedding OSS into hardware**
  - most routers use Linux, iptables, etc
  - Android
- **Service Contracts**
  - 'insurance' for your project
- **Sharing the costs - saving money!**
- **Project Consulting**
  - Help other companies to save money by using OSS.

What to avoid

- **Don't try to sell the same product you give away for free**
- **Respect the freedom!**
  - Don't force/restrict others to do something only because *you* need it
  - Respect the community
- **Don't rely (only) on a payroll**
  - don't push a project into one corner just because a customer likes it that way
- **OSS project planing is different than company projects**

Commercial impact of OSS

- **Most entertainment products nowadays use OSS inside**
  - Almost all modern TV
  - Almost all Blueray players
- **Android: ALv2**
- **Apple OSX/IOS (FreeBSD)**
- **Most Servers**
- **OpenJDK**
- **more than 50% of all software is OSS**

About the
Apache Software Foundation

History of the ASF

- **1994 - bunch of hackers started The Apache WebServer**
- **1998 - first 'commercial' companies kicked in**
- **1999 - 'ASF' officially formed as foundation**
- **1999 - Apache Jakarta got started**
  - in corporation with SUN microsystems
  - first Java projects at ASF
  - JSP and Servlets RI
  - tomcat, ant, maven, struts, etc
- **2002 - Apache Incubator got started**
- **2004 - Apache License v 2.0**
- **2014 - 142 TLP, 3729 committers, ...**

Founding Principles

- **"The Foundation exists to serve the development community"** - Bill Stoddard
- **"Let developers focus on what they do best: Code. The foundation exists to do the rest"** - Justin Ehrenkrantz
- **Not-for-profit Organisation**
- **Volunteer Organisation**
- **Not affiliated with nor controlled by any government or commercial entity**

Mission Statement

- **Provide legal and technical infrastructure for open source software development and to perform appropriate oversight of such software.**
- **http://www.apache.org**
- **Apache projects are defined by**
  - collaborative consensus based processes,
  - an open, pragmatic software license and
  - a desire to create high quality software that leads the way in its field.
- **"Community over Code"**
- **"Community for Code"**

The Apache License v 2.0 (ALv2)

- **http://www.apache.org/licenses/LICENSE-2.0**
- **Liberal open source software license**
- **Business friendly**
- **Requires attribution**
- **Includes Patent Grant**
- **Can be sub-licensed**
  - not to confuse with re-licensing, which is <u>not</u> allowed!
  - sub-licensing: allows adding your <u>own</u> code to the existing one in any license you like
  - re-licensing: would allow to change the license of all existing code

ASF Structure

- Contributors
- Committers
- PMC Members
- ASF Members
- Board of Directors
- Public Relations, Infra, Legal advisors

Contributor

- Report bugs
- Share ideas
- Get feedback on the mailing lists
- Provide patches via JIRA
- Provide Code and new modules via patches
- ASF requires a signed iCLA for non-trivial changes

Committer

- **Merit must be earned!**
- **Public recognition for your existing contributions**
- **Write-access to your project SCM**
- **Own yourid@apache.org email address**
- **http://people.apache.org/~yourid/ web space**

Project Management Comittee Member

- **The PMC manages a TLP and provides the direction for the project**
- **Binding count in release votes and fundamental decisions**
- **Must provide legal oversight of the project**
- **Access to private project mailing list**

Apache Software Foundation Member

- Legal member of the non-for-profit organisation
- Right to elect the board and official positions
- Access to almost everything within the ASF
- Must get voted in by existing members

Board of Directors

- **9 members elected annually**
- **Official representation of the ASF**

The Incubator

- **Own TLP which 'shepherds' new candidate projects**
- **Goal is to make those projects (code and people) 'mature' and TLPs on their own.**
- **Ensure all donations meet ASF technical standards**
- **Ensure all donations meet ASF legal standards**
  - provenance checks and IP clearance
- **Ensure that the community is diverse enough**
- **Incubation is a good chance to build up a broad community**
- **"Becoming an Apache project is a process, not just a decision" - Bertrand Delacrétaz**
- **http://incubator.apache.org**

How to contribute

- **Pick a project**
- **Become familiar with the topic**
- **Start reading the mailing list**
  - you can also read the archives via markmail, nabble, etc
- **Check out the Source Code**
- **Use the project!**
- **Start reporting bugs...**
- **... and ship patches.**

The 'Apache Way'

- **Goals**
  - Reduce barriers to project participation
  - Improve quality
  - Achieve consensus and resolve conflicts
  - Balance needs of corporate interest with needs of individual contributors

And now for something completely different (the fun parts)

Having fun with Java Bytecode

- **Java is basically a virtual CPU**
- **The Sun JVM works stack based**
- **The Android VM works register based**

How does JPA work?

JPA - A simple Entity

- **A JPA Entity is mapped to one or more DB tables.**

```
import javax.persistence.*;
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private Long id;

  private String name;
  private int age;

  @Lob
  @Basic(fetch=FetchType.LAZY)
  private String comment;
  // + getter and setter
}
```

JPA - How does the magic work?

- **ORM and state tracking**
- **2 ways to implement this**
  - comparison against 'old' loaded values stored in the EntityManager (Hibernate)
  - dirty/loaded tracking in the entities itself (OpenJPA, EclipseLink)

JPA data handling

- **We simply change the Java Bytecode 'on the fly'**
- **Created for all setters and getters + field access**

```
public String getName() {
  if (!isLoaded(NAME_FIELD)) {
    this.name = loadFromDB(NAME_FIELD);
  }
  return this.name;
}

public void setName(String name) {
  if (safeEquals(this.name, name) {
    return;
  }
  setDirty(NAME_FIELD);
  this.name = name;
}
```

How does CDI work?

What is CDI

- **Contexts and Dependency Injection for Java Enterprise**
- **JSR-299, JSR-346, JSR-365 (wip)**
- **Standard framework for DI in Java**
- **since Java EE6**

A small Example

- **Create a META-INF/beans.xml marker file**
- **Create a MailService implementation**

```
@ApplicationScoped
public class MyMailService
implements MailService {
  public send(String from, String to,
              String body) {
    .. do something
  }
}
```

A small Example

- **We also need a User bean**

```
@SessionScoped
@Named
public class User {
  public String getName() {..}
  ..
}
```

A small Example

- **Injecting the Bean**

```
@RequestScoped
@Named(„mailForm“)
public class MailFormular {
  private @Inject MailService mailSvc;
  private @Inject User usr;
  public String sendMail() {
    mailSvc.send(usr.getEmail(),
                 „other“, „sometext“);
    return “successPage”;
  }
}
```

A small Example

- **Use the beans via Expression Language**

```
<h:form>
  <h:outputLabel value="username"
   for"username"/>
  <h:outputText id="username"
   value="#{user.name}"/>
  <h:commandButton value="Send Mail"
     action="#{mailForm.sendMail}"/>
</h:form>
```

How Dependency Injection works?

- **Uses Inversion Of Control pattern for object creation**
- **Hollywood Principle**
  - "Don't call us, we call you"
- **Macho Principle**
  - "Gimme that damn thing!"
- **Contextual Instance factory pattern**
  - recursively fill all InjectionPoints

'Singletons', Scopes, Contexts

- **Each created Contextual Instance is a 'Singleton' in a well specified Context**
- **The Context is defined by it's Scope**
  - @ApplicationScoped ->ApplicationSingleton
  - @SessionScoped -> SessionSingleton
  - @ConversationScoped -> ConversationSingleton
  - @RequestScoped -> RequestSingleton
  - ... code your own ...

Software Design Patterns

- Categorised common solutions for common problems
- Introduced by Ward Cunningham and Kent Back in 1987
- More public attention in 1994 via Erich Gamma et al 'Design Patterns - Elements of Reusable Object Oriented Software' ISBN-10 0201633612
- Well known Design Patterns:
  - Singleton
  - Factory
  - Delegate
  - Builder
  - Proxy

Terms – „Bean“

- The term Bean means a Java Class and all it's rules to create contextual instances of that bean.
- 'Managed Beans' in JSR-299 and JSR-346 doesn't mean JavaBeans!
  - It does NOT mean @ManagedBean from EE-6
  - it does NOT mean @ManagedBean from JSF-2
- Interface Bean<T> extends Contextual<T>

The Bean<T> interface

```
public interface Contextual<T> {
  T create(CreationalContext<T> creationalContext);
  void destroy(T instance, CreationalContext<T> cc);
}

public interface Bean<T> extends Contextual<T> {
  Set<Type> getTypes();
  Set<Annotation> getQualifiers();
  Class<? extends Annotation> getScope();
  String getName();
  Set<Class<? extends Annotation>> getStereotypes();
  Class<?> getBeanClass();
  boolean isAlternative();
  boolean isNullable();
  Set<InjectionPoint> getInjectionPoints();
}
```

Terms – Contextual Instance

- **The term 'Contextual Instance' means a Java instance created with the rules of the Managed Bean `Bean<T>`**
- **Contextual Instances usually don't get injected directly!**

Terms – Contextual Reference

- **The term 'Contextual Reference' means a proxy for a Contextual Instance.**
- **Proxies will automatically be created for injecting @NormalScope beans.**

How Proxies work

- **'Interface Proxies' vs 'Class Proxies'**
- **Generated subclasses which overload all non-private, non-static methods**

```
public class User$$Proxy extends User {
  public String getName() {
    return getInstance().getName();
  }
  private T getInstance() {
    beanManager.getContext().get(...);
  }
}
```

- **Can contain Interceptor logic, Decorators,**
  - Usually in a 'MethodHandler' using reflection

Why use Proxies

- scope-differences - this happens e.g. if a @SessionScoped User gets injected into a @ApplicationScoped MailService.
- Passivation of non-serializable contextual instances (via it's contextual reference)
- Interceptors, Decorators
- Circular Injection A->B->A
- Non-normalscoped beans only get a proxy if they have an Interceptor or Decorator

Unproxyable Bean Types

- **The following classes cannot be proxied**
  - classes which don't have a non-private default ct
  - classes which are declared final or have non-static final methods,
  - primitive types,
  - array types.

Questions?