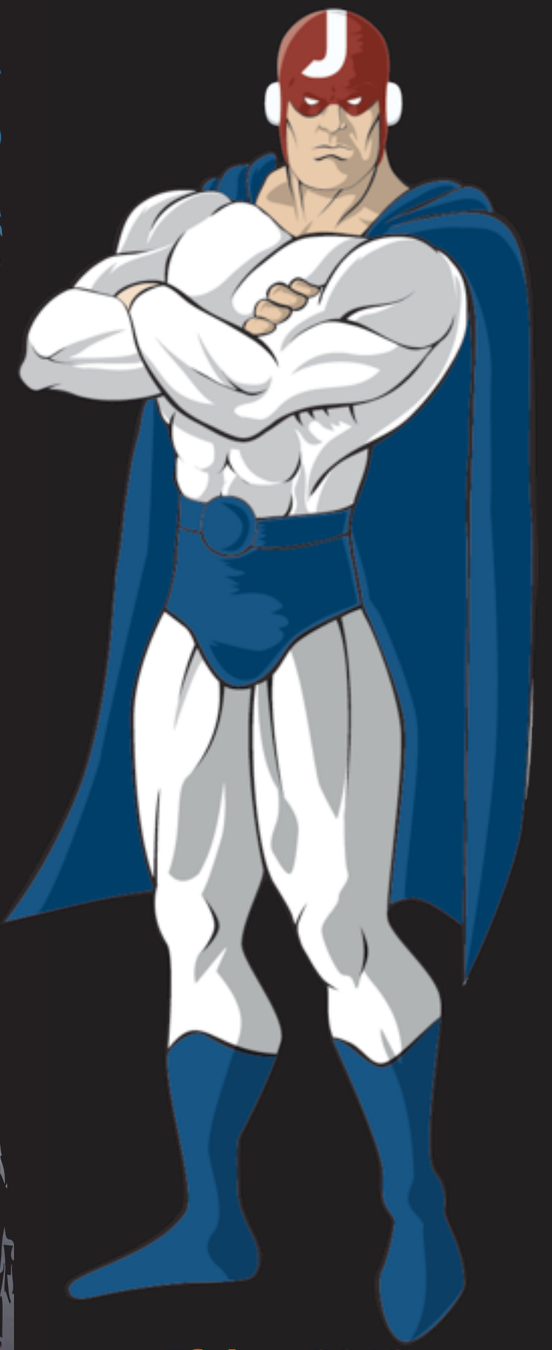
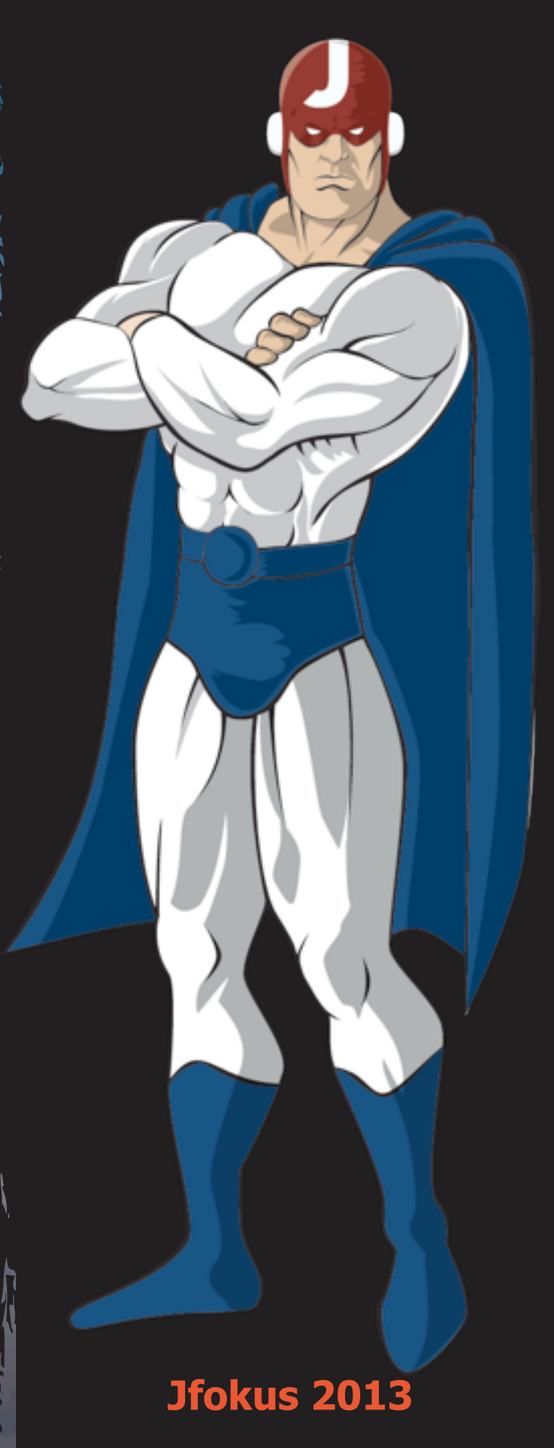
Modular Java EE in the cloud

A practical guide to mixing java EE and OSGi



Jfokus 2013

Practical Guide to Modularity in the Cloud Age



Jfokus 2013

Bert Ertman

Fellow at Luminis in the Netherlands

JUG Leader for NLJUG and a Java Champion

@bertertman

Paul Bakker

Architect at Luminis Technologies

@pbakker

A presentation for

Trend

- Applications tend to grow bigger and more complex
- Agile development and refactoring have become more common

This leads to a number of challenges :

Dependency Management

Versioning

Maintenance (long term)

Deployment

Apps are moving to the cloud

Cloud challenges require non-trivial non-functional requirements

- Zero-downtime deployments
- Modular deployments
- Customer specific extensions (saas)

Modularity

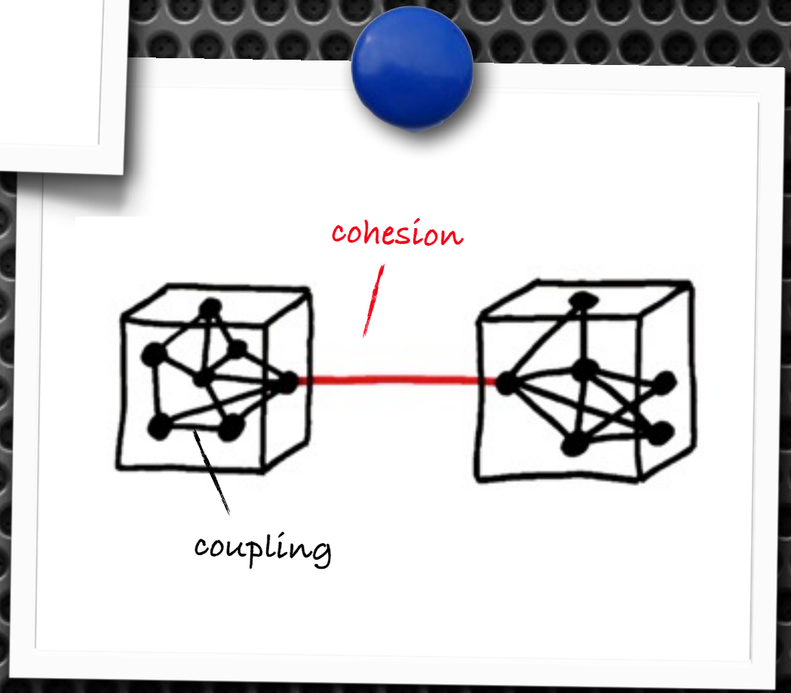
is the answer

But... what exactly is modularity?

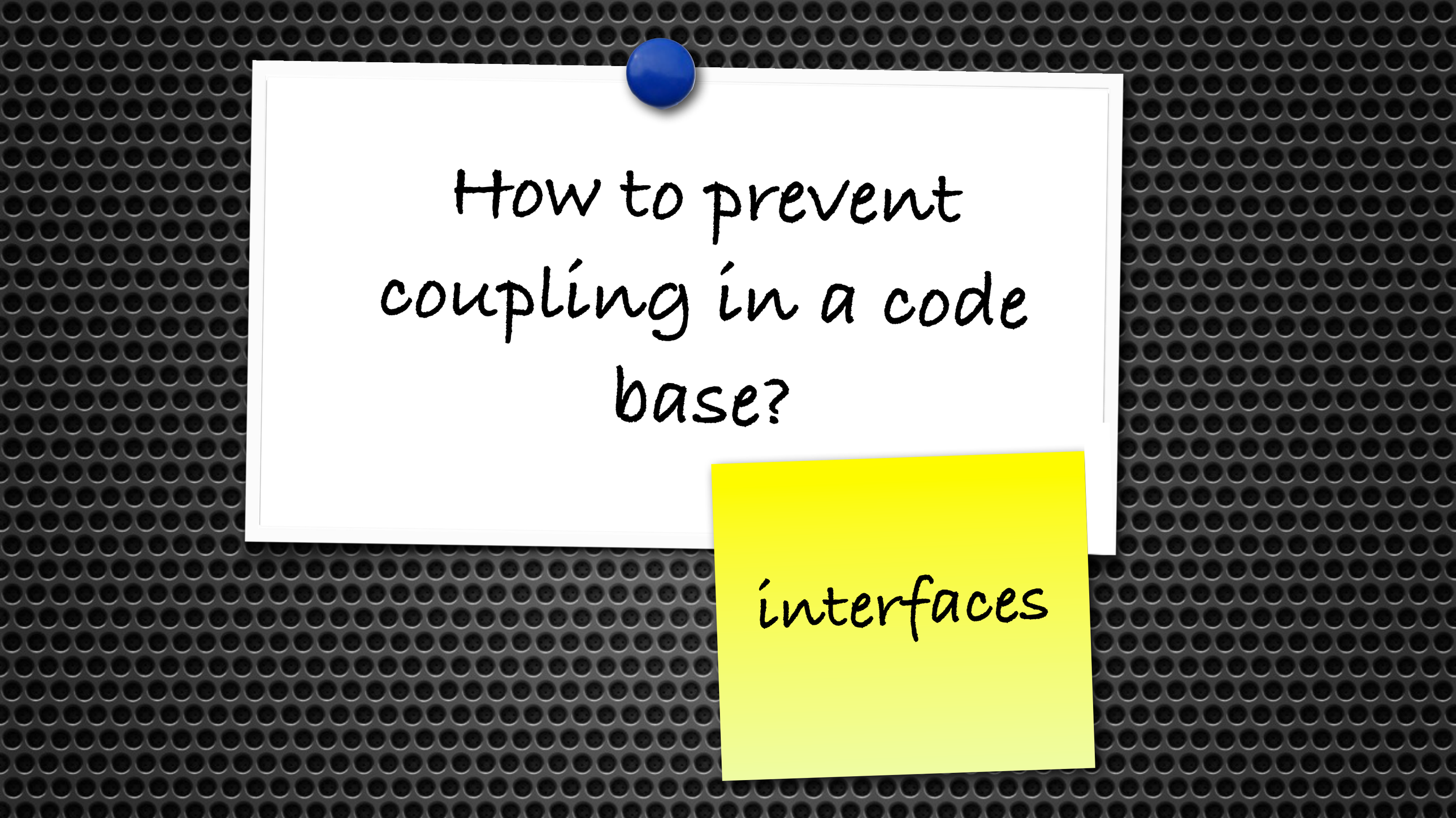
What we learned about OO design in university :

Promote cohesion

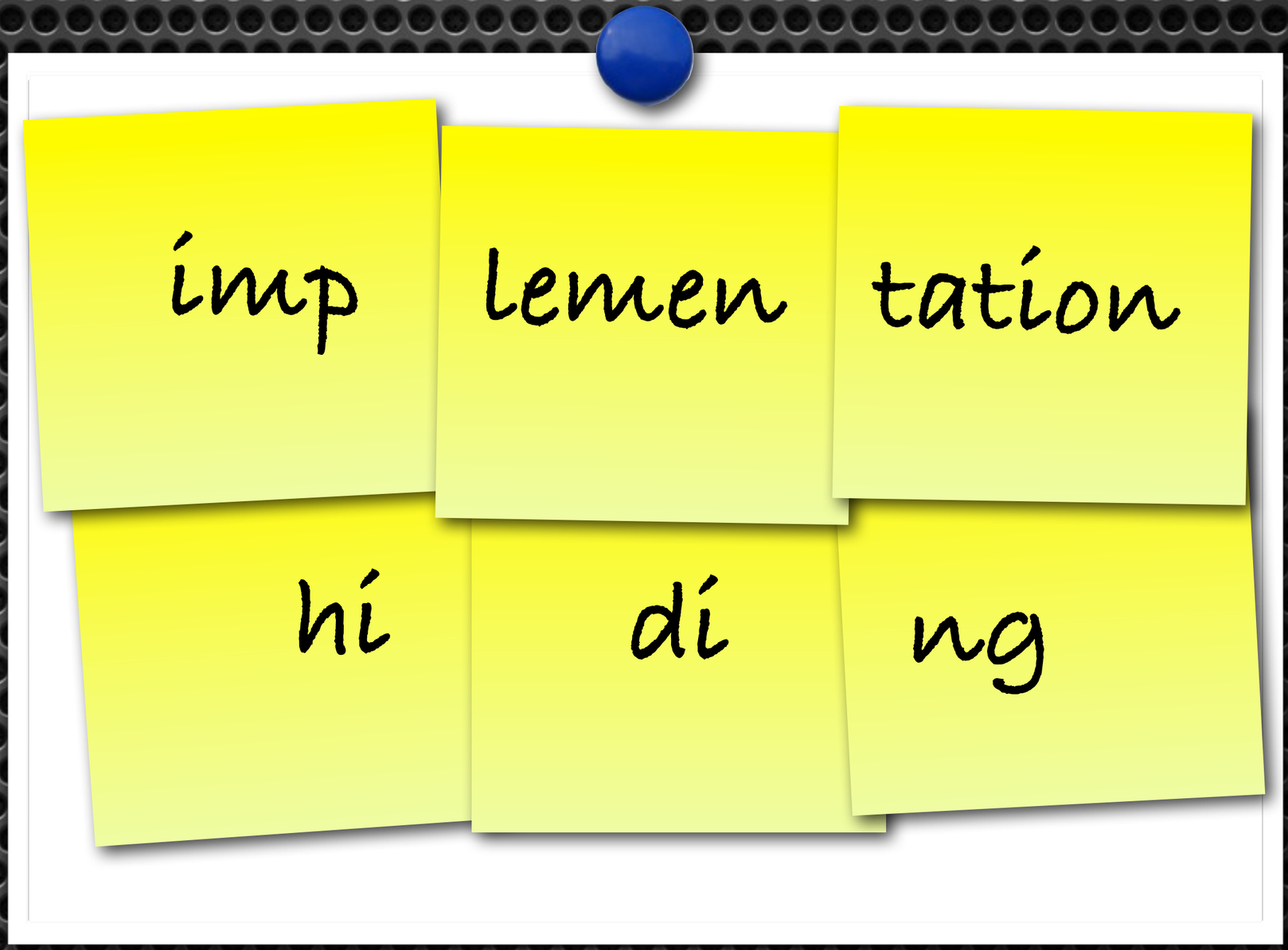
Prevent (tight) coupling



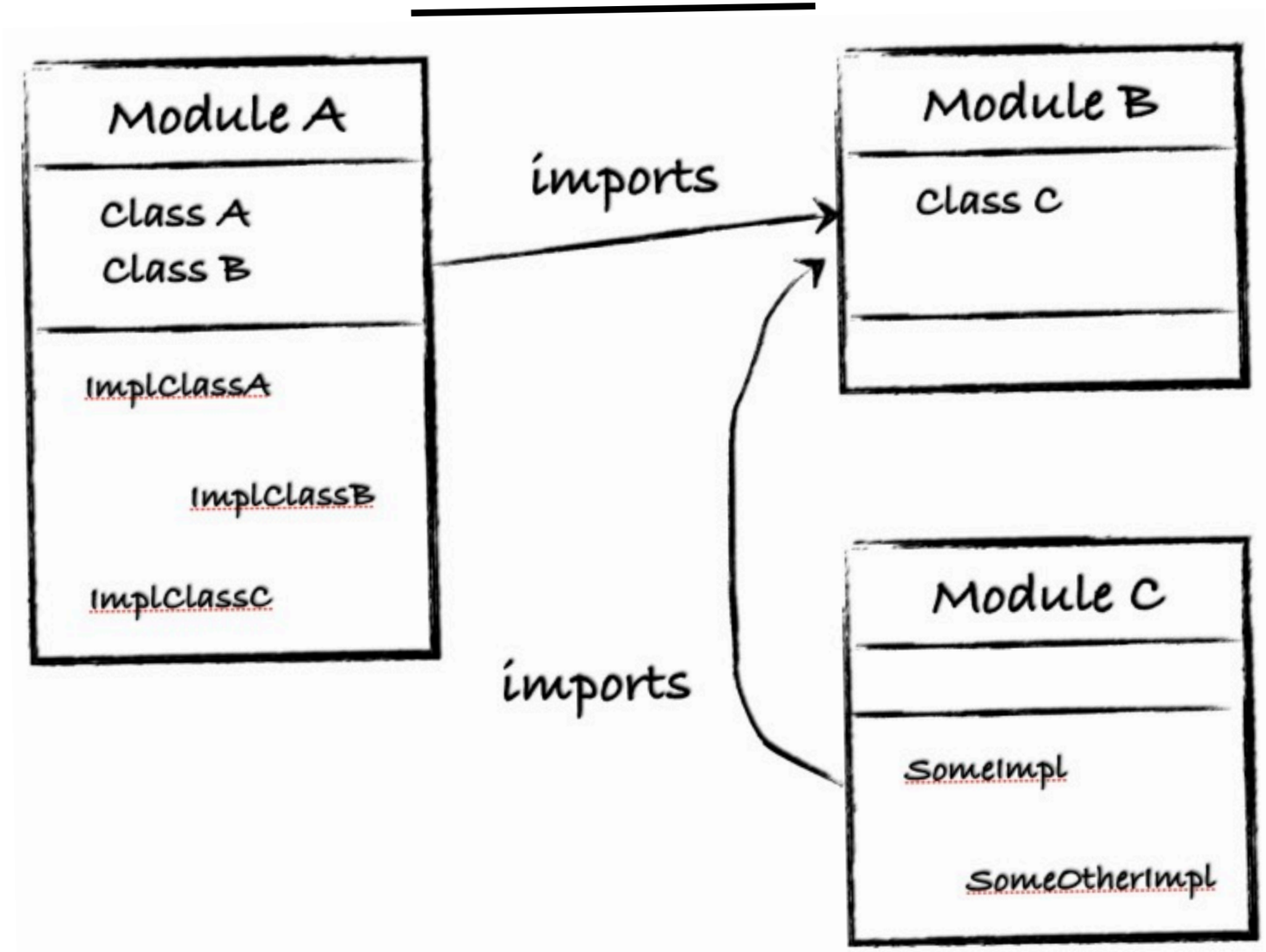
How to prevent coupling in a code base?


How to prevent coupling in a code base?
interfaces

But how to make sure nobody accidentally uses implementation classes?


imp
lemen
tation
hi
di
ng

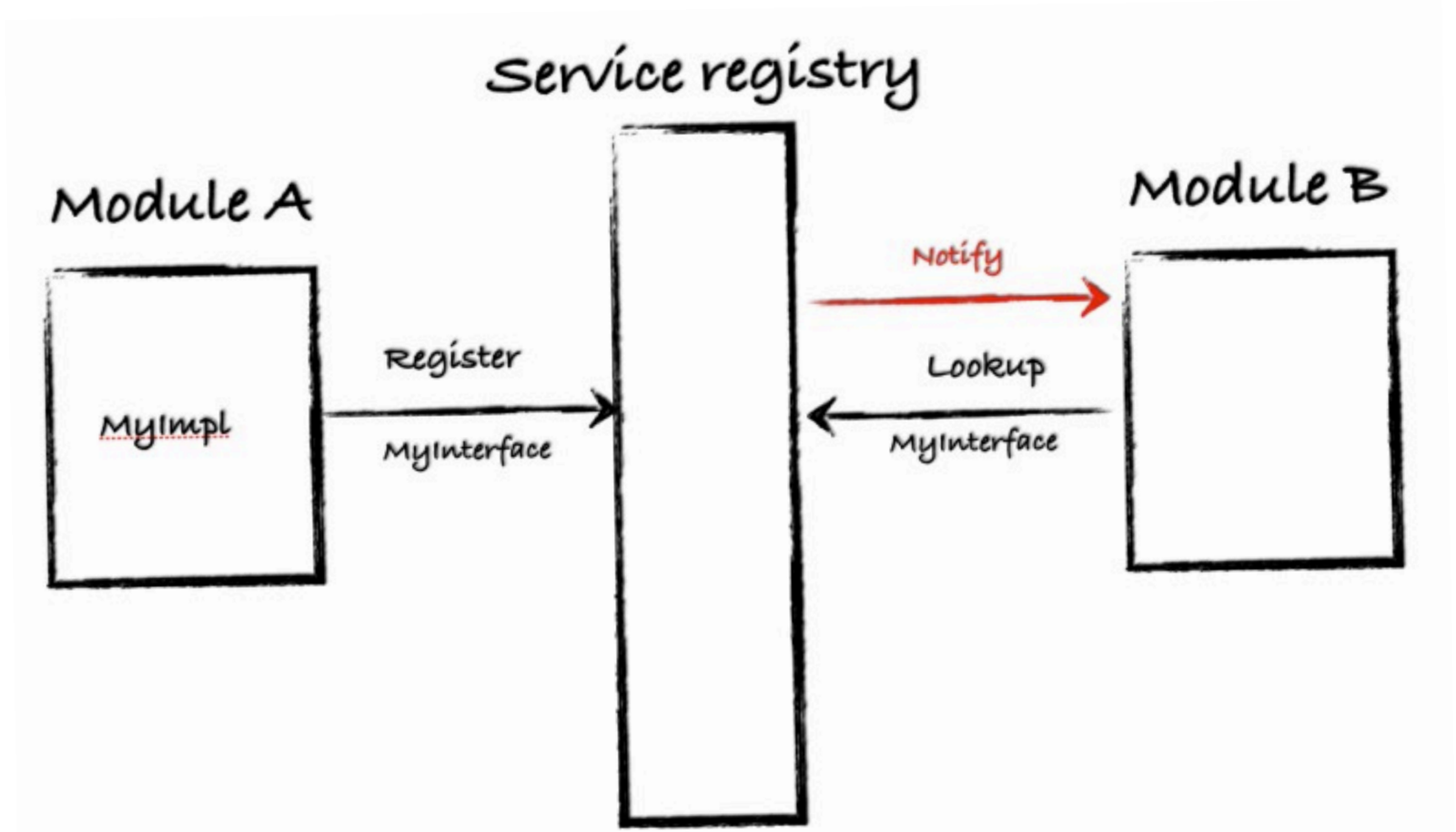
Modules


Module A
Class A
Class B
ImplClassA
ImplClassB
ImplClassC
imports
Module B
Class C
imports
Module C
SomeImpl
SomeOtherImpl

Ok, but how to create an instance of a hidden class?

```
MyInterface myI = new MyImplementation();
```

Service Lookups



Design time modularity

- Divide and conquer
- Forces separation of concerns
- Prevents spaghetti

The jar file is the unit of

runtime modularity

on the Java platform

What about classpath hell?

How to deal with

Runtime dynamics

- Module versioning
- Updating single modules
  - Intra-module dependency management

What do we need?

design consequences

Architectural focus on modularity

Runtime dynamic module framework

High-level enterprise APIs

let's not reinvent the wheel

Right now, OSGi is the only option

OSGi != modularity

OSGi is the de-facto standard module system for Java

What about Jigsaw?

OSGi is the de-facto standard module system for Java

OSGi != modularity

remember!
Modularity is an architectural principle

What about Jigsaw?

OSGi is the de-facto standard module system for Java

OSGi != modularity

remember!
Modularity is an architectural principle

Modularity does not come for free with a framework

What about Jigsaw?

Demo

OSGi vs. Java EE

OSGi vs. Java EE : opposite ends?

Java EE:
high level
APIs

OSGi:
low-level, mix
and match
yourself

OSGi vs. Java EE : opposite ends?

Java EE:
high level APIs

They don't have to be either / or

OSGi:
low-level, mix and match yourself

How? - 3 options

#1

OSGi as core architecture with EE APIs published as services

#2

Hybrid solution using an injection bridge

#3

OSGi a la carte

Option 1 - OSGi in Java EE App Servers

OSGi bundles as your core programming model

Java EE APIs available from within bundles

use app server capabilities like clustering, data sources, etc.

the (near) future!

Web Application Bundle (WAB)

Enterprise OSGi standard

WAR with a manifest

Easy registration of Servlets and JAX-RS components

Web Application Bundle (WAB)

mywab.jar

```
META-INF/manifest.mf
index.html
WEB-INF/classes/
MyResource.class
WEB-INF/faces-config.xml
WEB-INF/web.xml
```

/myweb/index.html

/myweb/agenda

manifest.mf

```
Bundle-Name: agenda.web.ui
Web-ContextPath: /myweb
Import-Package: javax.servlet
```

```java
@Path("agenda")
public class MyResource {
    @GET
    public String demo() {
        return "hello";
    }
}
```

EJB Bundles

Not a standard yet!

Use transactions and JPA like you normally would

Publish EJBs as OSGi services

EJB Bundles

```
@Local(AgendaStorage.class)
@Stateless
public class EjbAgenda implements AgendaStorage {

    @PersistenceContext EntityManager em;

    //...
```

manifest.mf

```
Bundle-Name:
agenda.storage.ejb
Export-EJB: ALL
```

OSGi CDI integration

RFP 146

Prototyped in WELD

Use CDI within bundles

Use CDI to produce / consume services

How are the app servers doing?

App server	Deploy bundles	WAB support	EJB/JPA support
Glassfish	✓	✓	✓
WebSphere	✓	✓	✓
WebLogic	✓	✓	✓
JBoss AS	✓	✓	✓

Option 2 : Hybrid solution

Best of both worlds?

- Your app is effectively split into two parts:
  - 'Adminstrative' enterprise part -> Java EE
  - 'Dynamic' part where services can be replaced without downtime -> OSGi
- Glue the two together using an 'injection bridge'

Example

```
@Component
public class HelloServiceImpl implements HelloService, Serializable {
    @ServiceDependency
    private volatile LogService logger;

    @Override
    public String hello(String name) {
        logger.log(LogService.LOG_INFO, "Hello " + name + ".");
        return "Hello, " + name + ".";
    }
}
```

```
@WebServlet(urlPatterns={"/worker"})
public class WorkerServlet extends HttpServlet {
    @Resource BundleContext bundleContext;

    public WorkerServlet() {
        super();
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();

        ServiceReference serviceReference = bundleContext.getServiceReference(HelloService.class.getName());
        if (serviceReference != null) {
            HelloService helloService = (HelloService) bundleContext.getService(serviceReference);
            out.println(helloService.hello("World"));
        }
    }
}
```

Option 3 - OSGi a la carte

Demo

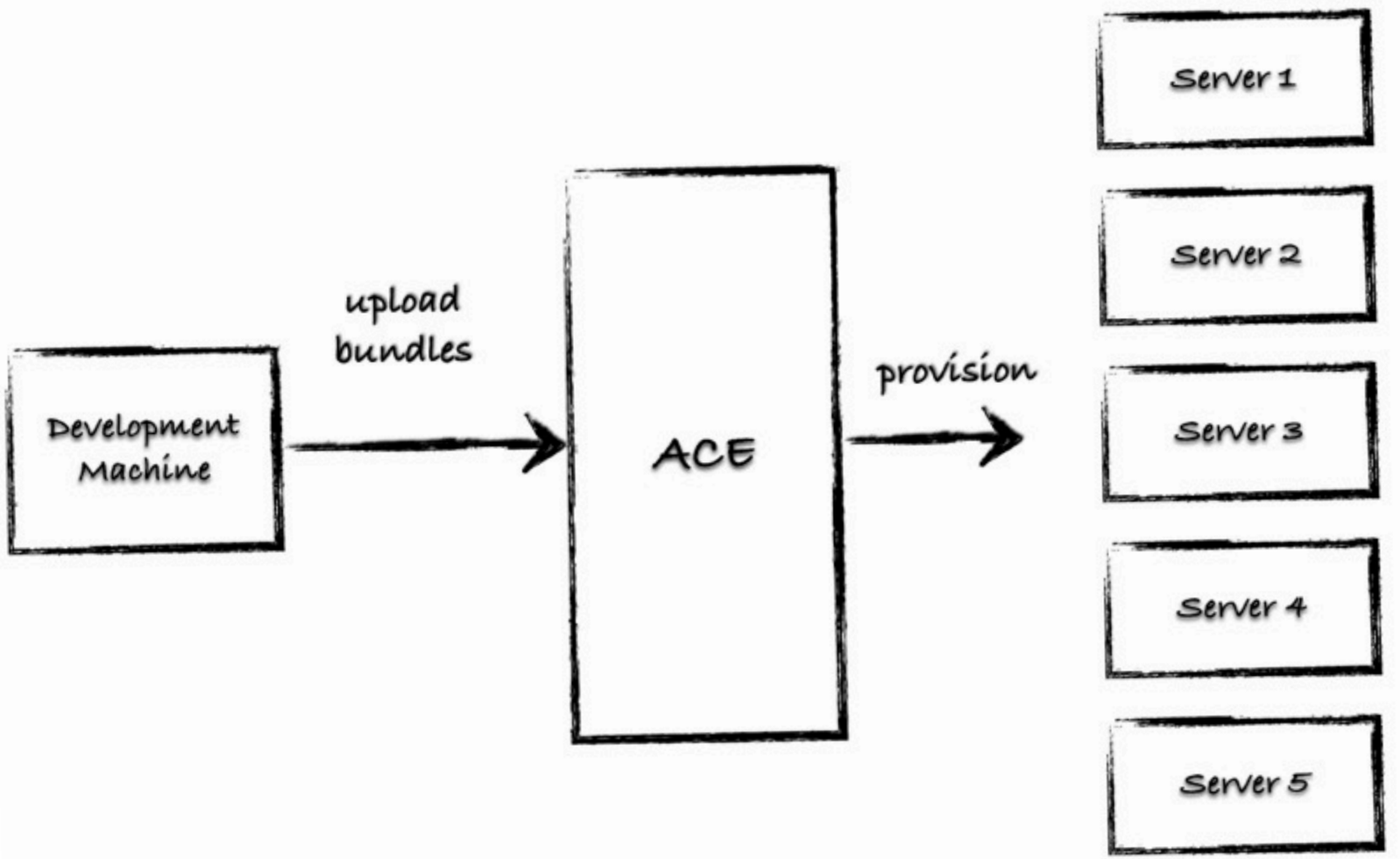
What about deployment?

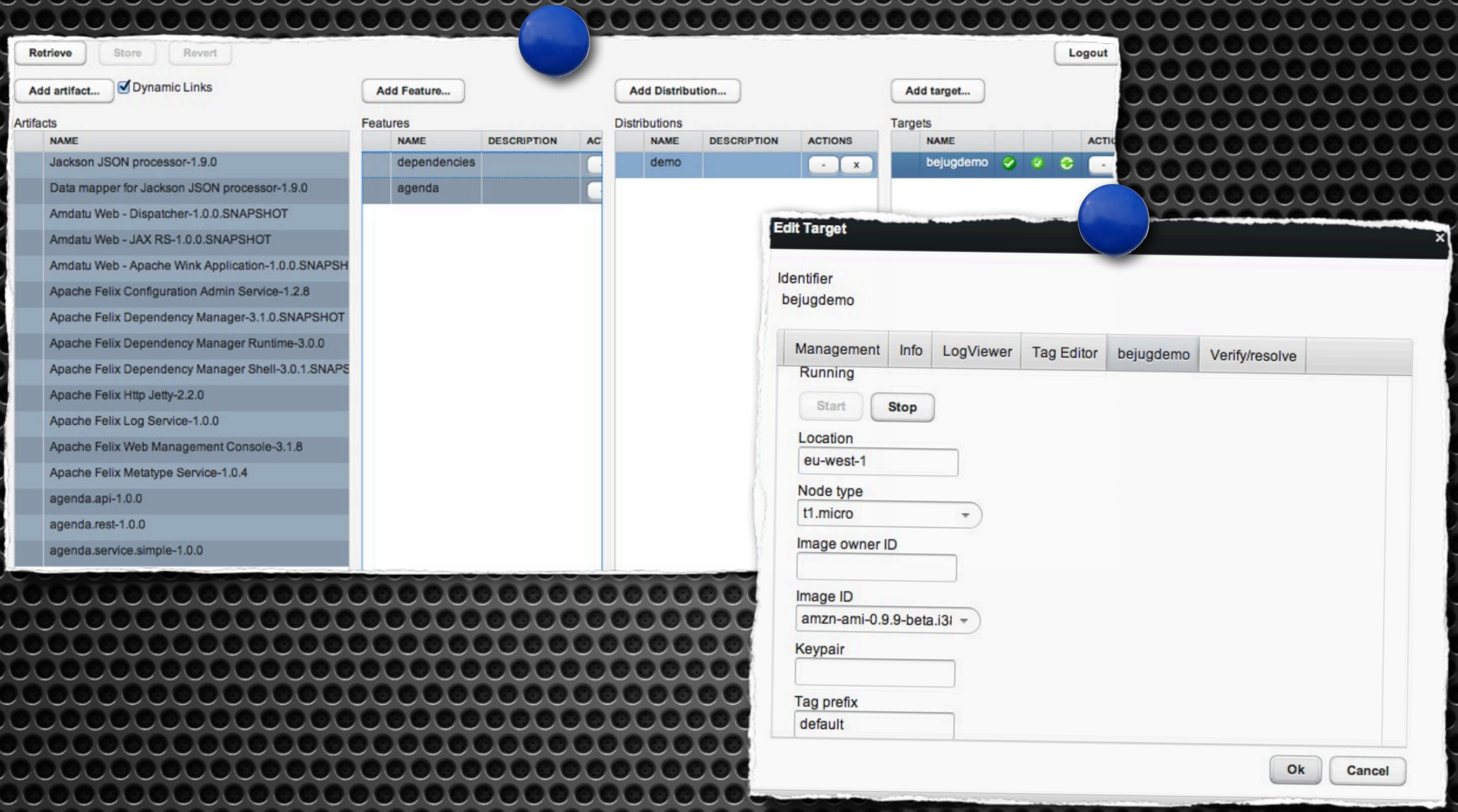
What about deployment?

Apache Ace

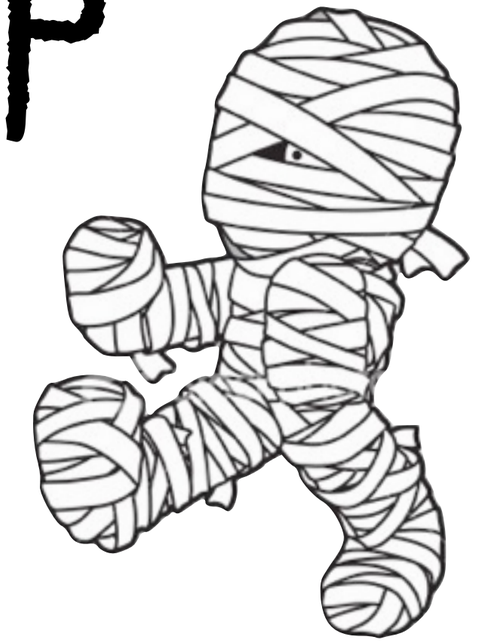
- demo -

Apache ACE


Development Machine
upload bundles
ACE
provision
Server 1
Server 2
Server 3
Server 4
Server 5


Retrieve
Store
Revert
Logout
Add artifact...
Dynamic Links
Add Feature...
Add Distribution...
Add target...
Artifacts
NAME
Jackson JSON processor-1.9.0
Data mapper for Jackson JSON processor-1.9.0
Amdatu Web - Dispatcher-1.0.0.SNAPSHOT
Amdatu Web - JAX RS-1.0.0.SNAPSHOT
Amdatu Web - Apache Wink Application-1.0.0.SNAPSH
Apache Felix Configuration Admin Service-1.2.8
Apache Felix Dependency Manager-3.1.0.SNAPSHOT
Apache Felix Dependency Manager Runtime-3.0.0
Apache Felix Dependency Manager Shell-3.0.1.SNAPS
Apache Felix Http Jetty-2.2.0
Apache Felix Log Service-1.0.0
Apache Felix Web Management Console-3.1.8
Apache Felix Metatype Service-1.0.4
agenda.api-1.0.0
agenda.rest-1.0.0
agenda.service.simple-1.0.0
Features
NAME
DESCRIPTION
dependencies
agenda
Distributions
NAME
DESCRIPTION
ACTIONS
demo
Targets
NAME
bejugdemo
Edit Target
Identifier
bejugdemo
Management
Info
LogViewer
Tag Editor
bejugdemo
Verify/resolve
Running
Start
Stop
Location
eu-west-1
Node type
t1.micro
Image owner ID
Image ID
amzn-ami-0.9.9-beta.i3
Keypair
Tag prefix
default
Ok
Cancel

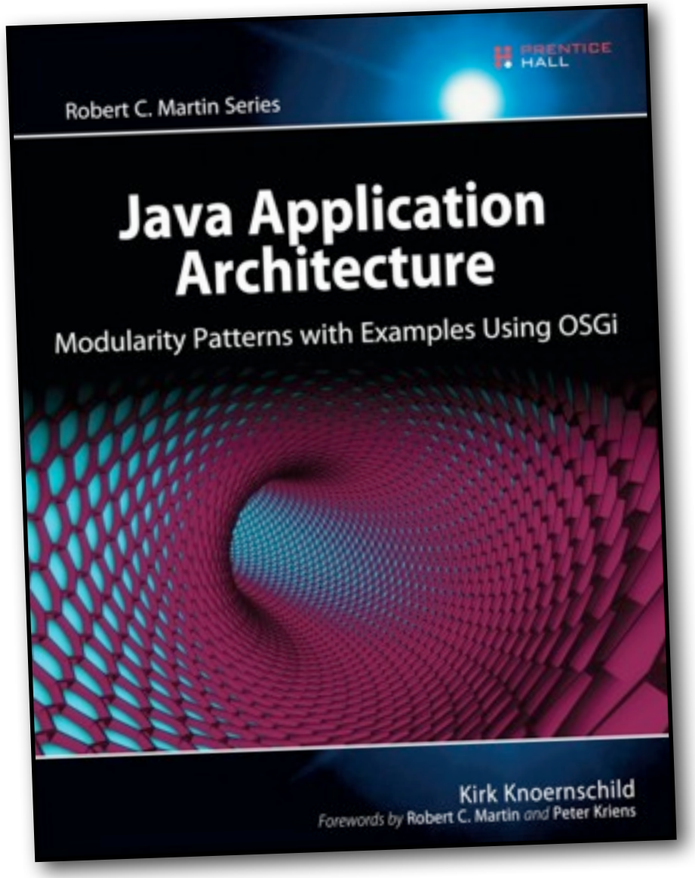
Wrap up

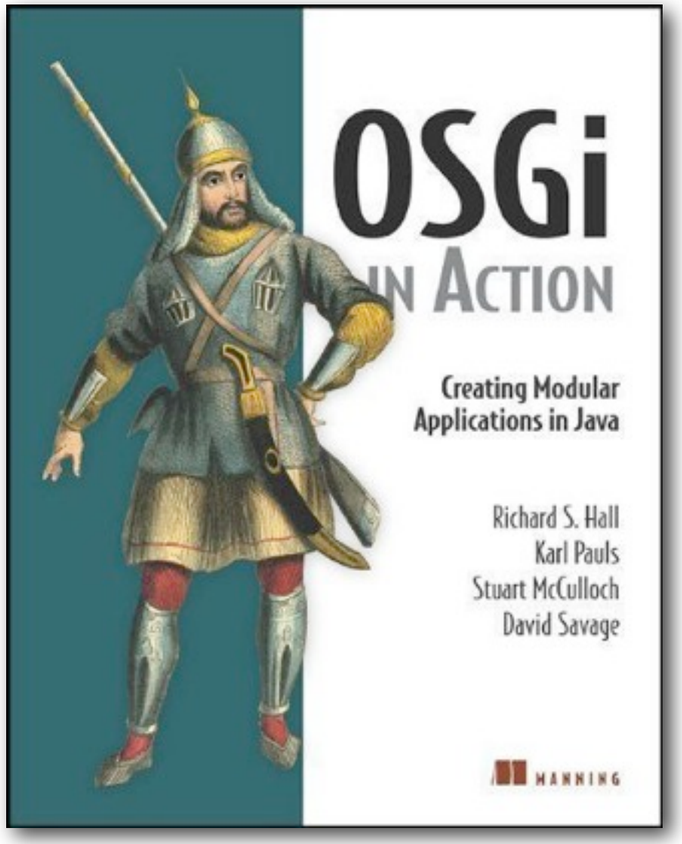


What have we learned?

- You've seen the future of enterprise development ;-)
- Why modularity is important (in the cloud)
- Practical solution for doing modularity now

Recommended reading





Cloud provisioning

http://ace.apache.org/

Cloud OSGi services

http://www.amdatu.org/

Bert Ertman

bert.ertman@luminis.eu

@BertErtman

Eclipse OSGi plugin

http://bndtools.org/

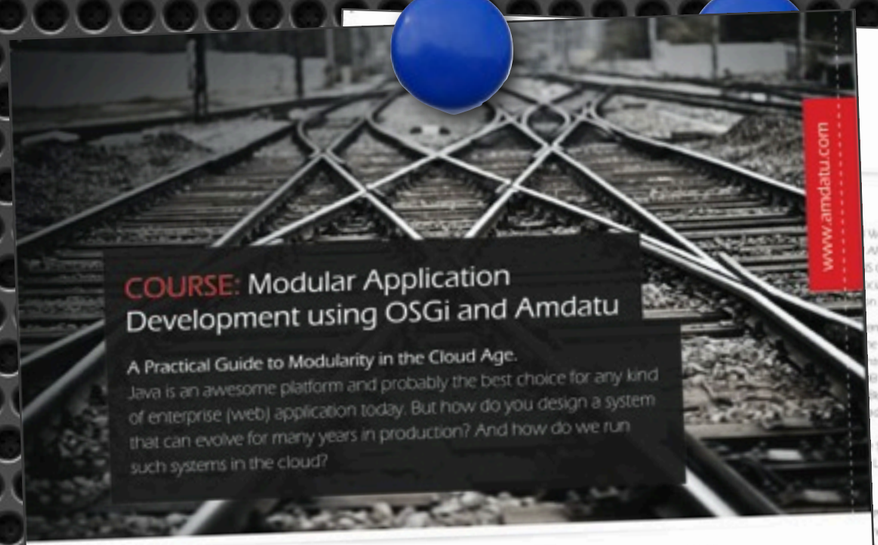
That's us

http://luminis.eu/

Paul Bakker

paul.bakker@luminis.eu

@pbakker



COURSE: Modular Application Development using OSGi and Amdatu

A Practical Guide to Modularity in the Cloud Age.

Java is an awesome platform and probably the best choice for any kind of enterprise (web) application today. But how do you design a system that can evolve for many years in production? And how do we run such systems in the cloud?

Designing a system that can evolve without creating a maintenance nightmare is far from trivial and there is no silver bullet to do this correctly. A modular architecture will help a lot to achieve this goal. Modularity forces separation of concerns which enables you to replace parts of a system without breaking others. The only mature modularity approach for Java is OSGi. OSGi is a framework that enables low-level modularity and services, but it is not an application framework like Java EE, i.e. you still need APIs to create web applications, use transactions, access data sources etc. Without these APIs you will have a hard time to actually build applications. Unfortunately OSGi and Java enterprise technologies did not interoperate well in the past, it was very challenging to mix both. But what if we want modularity in our architecture, but also the ease-of-use of familiar high level Java (enterprise) APIs? The next question is how to deploy such applications in the cloud. Of course you could use one of the existing PaaS providers, but these basically deploy your old, monolithic WAR/EAR in the cloud. This doesn't fit modularity very well. How can you update just parts of a running application without breaking the rest of it?

This lab-intensive, five day, course teaches you how to benefit from a modular programming model meanwhile dealing with these complex issues. It will present a practical and proven solution for building modular Java enterprise applications that can be deployed into a (public/private) cloud. The course will introduce OSGi, the Amdatu Open Source Cloud OSGi Components, and Apache ACE, an Open Source Provisioning Platform. We will not only focus on technical features, but also on architectural aspects like non-functional requirements, dependency management, design considerations, and testability.

amdatu

Course Contents

Web Services
API Design Considerations
S Clients
cal Clients
n Libs

nce Strategies
e Cloud
troduction
R
Robstores
DB

Stores (Memcached)
Libs

Provisioning
ity
undles
e
Auto-scaling
ets

s
Management and Semantic Versioning
Fr
tions Bundle Repositories
bloading and Dynamic Imports
les From Plain JARs
mming Strategies

ncepts to create flexible business

Operating System and IDE of their liking

ple as a Java Champion and is an active
and modularity

aker on conferences all over the world
, Bndtools, and JBoss Forge.

ise applications with Java.