OSGi vs Spaghetti
Part II
The Enterprise Strikes Back

Holly Cummins

IBM

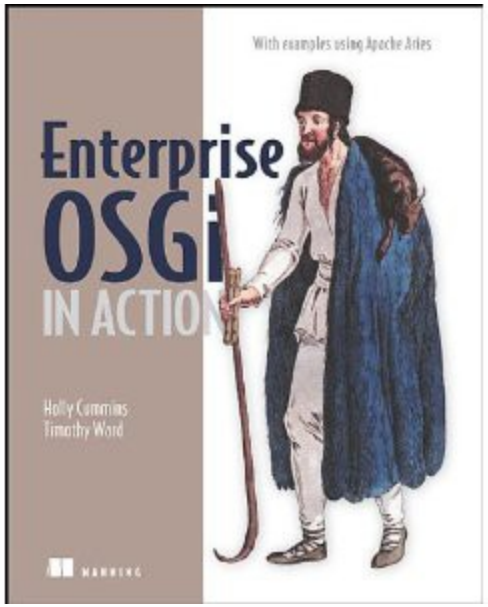
@holly_cummins

Who am I?

- @holly_cummins
- Developer with IBM
  - WebSphere Liberty Profile
  - OSGi
  - Java performance
- Apache Aries committer

Enterprise OSGi in Action

- I'm one of the authors :)
  - Early access available at http://www.manning.com/cummins



How did we get here?

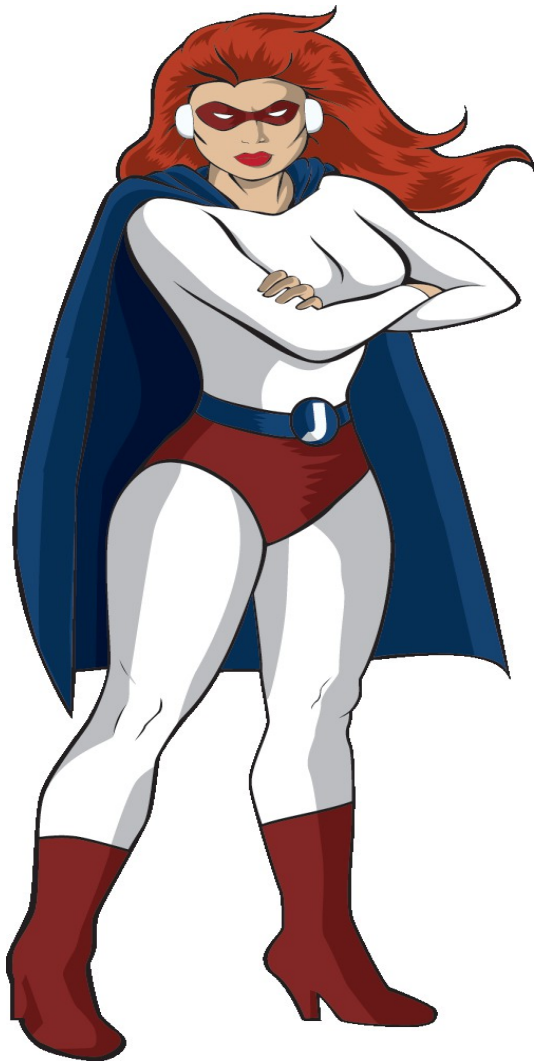
A New Hope

- A long time ago, in a galaxy far far away ...
- (well, maybe fifteen years ago) ...
- Java EE was born
- It was really good at the web …
- … and data …
- … but didn't have much to say on modularity …
- … or dynamism …

Another New Hope

- A long time ago, in a galaxy far far away ...
- (well, maybe fifteen years ago) ...
- OSGi was born
- It was really good at modularity …
- … and dynamism …
- … but didn't have much to say on data …
- … or the web …

As a user, you had a choice.



What is "Enterprise OSGi" and why do I need it?

- OSGi is a mature technology with a broad range of adoption
  - Eclipse
  - Embedded systems
  - Home automation
  - Java EE Application Servers
  -
- Enterprise OSGi is much newer (First release 2010)
  - Primary focus to improve OSGi's support for enterprise tools
  - Widely available in Open Source and Commercial servers

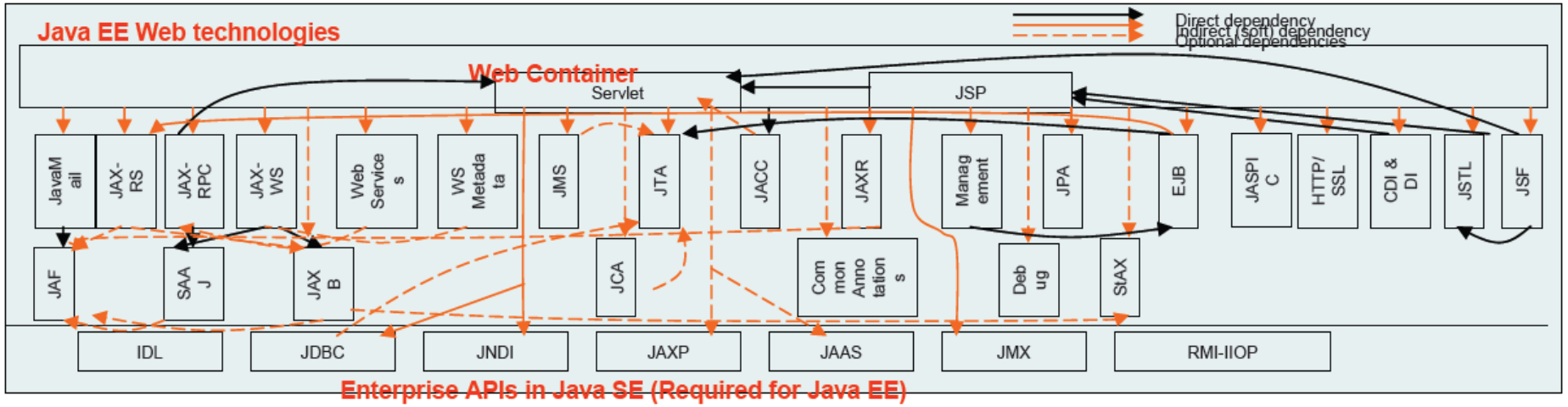
What is "Enterprise OSGi" and why do I need it? (2)

- Enterprise OSGi is just OSGi applied to "Enterprise" Applications
  - OSGi Web applications
  - Using databases from an OSGi framework
  - Managed Transactions for OSGi bundles
  - Remoting Services…

- But isn't this what Java EE is for?
  - Why is OSGi helpful?

What is "Enterprise OSGi" and why do I need it? (3)

- OSGi enables modularity
  - OSGi *enforces* modularity
  - Big systems are hard to maintain and understand because of the relationships between components:



- Big applications are just as complicated as servers (and usually have more external dependencies!)

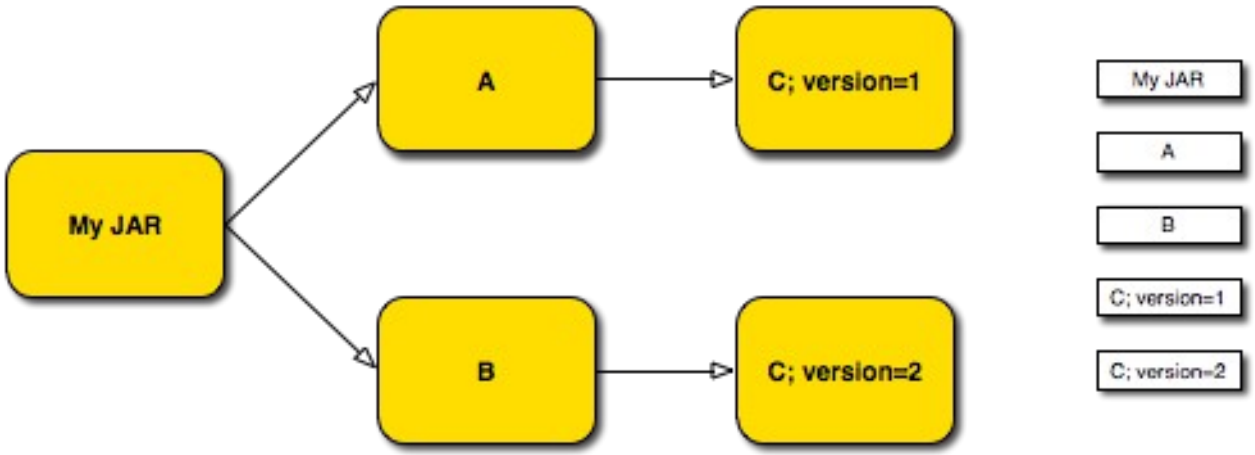
Why I need OSGi in my WAR

- Big WAR files are often bigger than the servers they run on
  - TomCat core is available to download as a 7 MB zip file!
- Why are these WARs so big?
  - Do I really need all those libraries?
  - Why does Maven insist on putting the Java Mail API in WEB-INF/lib?
- Why can't my WAR be more dynamic?
  - Do I really need to restart the application to add a new Payment Type?

Why else do I need OSGi in my WAR?

- Have you ever found that you need to use a library class, but it depends on another version of a library you were already using?
  - Java has a flat classpath, so you can only have one version of the class
  - If you can't change the code you can be forced into using brittle combinations of point releases

- OSGi has a classloader graph:
  - It all just works!



How can I use Enterprise OSGi in my WARs?

How can I use Enterprise OSGi in my WARs?

- Lots of Application runtimes offer support for OSGi applications
  - –WebSphere, Glassfish, Jboss, Geronimo, Karaf, Virgo, Aries…
- Most require little more than packaging your application as OSGi bundles
  - –A JAR with a special manifest

How can I use Enterprise OSGi in my WARs? (2)

- But we don't want to run a JAR, we want to run a WAR
  - WARs and JARs are similar, with different internal structure

	JAR	WAR
Manifest file	META-INF/Manifest.mf	META-INF/Manifest.mf
Web Descriptor	X	WEB-INF/web.xml
Classes location	/	/WEB-INF/classes
Nested libraries	X	/WEB-INF/lib
Non-classpath resources	X	/

How can I use Enterprise OSGi in my WARs? (3)

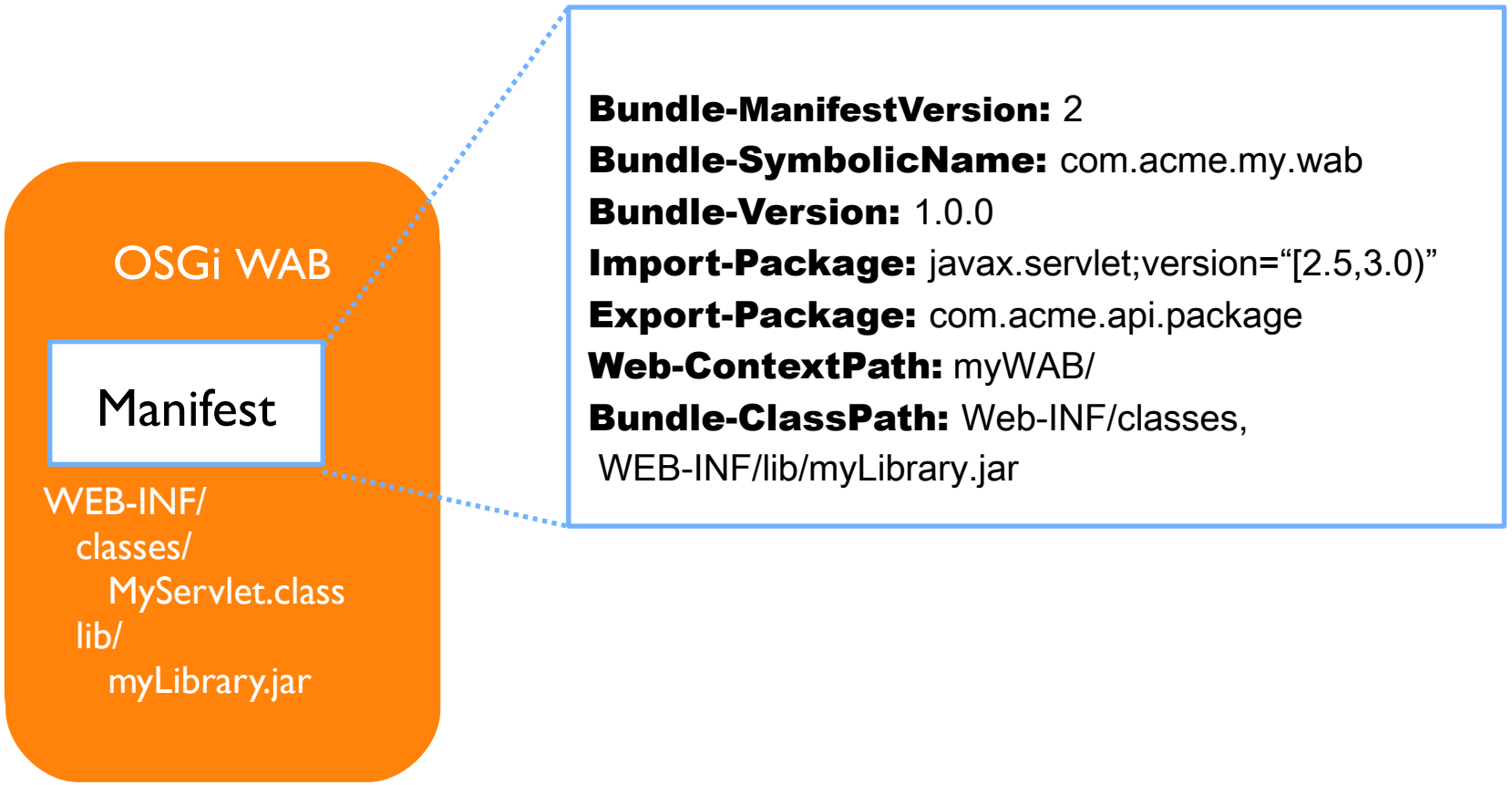
- The lack of WAR support in OSGi was a serious limitation
  - Moving to the OSGi HTTP Service is non-trivial!

- The OSGi Enterprise Expert Group created the OSGi Web Applications Specification
  - Simple support for Web Application Bundles (WABs)
  - Re-use existing Web deployment descriptors
  - It must be possible to be a valid WAR and a WAB at the same time!

Structure of a Web Application Bundle

- First and foremost a Web Application Bundle is an OSGi bundle
  - It must specify the required OSGi metadata
- Secondly it must include the Web-ContextPath header
  - This defines the context root for the WAB
- Thirdly, if you want to use the standard WAR classpath
  - Bundle-ClassPath: WEB-INF/classes, WEB-INF/lib/myJar.jar…

Structure of a Web Application Bundle (2)



So what does migration give me?

- If you just put the relevant OSGi metadata in your WAR's manifest you have migrated your WAR to OSGi!
  - Your WAR is still the same size as before (approximately)
  - You aren't using any of OSGi's features
- Remember the Import-Package header?
  - Using this allows you to move JARs out of your WAR
    - You can also move out the dependencies that JAR pulled in!
- Version conflicts between higher order dependencies disappear!
  - Deployment is faster (particularly annotation scanning!)

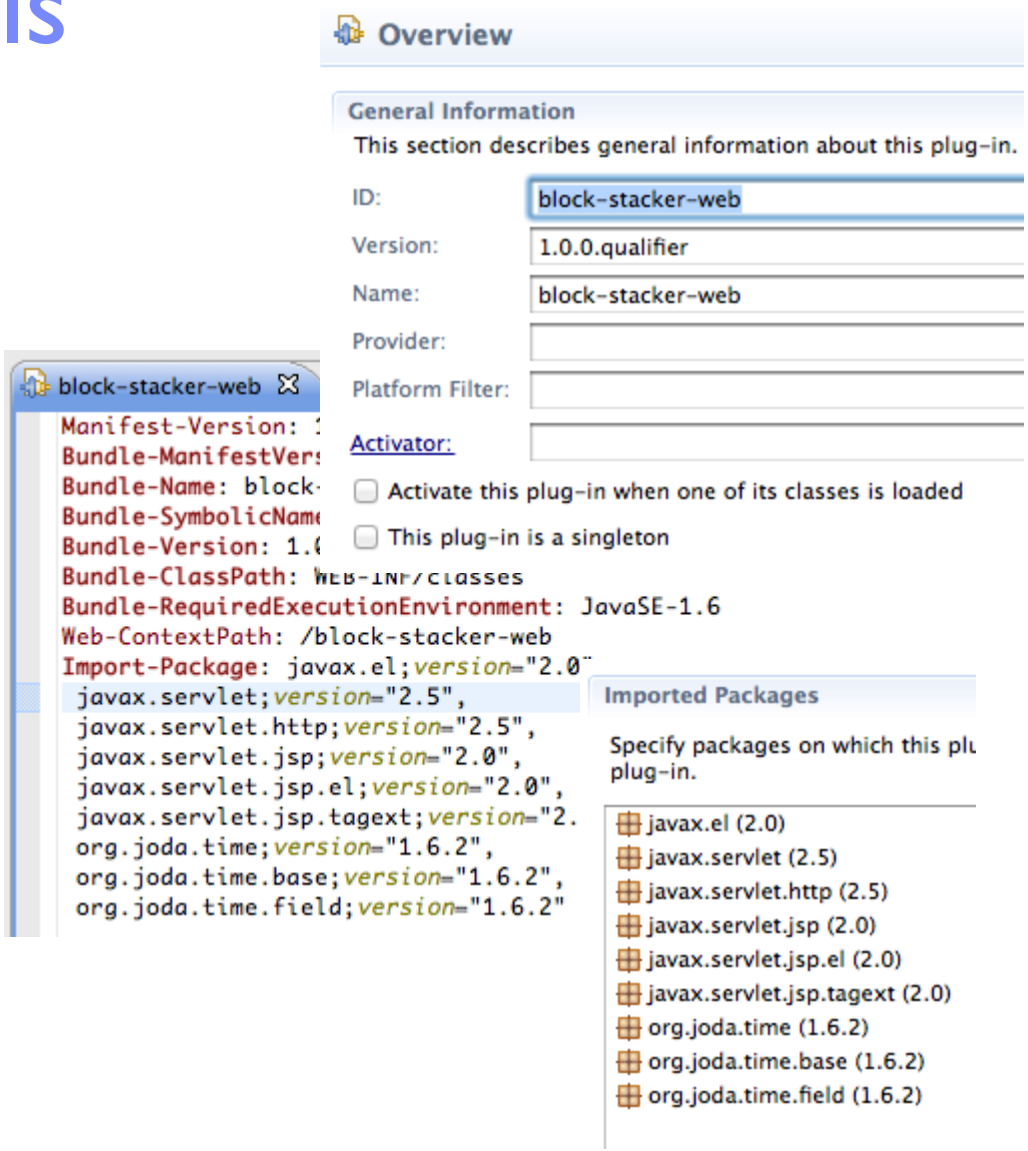
How do I develop and build a WAB?

Before you start: The Great Manifest Debate

- We all like tools
  - Tools can *help* with the manifest
    - "Manifest-first" approach
  - Tools can *write* the manifest
    - "Code-first" approach

Manifest-first tools

- Develop
  - Eclipse PDE
- Build
  - Maven Tycho
  - Plain-old-Ant



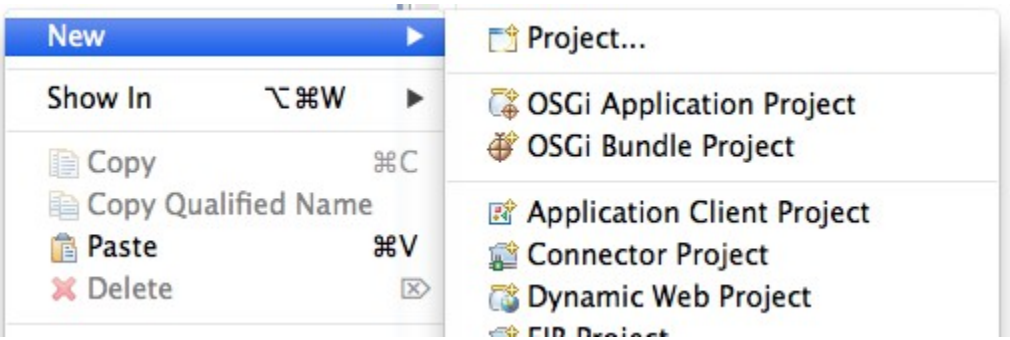
Code-first tools

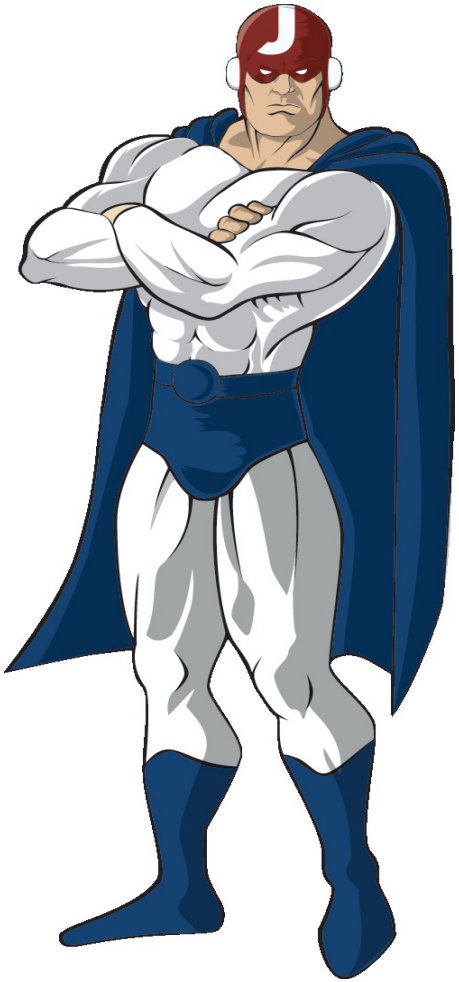
- Develop:
  - Whatever you like!
    - (Or Eclipse and BndTools if you're feeling fancy :))
- Build
  - Maven build plugin

```
<packaging>bundle</packaging>
```

Enterprise-OSGi tools

- Eclipse Libra
- IBM's OSGi Application Development Tools
  - (What we'll be using today)





Handling third-party libraries

Options for dependencies

- Your jar is already a bundle!
  - Remember, a bundle *is* a jar
- Use a newer version, which is a bundle
- Find a wrapped bundle Somewhere
- Consider an alternative
- Wrap your own bundle
- Embed the jar into your bundle

Finding OSGi-fied bundles

- SpringSource Enterprise Bundle Repository
- Maven Central
  - Same bundle, different group id

Wrapping bundles

- Use bnd

```
java -jar biz.aQute.bnd.jar wrap some.jar
```

  - Creates “some.bar"
  - All classes externally visible
  - All dependencies optional
  - You may wish to adjust these defaults

Embedding jars

- Eliminates many classloading visibility problems
- Your bundle and third-party library share a class-space
- Not the most-space-efficient option

One more thing ...

The magic of OSGi services

- Elegant solution to the factory pattern
- Look up service based on interface
- 100% Dynamic
- But …
  - Don't use the API directly
  - Inject dependencies
    - Blueprint
    - Declarative Services

Demo

Maybe

With luck

With lots of luck

Summary

Things to remember

- OSGi isn't as hard as you've been led to believe!
  - But it isn't magic either
  - You can still have a badly-modularised application, even with OSGi

Useful Resources

- The OSGi specifications are available at http://www.osgi.org
- Apache Aries for implementations http://aries.apache.org/
- Manning have several good OSGi books
  - *Enterprise OSGi in Action* – Get up and running with Web Apps, Transactions, JPA, Remoting, IDEs and build tools
    - Use the code eosgi37 at http://www.manning.com/cummins for 37% off!
  - *OSGi in Action* – Great examples and coverage of core OSGi and compendium services
  - *OSGi in Depth* – Detailed coverage of architectural patterns for OSGi
- OSGi Articles available at http://www.developerworks.com

Common problems

Reflection

- No bytecode dependency
- No auto-detection of import by bnd

Properties files

- The same rules apply to properties files as classes
- Export pseudo-package for properties

```
Export-Package: some.props.folder
```

- Import pseudo-package to read them

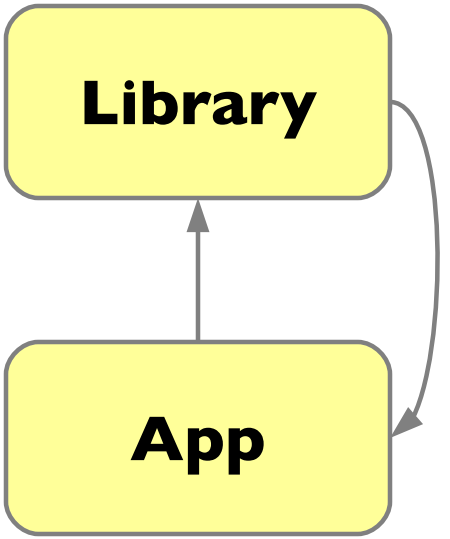
```
Import-Package: some.props.folder
```

Late binding

- What if imported package isn't known at compile-time?
- Use `DynamicImport-Package`

Thread Context Classloader

- Allows cross-classloader classloading
- Works around one-way classloader visibility



Thread Context Classloader

- Sometimes *assumes* one-way classloader visibility
  - A TCCL probably cannot load your internals!