Everything can be a bundle
Automatically repairing pre-OSGi code

...or “How to run jEdit as a bundle”

Erik Wistrand, wistrand@makewave.com

makewave

What's the problem?

- Lots of Java code has been written taking stuff for granted
  - Just one classloader
  - System.exit() is OK to call
  - A file system is present
  - A static main() method is used for startup
  - ...

...this leads to classloading problems, premature exit of the entire framework and various internal bugs

Classloading problems

Old, non-OSGi-aware libraries can easily be wrapped as an OSGi bundle

Sounds easy, just create a manifest file and export/import the necessary stuff

...but they may **still** run into classloading problems

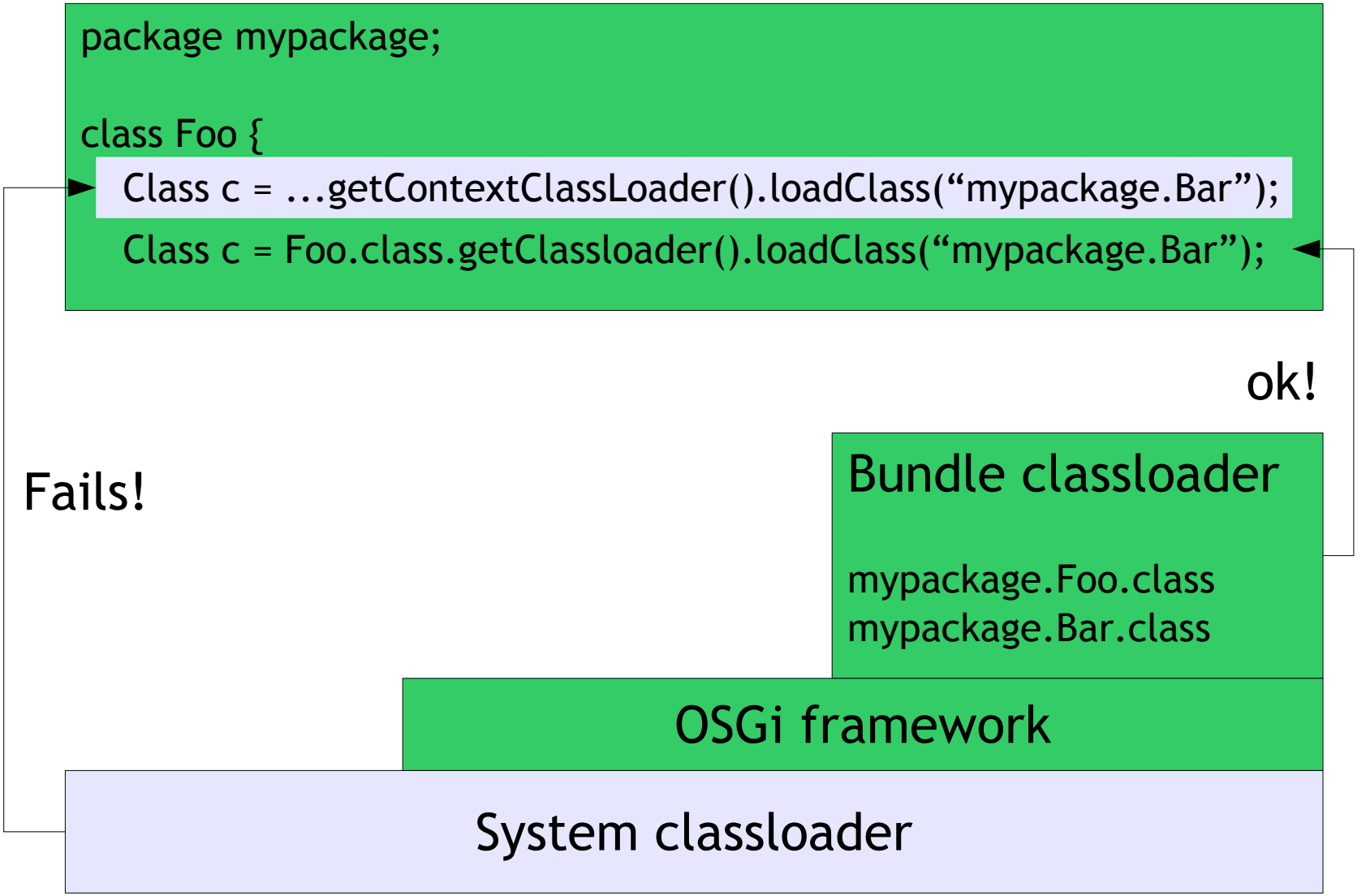
```
ClassNotFoundException: Cannot find com.acme.ICustomer
```

Why ClassNotFoundException

For the OSGi import/export mechanism to work, the bundle **must** use the Bundle classloader

However, many libraries uses the system classloader or the thread's classloader

- Works great in standalone apps
- Does **not work at all** in typical OSGi settings


package mypackage;
class Foo {
Class c = ...getContextClassLoader().loadClass("mypackage.Bar");
Class c = Foo.class.getClassloader().loadClass("mypackage.Bar");
ok!
Fails!
Bundle classloader
mypackage.Foo.class
mypackage.Bar.class
OSGi framework
System classloader

What can be done?

The Knopflerfish OSGi framework contains code that can

- Start jar files with only a static main method
- Automatically import/export packages
- Patch code that uses bad methods regarding classloading etc.

This session will focus on **class patching**

Common workarounds to fix classloading

- Modifying the library source to be OSGi-aware and recompile
  - Impractical
  - License issues
- Wrapping each library call in code that sets the thread's context class loader
  - Tricky to find all cases
  - Boilerplate code is a Bad Thing
- Put all classes on the system class path!
  - Really just hurts

A new approach - automatic byte code modification

Since all bundle classes are loaded via the OSGi framework, the framework can **modify** the bytecode on the fly to use the correct classloader

1. Load original byte code
2. Find occurances of wrong method calls
3. Replace with call to right method
4. Use the modified class

(5. Profit!)

The Knopflerfish implementation

- Uses the ASM byte-code manipulation library
- A configuration file can specify which bundles/classes should be modified
- Bundles are automatically modified as they are loaded

Yes, this is a poor man's **aspect oriented** framework

btw, LDAP filters are the best thing since sliced bread. Yes, they are!

Demo - run jEdit as a bundle

http://www.knopflerfish.org/econ2008/knopflerfish-econ2008.zip

Example patch

Let's patch all library calls to

```
Class.forName(String name,
              boolean init,
              ClassLoader cl)
```

First, write the wrapper method

```
public static
 Class forName3Wrapper(String name,
                        boolean initialize,
                        ClassLoader cl,
                        long        bid,
                        Object      context) throws
 ClassNotFoundException {
 return ... // use bundle classloader instead
}
```

Next, find signature strings

Find the string signature of the method to be patched

```
java/lang/Class.forName(Ljava/lang/String;ZLjava/lang/ClassLoader;)Ljava/lang/Class;
```

Find the string signature or the static wrapper method that should replace the above call

```
org/knopflerfish/framework/ClassPatcherWrappers.forName3Wrapper
```

Create config file

```
patch.1.from= java/lang/Class.\
  forName(Ljava/lang/String;ZLjava/lang/ClassLoader;)\
  Ljava/lang/Class;
patch.1.to= org/knopflerfish/framework/ClassPatcherWrappers.\
  forName3Wrapper
patch.1.static=true
```

...and launch

```
java \
 -Dorg.knopflerfish.framework.patch=true \
 -Dorg.knopflerfish.framework.patch.configurl=\
   file:patches.props \
 -jar framework.jar:asm-3.1.jar \
org.knopflerfish.framework.Main
```