Do you really get class loaders?

Jevgeni Kabanov
Founder & CTO of ZeroTurnaround

What do I do?

- Creator and core developer of JRebel
  - *JRebel maps you[illegible] proje[illegible] workspace directly to a runni[illegible] s the changes you [illegible], then intelligen[illegible] pplication.*
- "How to elimi[illegible] e[illegible] loys and gain 3-7 weeks a year?"
  - Lightning talk later today, find out more
  - 3-7 weeks a year based on survey results

JFokus attendees get a free license! www.jrebel.com/jfokus

ZEROTURNAROUND

To create JRebel we…

- Hooked into class loading on the JVM level
- Integrated with the class loading mechanism in more than 10 different servers
- Solved hundreds of issues connected to class loading
- Learned a lot more about class loaders than we wanted to ☺

Overview

- Basics
  - What is class loading?
  - How was it meant to work?
- Problems and solutions
- How do class loaders leak?
- *OSGi, Spring dm, JBoss and others???*
- Conclusions

BASICS

Class loader API

```
public abstract class ClassLoader {
  public Class loadClass(String name);
  protected Class defineClass(byte[] b);

  public URL getResource(String name);
  public Enumeration getResources(String name);

  public ClassLoader getParent()
}
```

Class loading

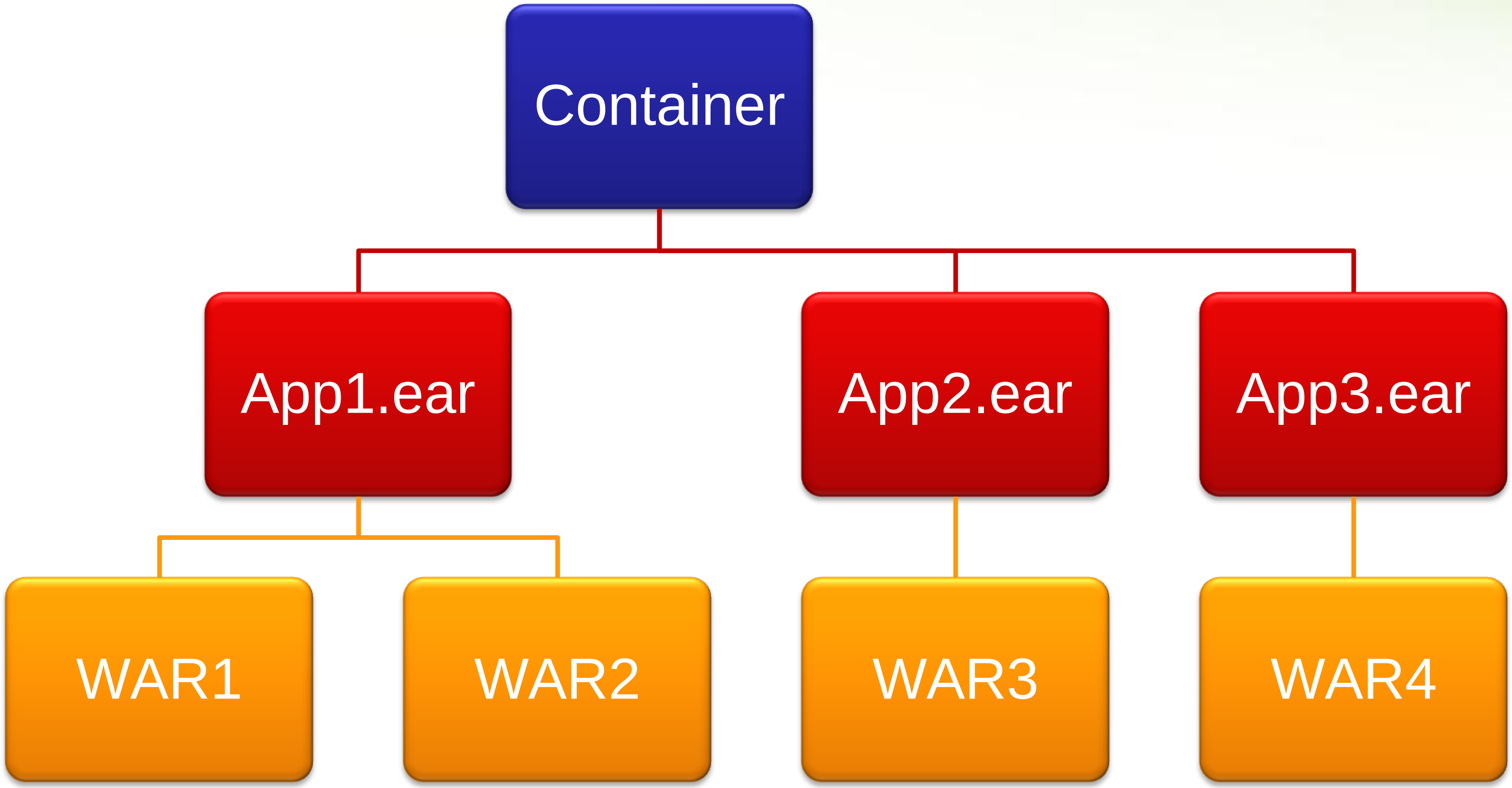
```
public class A {
  public void doSmth() {
    B b = new B();
    b.doSmthElse();
  }
}
```

Causes a call to
`A.class.getClassLoader().loadClass("B");`

Delegation

- Class loaders have a *parent* class loader
- The parent is usually consulted first
  - Avoids loading same class several times
  - However in a Java EE web module local classes are searched first
- In Java EE each WAR module of an EAR gets its own class loader
  - This allows separate namespaces for applications in same container

Java EE Delegation



PROBLEMS AND SOLUTIONS

No class found

- Variants
  - ClassNotFoundException
  - ClassNoDefFoundException
- Helpful
  - IDE class lookup (Ctrl+Shift+T in Eclipse)
  - *find *.jar -exec jar -tf '{}' \;* | *grep MyClass*
  - URLClassLoader.getUrls()
  - Container specific logs

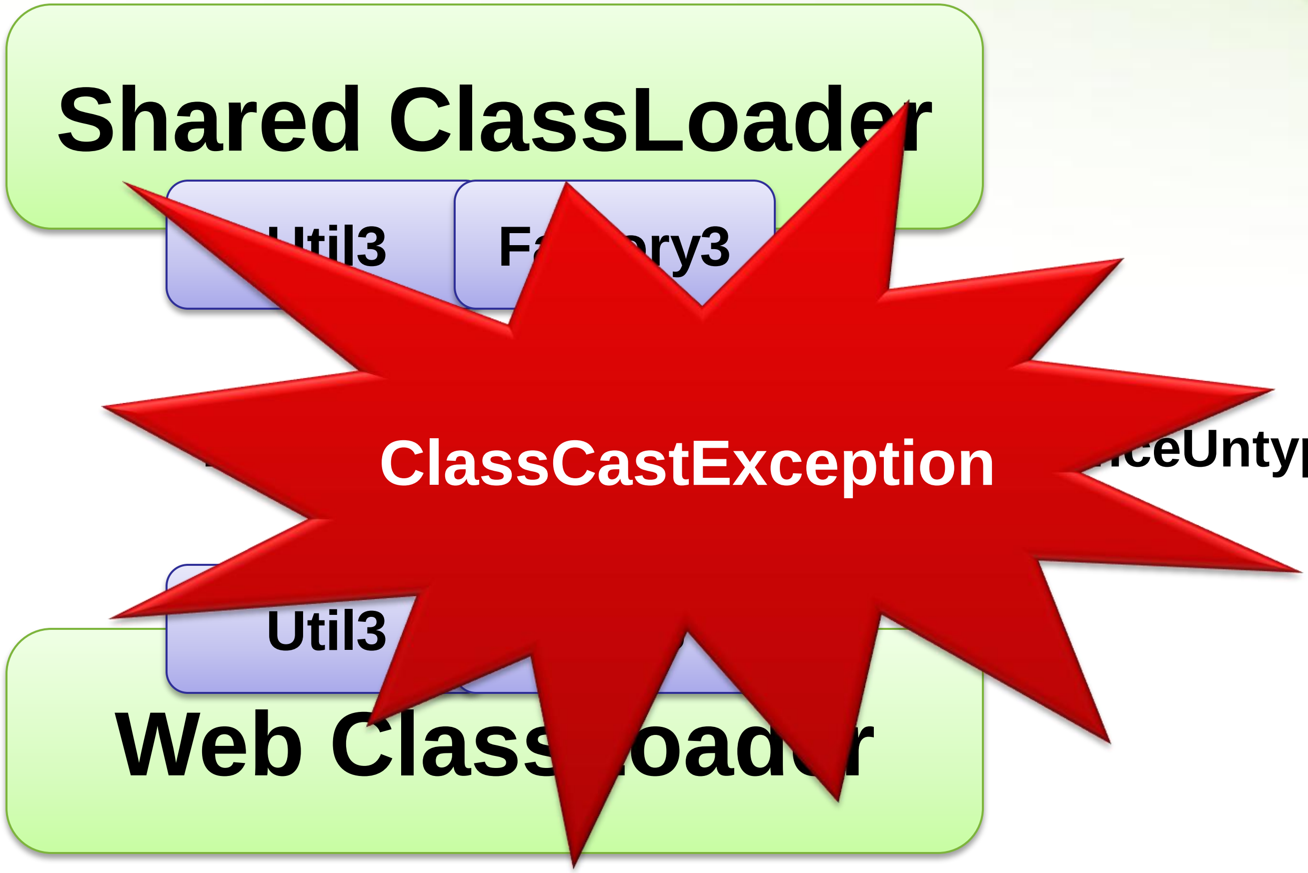
Wrong class found

- Variants
  - IncompatibleClassChangeError
    - AbstractMethodError
    - NoSuch(Method|Field)FoundError
  - ClassCastException, IllegalAccessError
- Helpful
  - -verbose:class
  - ClassLoader.getResource()
  - *javap -private MyClass*

More than one class found

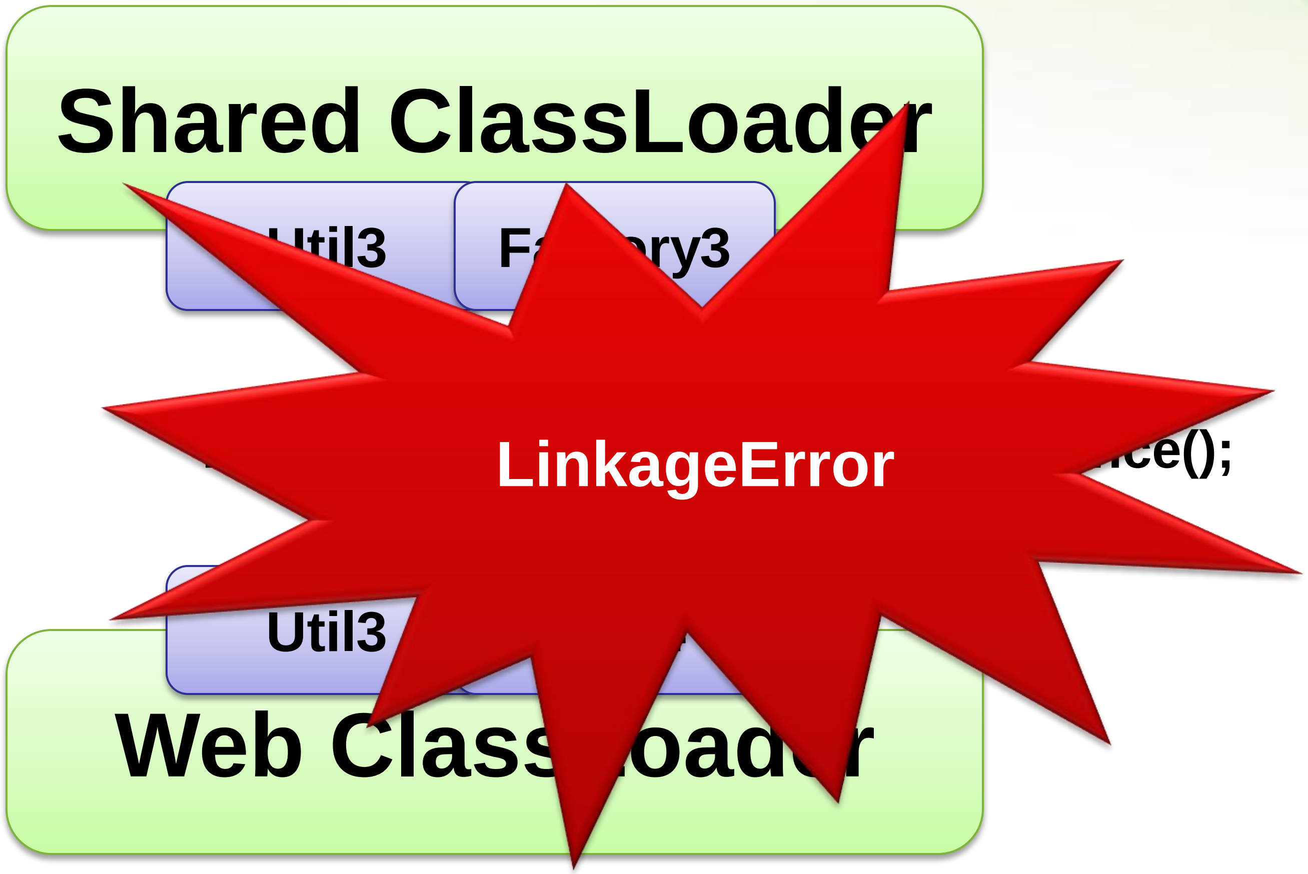
- Variants
  - LinkageError (class loading constraints violated)
  - ClassCastException, IllegalAccessError
- Helpful
  - -verbose:class
  - ClassLoader.getResource()

More than one class found



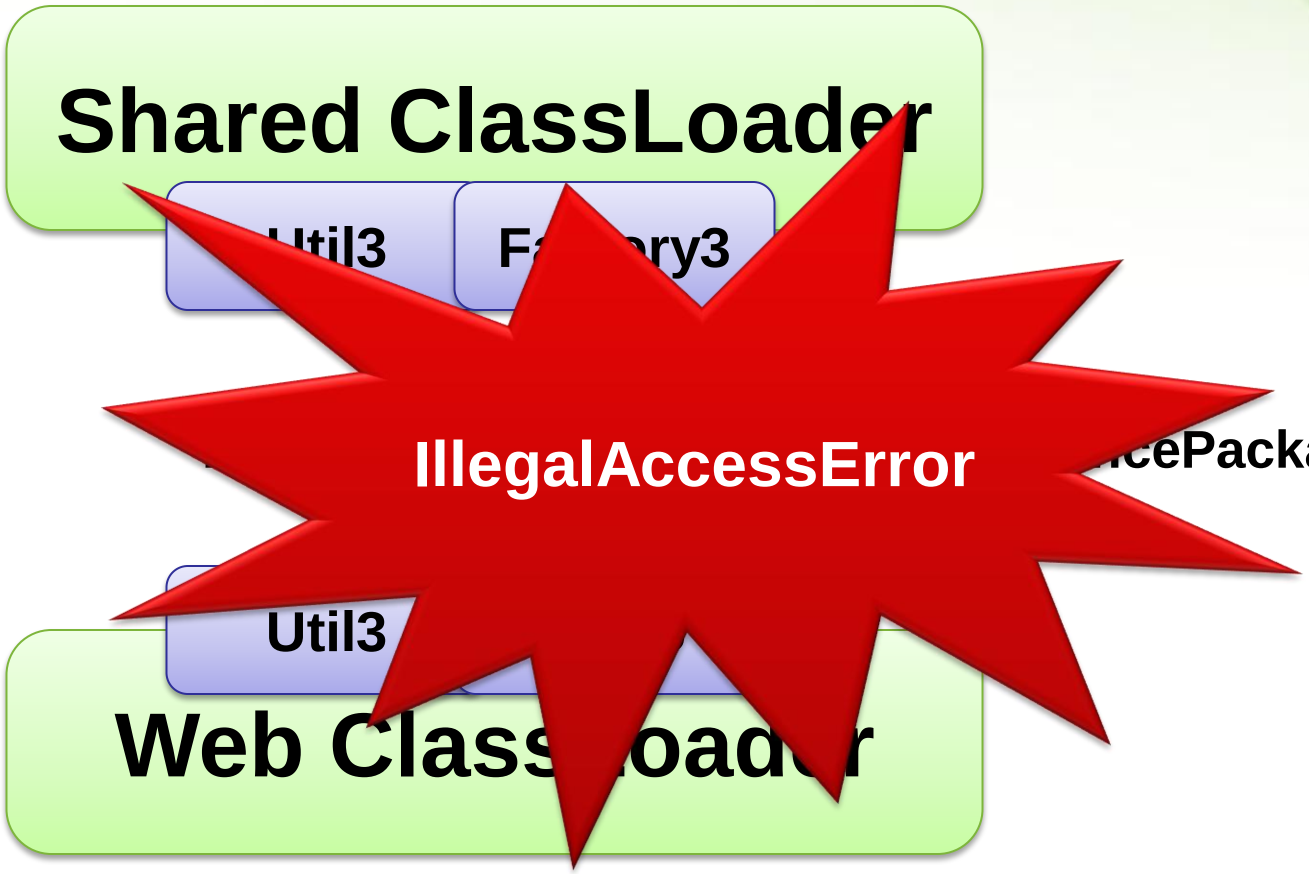
```
Util3 u = (Util3) Factory3.instanceUntyped();
```

More than one class found



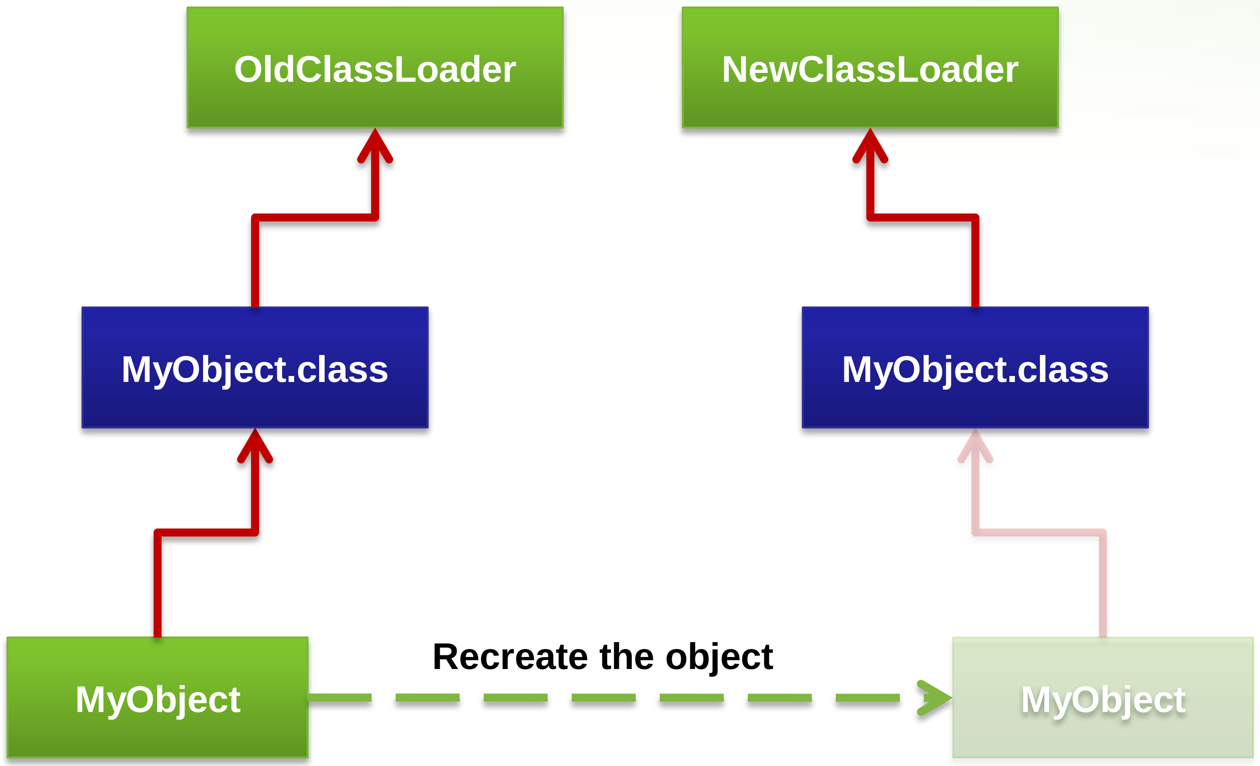
Factory3.instance().sayHello();

More than one class found

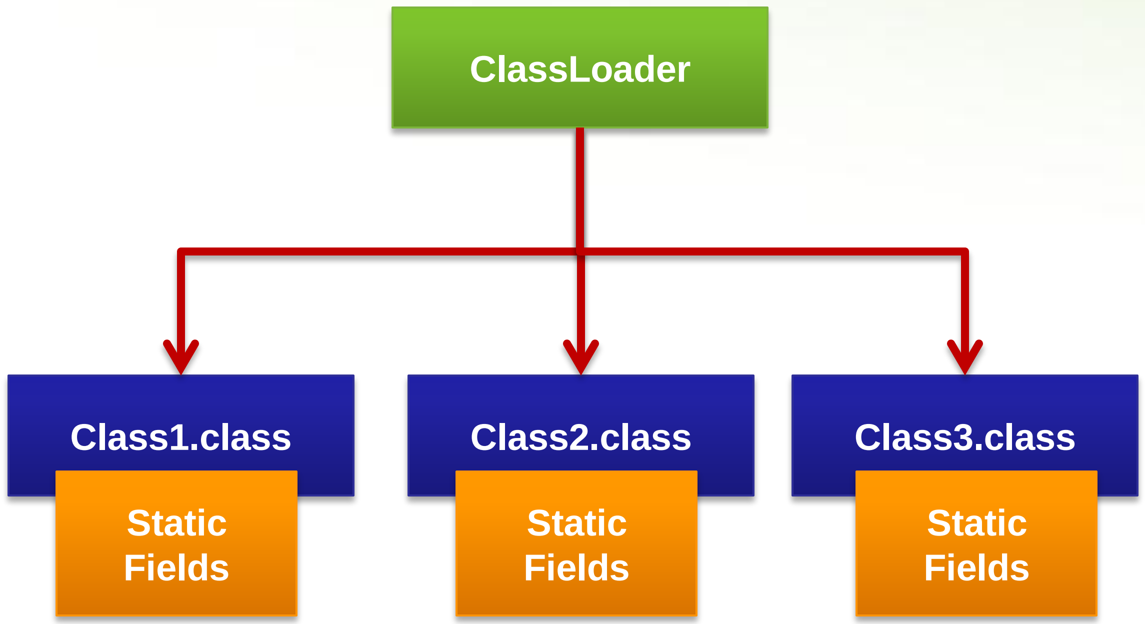


Util3 u = (Util3) Factory3.instancePackage();

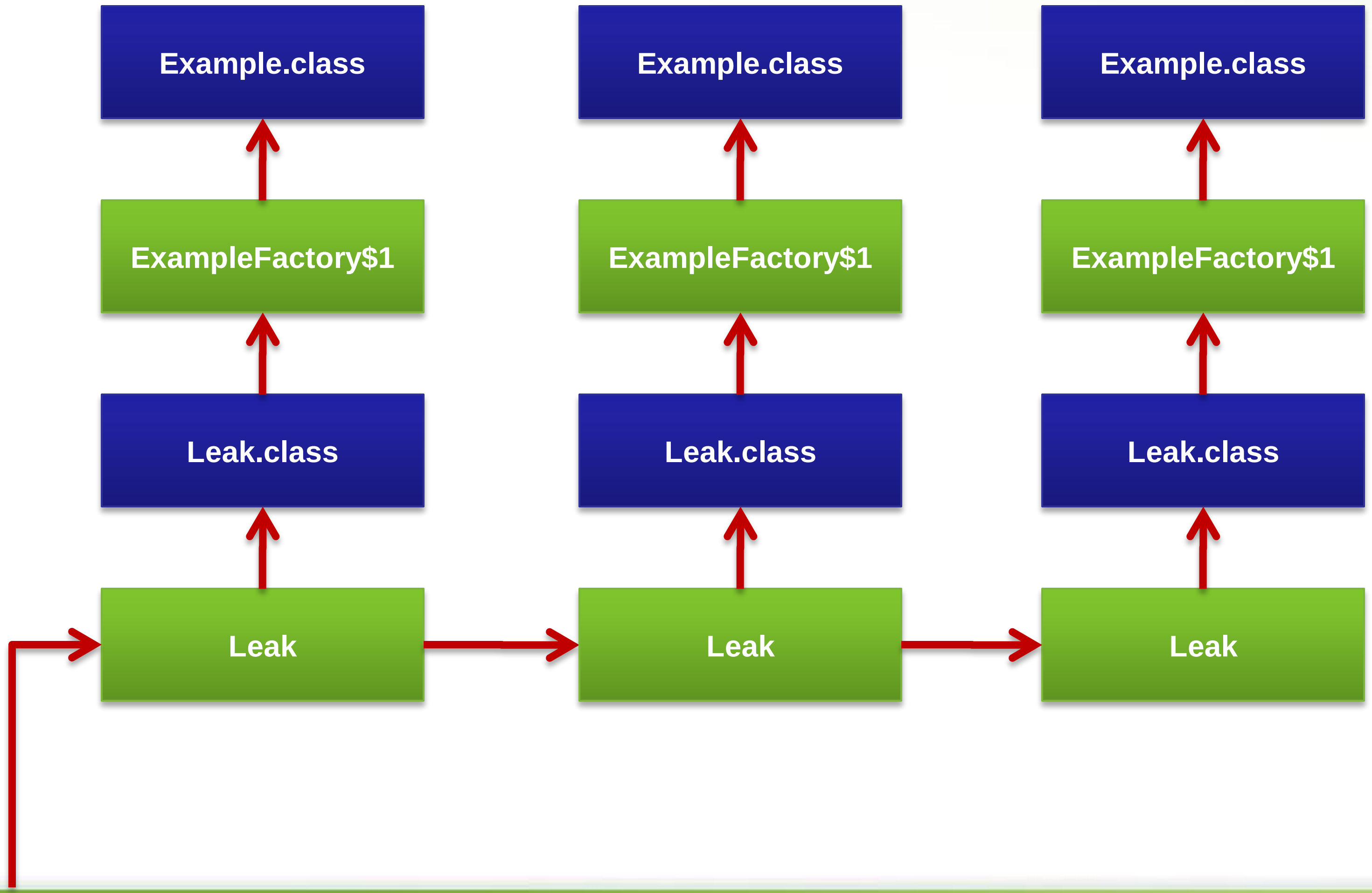
Reloading an Object



Leaking ClassLoaders



Leaking ClassLoaders



STATE OF THE ART

Hierarchy is not enough?

- Isolation
  - Different versions of the same library
- Performance
  - Class lookup is very slow
- Restricted
  - Why siblings can’t see each other’s classes?

- OSGi, JBoss, NetBeans and others implement a different system

The Modern Way

- Each JAR has own class loader
- All class loaders are siblings, with one central repository
- Each JAR explicitly declares
  - Packages it exports
  - Packages it imports
- Repository can find relevant class loaders by package

Modern Filtering

```
class MClassLoader extends ClassLoader {
  // Initialized during startup from imports
  Set<String> imps;

  public Class loadClass(String name) {
    String pkg = name.substring(0,
      name.lastIndexOf('.'));

    if (!imps.contains(pkg))
      return null;

    return repository.loadClass(name);
  }
}
```

Modern Lookup

```
class MRepository {
  // Initialized during startup from exports
  Map<String,List<MClassLoader>> exps;

  public Class loadClass(String name) {
    String pkg = name.substring(0,
      name.lastIndexOf('.'));
    for (MClassLoader cl : exps.get(pkg)) {
      Class result = cl.loadLocalClass(name);
      if (result != null) return result;
    }
    return null;
  }
}
```

Troubleshooting

- The same tricks also work with Modern class loading systems
  - ClassLoader.getResource();
  - -verbose:class
- Often can be supplemented with custom tools
- Need to think in terms of export/import in addition to classpath
  - Looking at the pseudocode can help

Problems

- Too restrictive
  - Import is a one-way street
  - If you want to use Hibernate, you import it, but it cannot access your classes
- Easy to leak
  - Any references between class loaders are leaks waiting to happen
- Deadlocks
  - JVM enforces a global lock on loadClass()

Conclusions

- The trick of troubleshooting class loaders is **understanding** how they work :)
- Modern systems add **a level of complexity** on top of an abstraction that nobody gets to begin with
- When redeploying or reloading classes **leaking is easy** and leads to OOM
- We need **better** tools to troubleshoot class loaders!