Java ClassLoader

Charles Sharp
Object Computing, Inc.

Saint Louis Java User Group

Facts &Terms

CLASSLOADERS...

✳ Are responsible for loading classes (no, really...) *and* resources.

✳ Come in only two types, bootstrap and user-defined (more on these in a minute).

✳ Subclass the abstract class `ClassLoader` (or one of its subclasses).

✳ User-defined have a parent ClassLoader (probably)

Terms

✴ Initiating ClassLoader (next slide)

✴ Defining ClassLoader (next slide)

✴ Primordial ClassLoader -- The bootstrap class loader

✴ Class Name -- fully-qualified (as of 1.5, the binary name)

✴ Class (or interface) -- Class name and the Defining ClassLoader (next slide)

Defining and Initiating

A class loader L may create Class or Interface C by defining it directly or by delegating to another class loader. If L creates C directly, we say that L defines C or, equivalently, that L is the defining loader of C.

When one class loader delegates to another class loader, the loader that initiates the loading is not necessarily the same loader that completes the loading and defines the class. If L creates C, either by defining it directly or by delegation, we say that L initiates loading of C or, equivalently, that L is an initiating loader of C.

At run time, a class or interface is determined not by its name alone, but by a pair: its fully qualified name and its defining class loader. Each such class or interface belongs to a single runtime package. The runtime package of a class or interface is determined by the package name and defining class loader of the class or interface.

http://java.sun.com/docs/books/jvms/second_edition/html/ConstantPool.doc.html#51579

Creating a Class

* If the name denotes a nonarray class:

    * If the referencing class was loaded by the bootstrap, then the bootstrap class loader initiates the loading.

    * If the referencing class was loaded by a user-defined loader, then that same user-defined class loader initiates loading of the new class.

    * The JVM creates array classes directly.

http://java.sun.com/docs/books/jvms/second_edition/html/ConstantPool.doc.html#51579

ClassLoaders

The Standard Three

All Java applications have at least three ClassLoaders:

✳ bootstrap class loader

✳ extension class loader

✳ system class loader

bootstrap class loader

bootstrap

✳ Loads the core Java libraries (rt.jar, core.jar, server.jar, classes.jar, whatever...)

✳ Does **not** validate the classes -- they're trusted

✳ Has no parent (bootstrap loaded classes receive null on calls to getClassLoader())

✳ Loads the classes specified in the command line "non-standard" option "-Xbootclasspath"

-Xbootclasspath

-Xbootclasspath:bootclasspath

Specify a colon-separated list of directories, JAR archives, and ZIP archives to search for boot class files. These are used in place of the boot class files included in the Java 2 SDK. Note: Applications that use this option for the purpose of overriding a class in rt.jar should not be deployed as doing so would contravene the Java 2 Runtime Environment binary code license.

http://java.sun.com/javase/6/docs/technotes/tools/solaris/java.html

-Xbootclasspath/p

-Xbootclasspath/p:path

Specify a colon-separated path of directories, JAR archives, and ZIP archives to prepend in front of the default bootstrap class path. Note: Applications that use this option for the purpose of overriding a class in rt.jar should not be deployed as doing so would contravene the Java 2 Runtime Environment binary code license.

http://java.sun.com/javase/6/docs/technotes/tools/solaris/java.html

-Xbootclasspath/a

-Xbootclasspath/a:path

Specify a colon-separated path of directories, JAR archives, and ZIP archives to append to the default bootstrap class path.

http://java.sun.com/javase/6/docs/technotes/tools/solaris/java.html

extensions class loader

extensions

* Loaded at JVM start time by the bootstrap class loader

* A user-defined ClassLoader (subclass of ClassLoader).

* Loads the extensions classes (next slide)

* Does **not** use the class path. (What happens when one of the classes in a jre/lib/ext JAR file needs to load a class that is not a core Java or extension class?)

Shared Extensions in Java 6

✳ Extension JAR files may be placed in a location that is independent of any particular JRE, so that extensions can be shared by all JREs that are installed on a system.

✳ Prior to Java 6, the value of java.ext.dirs referred to a single directory, but as of Java 6 it is a list of directories (like CLASSPATH)

✳ The first element of the path is always the lib/ext directory of the JRE. The second element is a directory outside of the JRE.

✳ Allows extension JAR files to be installed once and used by several JREs installed on that system.

✳ The location varies depending on the operating system:
   * SolarisTM Operating System: /usr/jdk/packages/lib/ext
   * Linux: /usr/java/packages/lib/ext
   * Microsoft Windows: %SystemRoot%\Sun\Java\lib\ext

✳ An installed extension placed in one of the above directories extends the platform of each of the JREs (Java 6 or later) on that system.

system class loader

system (aka application)

✳ Loads the application's classes

✳ Serves as parent for all the classes loaded by the application unless it is replaced by a different class loader.

✳ Locates classes in the directories and JAR files on the class path -- java.class.path system property. (Set by either the CLASSPATH environment variable or the -classpath command-line option.)

Implementation

* bootstrap class loader is implemented in the JVM (typically in C); *not* specified by ClassLoader specifications

* In Sun's Java implementation, the extension and system class loaders are implemented in Java. Both are instances of the URLClassLoader class.
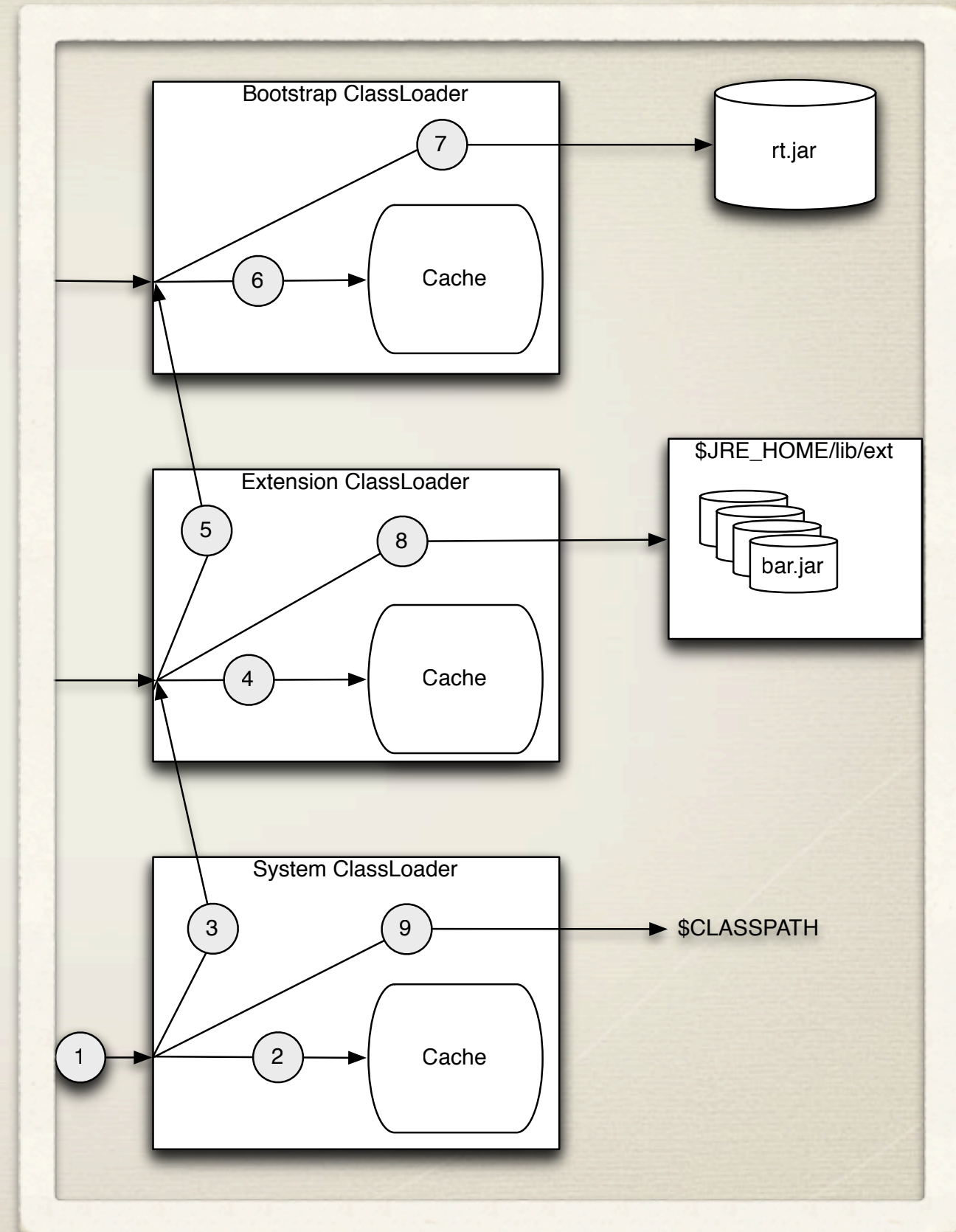
How ClassLoaders Load

ClassLoader.loadClass()

When a class needs to be loaded (is referenced):

1. If the class has already been loaded, return it.

2. Otherwise, delegate the search to the parent class loader

3. If the parent doesn't find it (shown when it throws ClassNotFoundException), call findClass() to find (load, define, and resolve) the class

ClassLoader Process

1. findClass (loadClass) is invoked

2. Current(system) checks cache

3. Not loaded; delegates to parent(extension)

4. extension checks cache

5. Not loaded; delegates to parent(bootstrap)

6. bootstrap checks cache

7. Not loaded; checks "Core Java" jars; not found; throws ClassNotFoundException

8. extension catches Exception; checks extension jars; not found; throws ClassNotFoundException

9. system catches Exception; chases CLASSPATH looking for class; finds it (surely!) and reads in the bytes

10. invokes defineClass() for resolution

Bootstrap ClassLoader

7

rt.jar

6 → Cache

$JRE_HOME/lib/ext

Extension ClassLoader

5

8

bar.jar

4 → Cache

System ClassLoader

3

9

$CLASSPATH

1

2 → Cache

But Wait!
There's One More
(ClassLoader) Thing...

What good is it ?

classloader inversion

✓Application code's helper method calls `Class.forName(classNameString)`.

✓Helper method is called from a plugin class.

✓The `classNameString` specifies a class that is contained in the plugin JAR.

http://www.informit.com/articles/article.aspx?p=1187967&seqNum=1
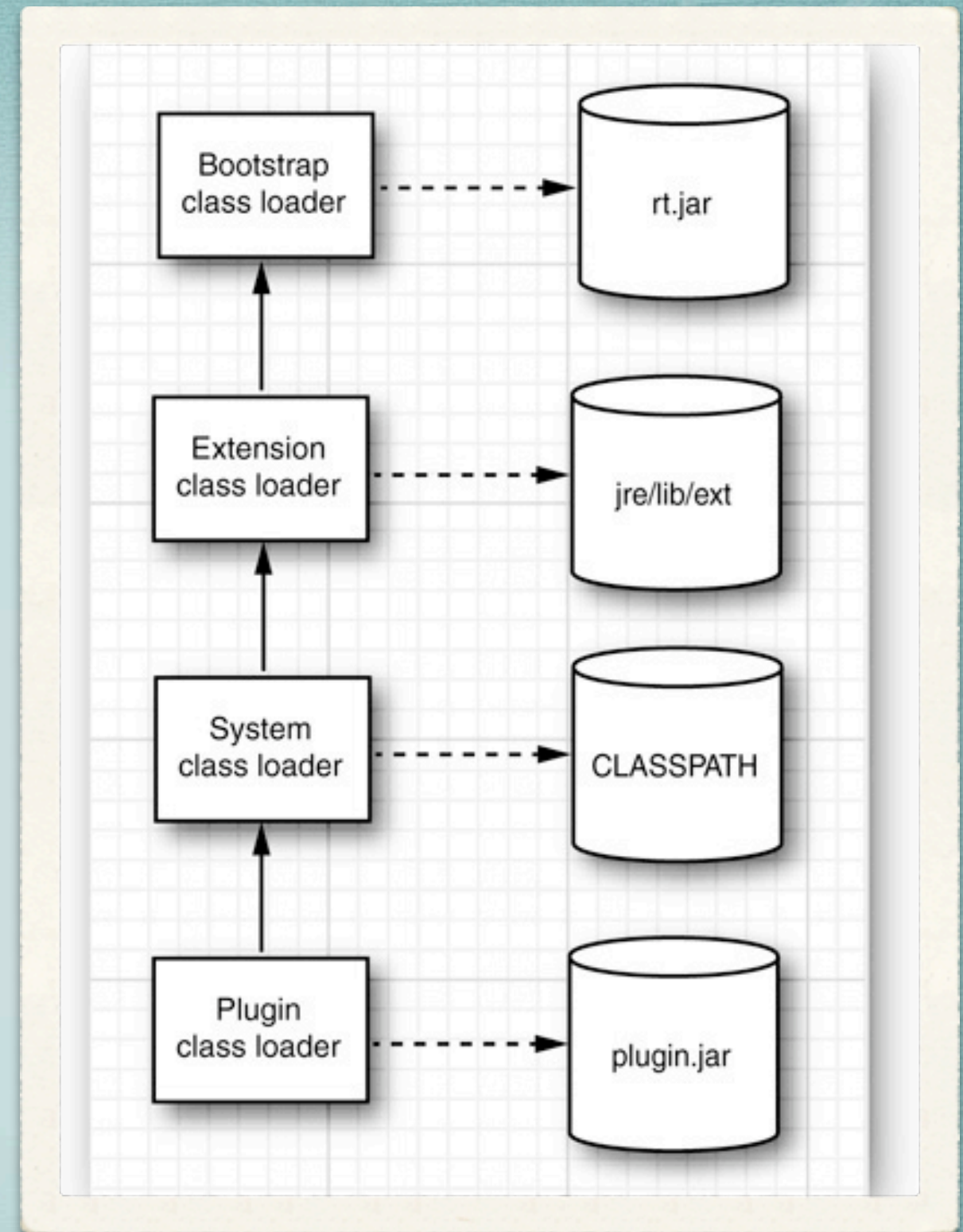
Solution

The helper method needs to use the correct class loader. It can require the class loader as a parameter. Alternatively, it can require that the correct class loader is set as the context class loader of the current thread. This strategy is used by many frameworks (such as the JAXP and JNDI frameworks)

Each thread has a reference to a context class loader. The main thread's context class loader is the system class loader. When a new thread is created, its context class loader is set to the creating thread's context class loader.

http://www.informit.com/articles/article.aspx?p=1187967&seqNum=1

context class loader

context

✳ Made for the world of threads -- each thread has one

✳ Set with:

```
Thread t = Thread.currentThread();
t.setContextClassLoader(loader);
```

✳ Retrieved with:

```
Thread t = Thread.currentThread();
ClassLoader loader = t.getContextClassLoader();
Class<?> cl = loader.loadClass(className);
```

✳ If no context class loader is set, all threads will have the system class loader as their context class loader

Writing Your Own ClassLoader

(and what about those Exceptions?)

Why write a ClassLoader

✳ Allows specialization of the load process:

  ✳ custom class checks (checksums)

  ✳ custom loading techniques (loading from a specialized device.   Or lower-case class names...)

How to write a ClassLoader

＊ Extend ClassLoader (or one of its subclasses)

＊ Override `findClass(String className)`

＊ The `loadClass` method of the ClassLoader superclass takes care of the delegation to the parent and calls the `findClass` method only if the class hasn't already been loaded and if the parent class loader was unable to load the class.

findClass()

1. Load the bytecodes for the class from the local file system or from some other source.

2. Call the `defineClass` method of the ClassLoader superclass to present the bytecodes to the virtual machine.

Exceptions
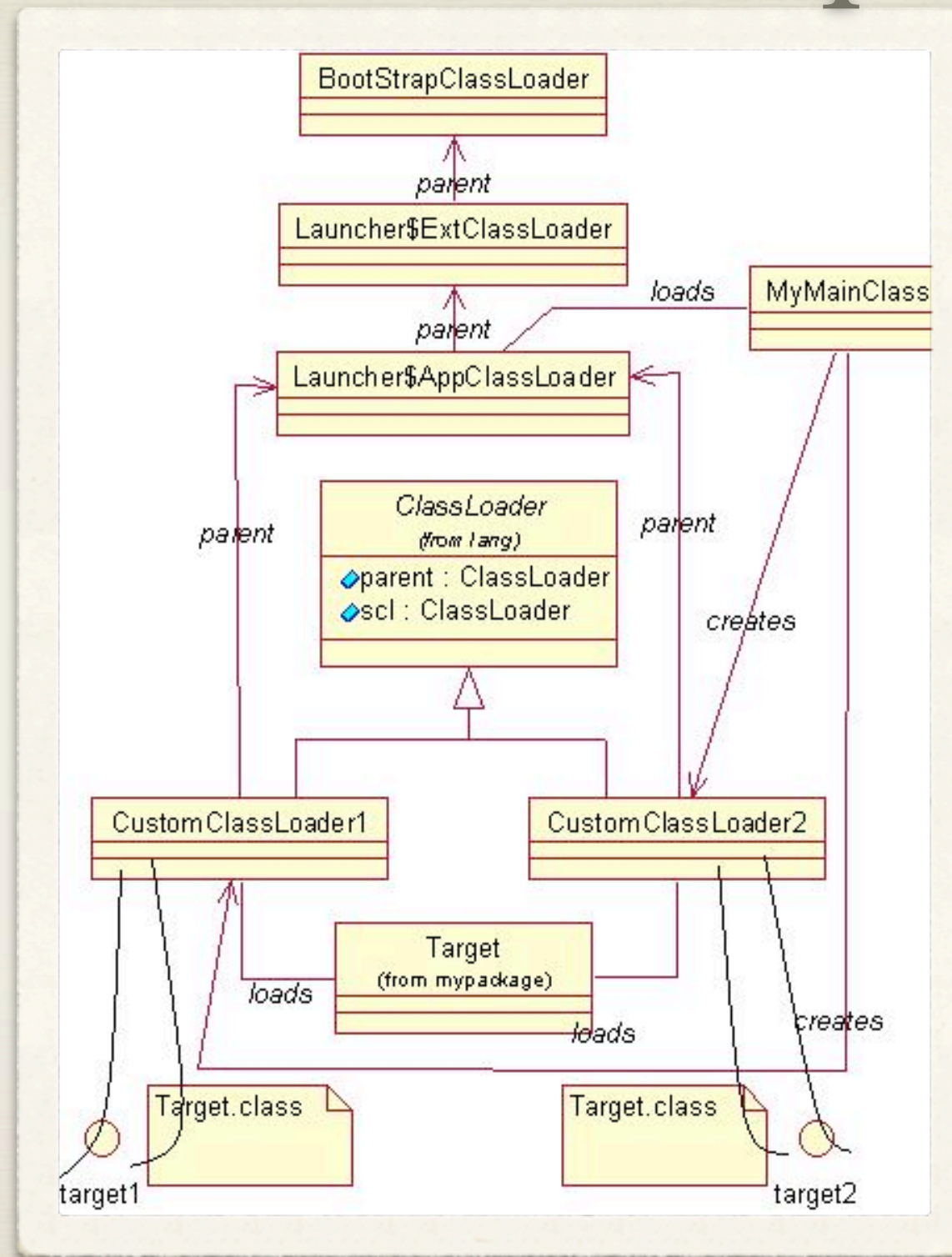
ClassLoader Exceptions

✳ ClassNotFoundException

   ✳ loadClass()

   ✳ findClass()

✳ NoClassDefFound

   ✳ defineClass() -- name in byte array does not match the name in the class

ClassCastException

http://www.onjava.com/pub/a/onjava/2005/01/26/classloading.html

Information Providers

✳ **Find a way out of the ClassLoader maze**, Vladimir Roubtsov, JavaWorld.com, 06/06/03. http://www.javaworld.com/javaworld/javaqa/2003-06/01-qa-0606-load.html

✳ **Understanding Network Class Loaders**, Qusay H. Mahmoud, October 2004. http://java.sun.com/developer/technicalArticles/Networking/classloaders/ ;

✳ **Chapter 5, Loading, Linking, and Initializing**, The Java Virtual Machine Specification, 2nd Ed., Tim Lindholm and Frank Yellin. http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html

✳ **A Look At The Java Class Loader**, Kumar Matcha, Oct 15, 2005. http://www.javalobby.org/java/forums/t18345.html

✳ **Introduction to Java's Architecture**, Bill Venners. http://www.artima.com/insidejvm/ed2/introarchP.html

✳ **Core Java Security: Class Loaders, Security Managers, and Encryption**, Cay S. Horstmann, Gary Cornell, Apr 16, 2008. http://www.informit.com/articles/article.aspx?p=1187967&seqNum=1

✳ **Inside Class Loaders**, Andreas Schaefer, 11/12/2003. http://www.onjava.com/pub/a/onjava/2003/11/12/classloader.html

✳ **Internals of Java Class Loading**, Binildas Christudas, 01/26/2005. http://www.onjava.com/pub/a/onjava/2005/01/26/classloading.html

✳ **Understanding Extension Class Loading**, Java Tutorial. http://java.sun.com/docs/books/tutorial/ext/basics/load.html

✳ **Installed Extensions**, Java Tutorial. http://java.sun.com/docs/books/tutorial/ext/basics/install.html