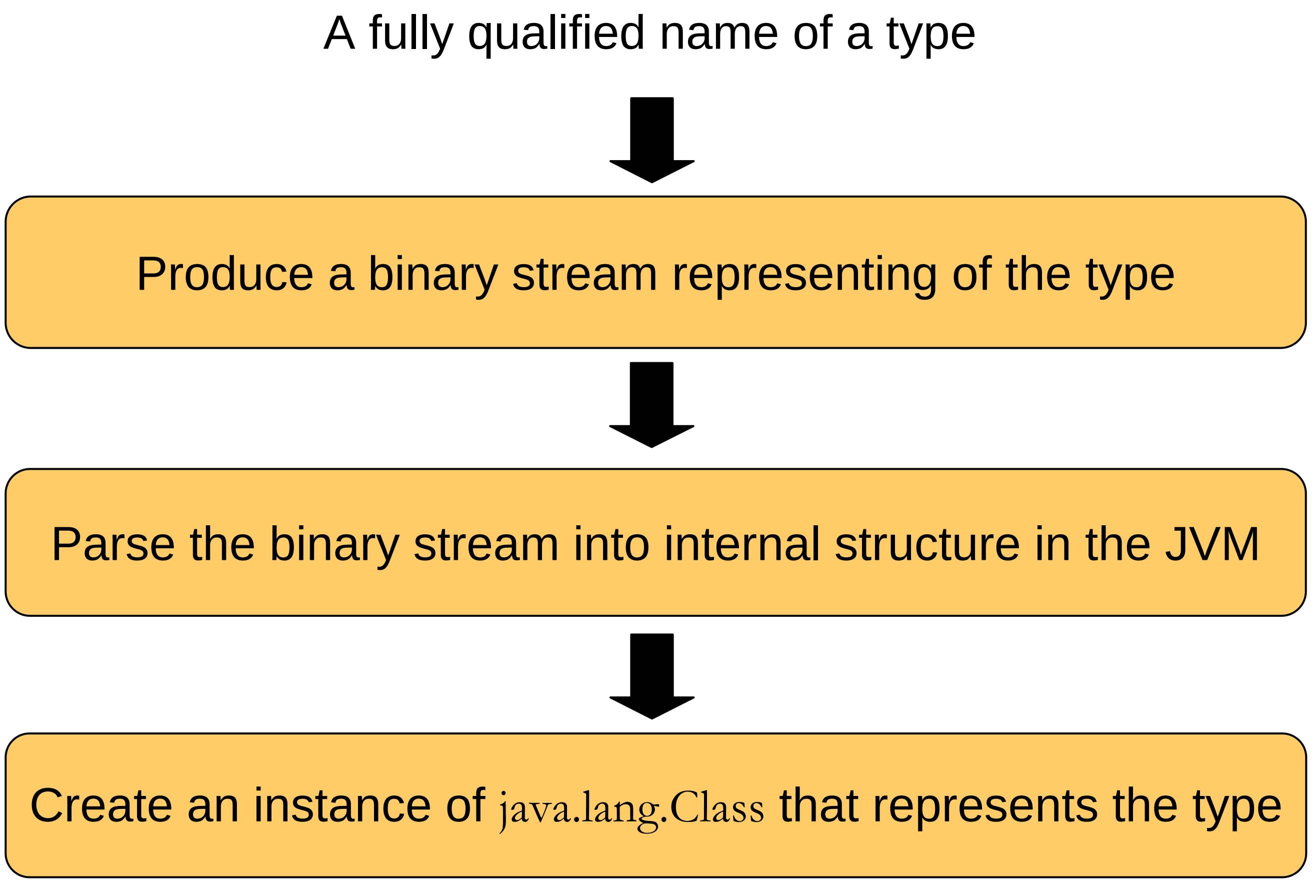
Object-Oriented Programming with Java

Recitation No. 12: Class Loading

Class Loading



A fully qualified name of a type

↓

Produce a binary stream representing of the type

↓

Parse the binary stream into internal structure in the JVM

↓

Create an instance of java.lang.Class that represents the type

Class Loaders

Two types of class loaders:

- The bootstrap (primordial) class loader:
  - an integral part of the JVM
  - loads the core Java classes
- Custom (user-defined) class loaders
  - are subclasses of java.lang.ClassLoader
  - are ordinary Java classes

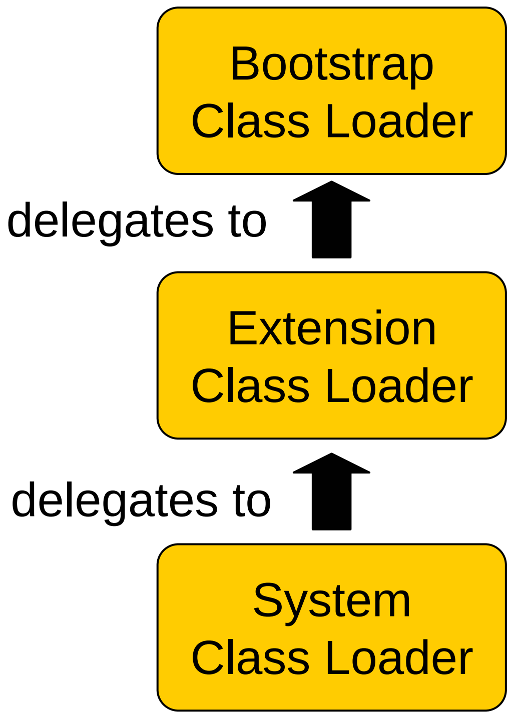
Class Loaders

- Every Class object holds a reference to its class loader
- The Class.getClassLoader() method returns:
  - a ClassLoader object
  - null for representing the bootstrap class loader
  - For arrays, returns the class loader of the element type

Delegation Model

- Class loaders are hierarchically arranged
- The root is the bootstrap class loader
- Each custom class loader has a parent class loader:
  - it is the system class loader by default
  - can be provided as a construction argument
- A custom class loader first delegates the search to its parent class loader

Typical Default Delegation Model



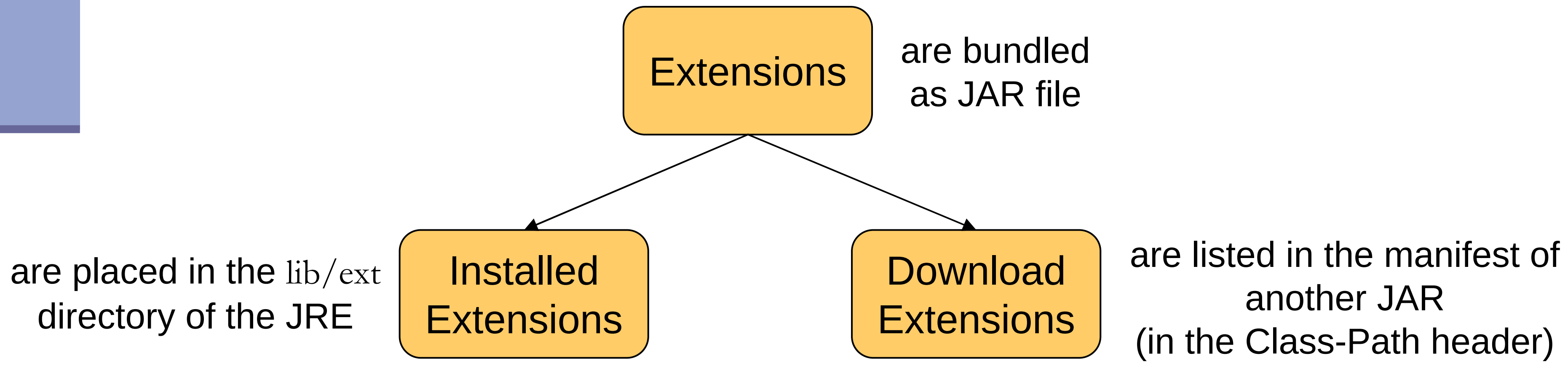
loads core Java classes (java.* etc.)

loads classes in JAR files from extension directories

loads classes from the system class path (ClassLoader.getSystemClassLoader())

The Extension Mechanism

- A standard way to make custom APIs available to all Java applications
- No need to name the extension classes on the class path


Extensions
are bundled as JAR file
Installed Extensions
are placed in the lib/ext directory of the JRE
Download Extensions
are listed in the manifest of another JAR (in the Class-Path header)

Custom Class Loaders

- Extend the java.lang.ClassLoader abstract class
- Main methods of java.lang.ClassLoader:
  - public Class **loadClass**(String name)
  - protected Class **loadClass**(String name, boolean resolve):

    - Invokes findLoadedClass(String)

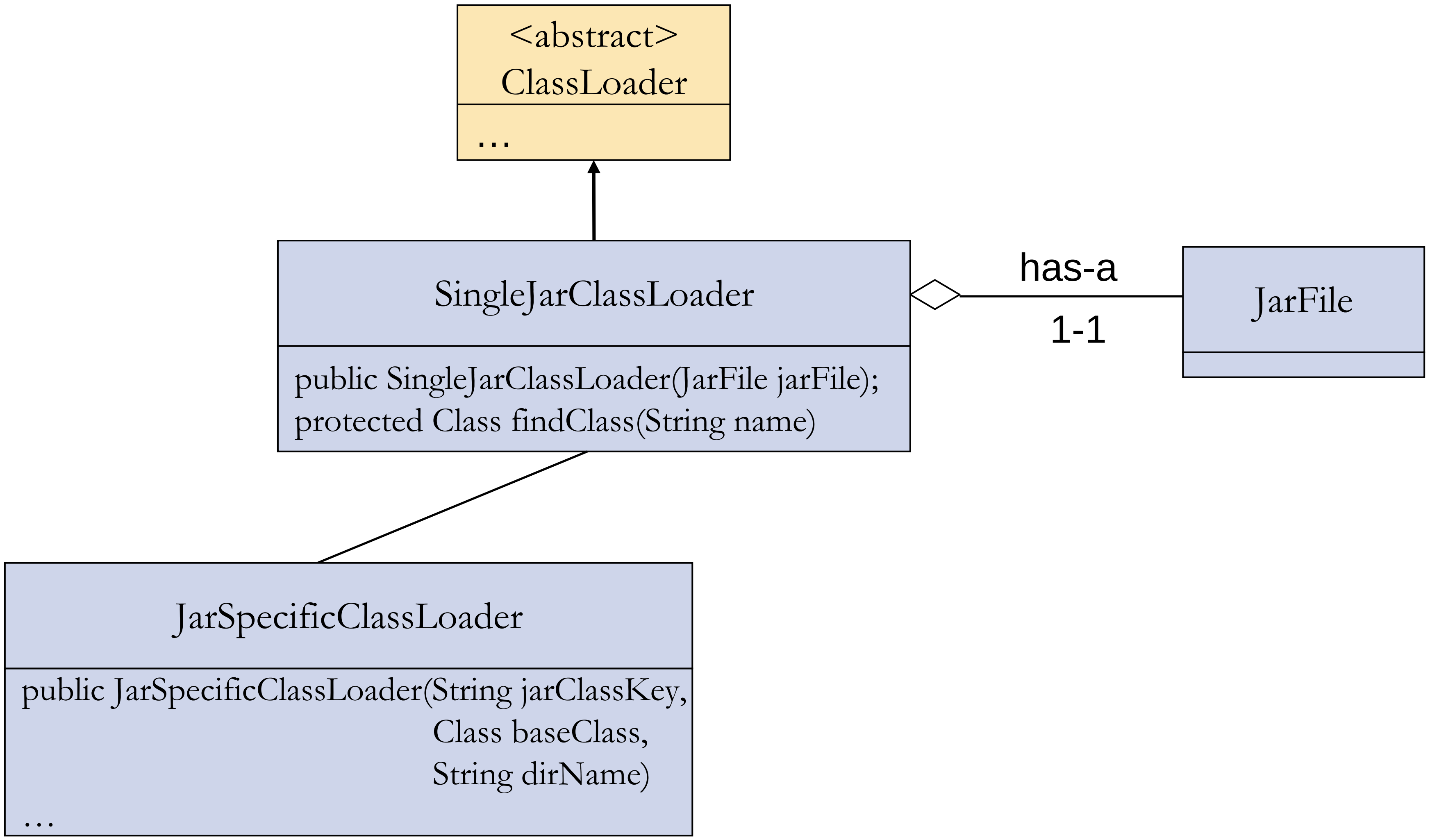
    - Invokes the loadClass method on the parent class loader.

    - Invokes findClass(String)

    - If the class was found and the resolve flag is true:
      invokes the resolveClass(Class) method
  - protected final Class **defineClass**(String name, byte[] b,
    int off, int len)

Subclasses are encouraged to override findClass() instead of loadClass()

An Example



<abstract>
ClassLoader
…
SingleJarClassLoader
public SingleJarClassLoader(JarFile jarFile);
protected Class findClass(String name)
has-a
1-1
JarFile
JarSpecificClassLoader
public JarSpecificClassLoader(String jarClassKey,
Class baseClass,
String dirName)
…