



the
POWER
of
JAVA™



JavaOne
Part of the Oracle Java Software Suite

JSR-277: Java Module System

Stanley M. Ho

JSR-277 Specification Lead
Senior Staff Engineer
Sun Microsystems

Michal Cierniak

Senior Software Engineer
Google

BOF-0684

Bryan Atsatt

Consulting Member of
Technical Staff
Oracle

Three Kinds of Hell

- Classpath Hell
- JAR Hell
- Extension Hell

Timeline

- ✓ JSR-277 Proposal 2005 / June
- ✓ JSR Review Ballot 2005 / June
- ✓ EG Formation 2005 / July – Aug
- ✓ EG Kick-off 2005 / Sept
- Early Draft Review (EDR) Coming soon

Agenda

Modules: Development vs. Deployment

Module files

Versioning Scheme

Distribution Format

Repository

Runtime Support

Summary

Development Modules vs. Deployment Modules

What's the distinction?

- Development Modules
 - A language construct
 - Require direct VM support to enforce semantics (access control)
- Deployment Modules
 - Unit of packaging and distribution
 - Require extensive tool and library support, but not necessarily language or VM support
- Concepts do interact, but the interface between them is relatively narrow

Development Modules

JSR-294

- Handled by JSR-294, whose concerns include:
 - Information Hiding
 - How to develop a system made of several subsystems without exposing subsystems internals
 - Separate Compilation
 - Need to be able to compile against the interface of another “module” without the implementation

- TS-3885: Superpackages: Development Modules in Dolphin
 - Wednesday 12:15 PM - 01:15 PM

Deployment Modules

JSR-277

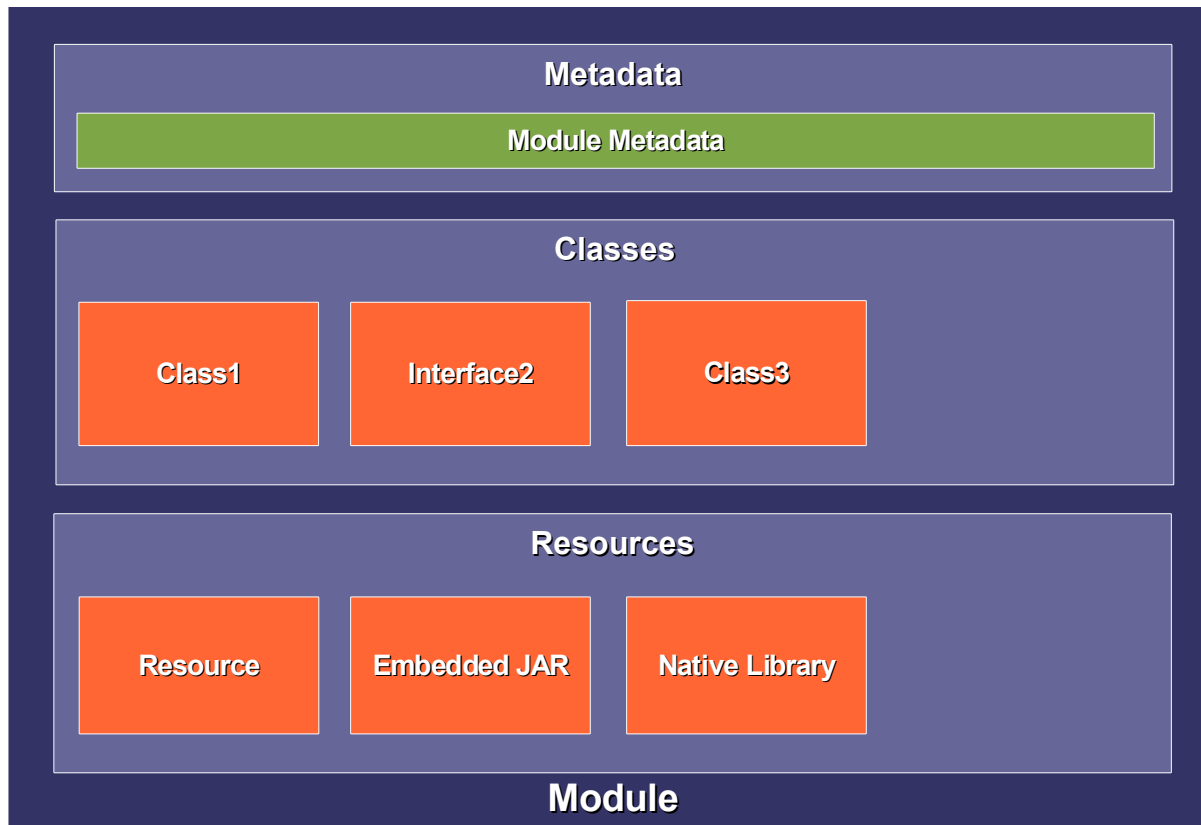
- Versioning
- Distribution and Packaging
- Repositories
- Module Interconnect, and more ...
- In ideal world, a comprehensive solution at language and run time levels covers everything
- In reality, a long standing open research issue
 - Development time issues handled with conservative, simple language constructs
 - Deployment issues handled by tools on top of reflective run time API

Deployment Modules

Continued

- **Module Definition**
 - Identify a logical module
 - Specify what code and resources are provided by the modules, and what the module imports and exports
 - Inherently stateless
- **Module Instance**
 - Instantiation of a module definition
 - Multiple module instances can coexist at run time
- Both exposed as reflective run time APIs
- Enable interoperation with other module systems

Logical View of a Module Definition



Agenda

Modules: Development vs. Deployment

Module files

Versioning Scheme

Distribution Format

Repository

Runtime Support

Summary

Defining a Superpackage

```
super package com.sun.myModule {  
  
    // exported classes and/or interfaces  
    export com.sun.myModule.myStuff.*;  
    export com.sun.myModule.yourStuff.Interface;  
  
    // imported modules  
    import org.someOpenSource.someCoolStuff;  
  
    // module membership  
    com.sun.myModule.myStuff;  
    com.sun.myModule.yourStuff;  
    com.sun.SomeOtherModule.theirStuff;  
}
```

Module Files

Don't take “file” too literally

- The authoritative binary definition of a module
 - Name
 - Exports
 - Imports
 - Membership
 - Metadata
- Class files can claim membership in a module
- Claims must be cross checked with module file
- VM uses membership and export info to enforce access control

Module Files

Continued

- Other information is useful for JSR-277
 - For example, import information can be used to validate configurations
- A module file from JSR-294 corresponds to the metadata of the module definition in JSR-277

Defining a Superpackage with Additional JSR-277 Metadata

```

@Version("1.0.0")
@Exports({
    "icons/graphics1.jpg",
    "icons/graphics2.jpg",
    "META-INF/**"})
@ModuleAttributes({
    @ModuleAttribute("my.greeting", "Hello World!"),
    @ModuleAttribute("java.magic.number", "CAFEBABE")})
@MainClass("com.sun.myModule.myStuff.MainApp")
super package com.sun.myModule {
    export com.sun.myModule.myStuff.*;
    export com.sun.myModule.yourStuff.Interface;
    import org.someOpenSource.someCoolStuff;
    com.sun.myModule.myStuff;
    com.sun.myModule.yourStuff;
    com.sun.SomeOtherModule.theirStuff;
}
  
```

Agenda

Modules: Development vs. Deployment

Module files

Versioning Scheme

Distribution Format

Repository

Runtime Support

Summary

Versioning Scheme

- Simple Version
 - major.minor[.micro[_patch]][-qualifier]
 - 1.7.0
 - 1.7.0_01
 - 1.7.0_01-b32-beta
- Follows existing Java practice

Versioning Policy

How should version number be updated?

- Major number
 - Incremented if not backwardly compatible
- Minor number
 - Incremented if changes are largely backward compatible
- Micro number
 - Incremented if changing implementation details
- Patch number
 - Incremented for bug fixes and patches
- Qualifier
 - Changed for build number or milestone

Version Ranges

- Version Range

- JSR-56 {
- 1.7.0+ // 1.7.0 or later
 - 1.7.0* // 1.7.0 <= x < 1.7.1
 - 1.[7.0+] // 1.7.0 <= x < 2.0.0
 - 1.[7.0+];2*;3* // 1.7.0 <= x < 4.0.0 (union of ranges)
- } family
- Specify version constraint in dependencies
 - Either a simple version or a version range

Agenda

Modules: Development vs. Deployment

Module files

Versioning Scheme

Distribution Format

Repository

Runtime Support

Summary

Distribution Format

- Physical representation of a module definition
- Unit of packaging and deployment
- Optimize for size – based on JAR format
- Contents
 - Module metadata
 - Classes
 - Resources, e.g. image, audio, text, etc.
 - Embedded JAR files
 - Native libraries
- Code signing

JAM (JAVa Module) Format

org.foo.Xml-1.2.3.jam:

/META-INF

MANIFEST.MF

module/org.foo.Xml.module

Module metadata –
under /META-INF/module

/ClassA.class

/InterfaceB.class

Resources – e.g. classes,
icons, etc.

/ClassC.class

/icon/graphics.jpg

/bin/xml-windows.dll

Native libraries –
under the /bin directory

/bin/xml-linux.so

/lib/xml-parser.jar

Embedded JAR files –
under the /lib directory

Agenda

Modules: Development vs. Deployment

Module files

Versioning Scheme

Distribution Format

Repository

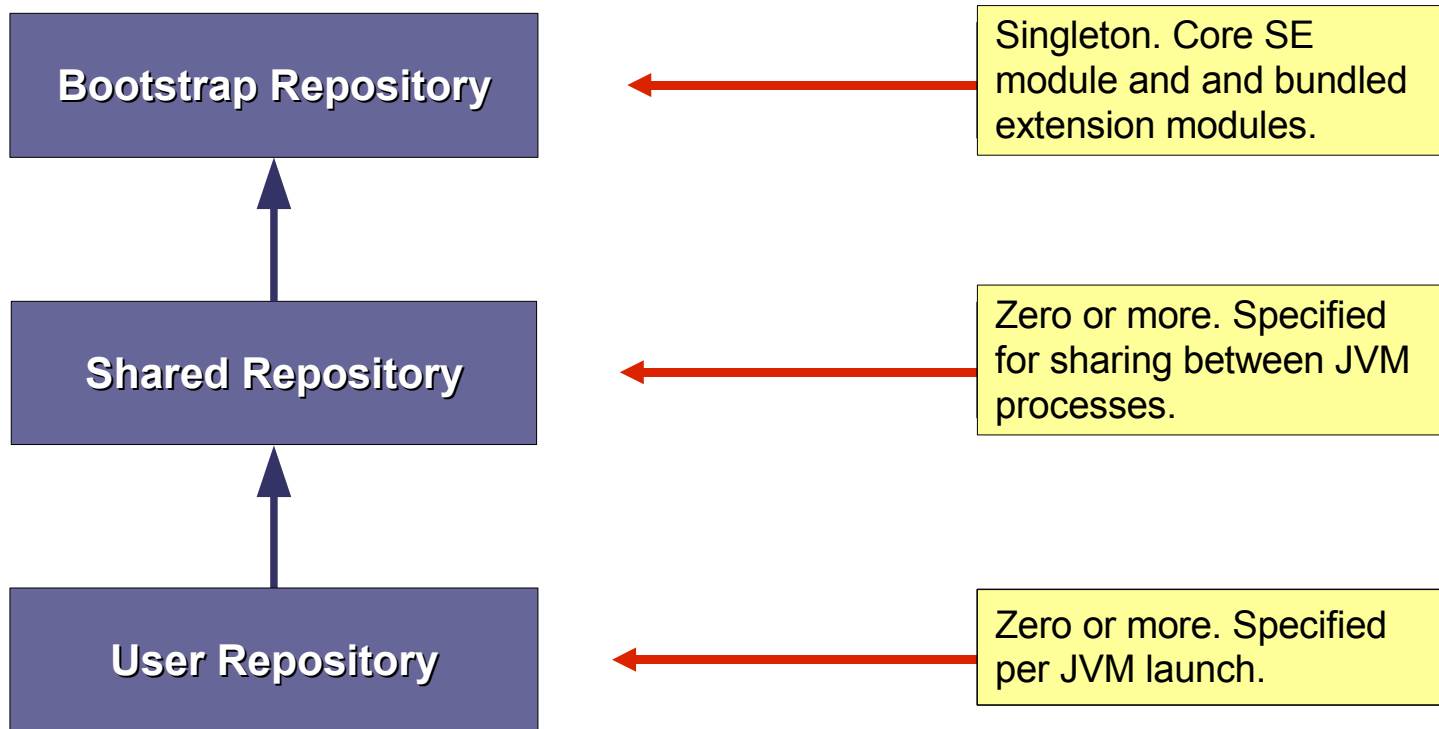
Runtime Support

Summary

Repository

- Discovering, storing, and retrieving module definitions
- Enable side-by-side deployment
 - Allows more than one version of a module definition installed
 - Each module definition is installed in isolation
- Multiple repositories
 - Bootstrap repository
 - Shared repository
 - User repository
- Delegation Model

Repository in Delegation



Agenda

Modules: Development vs. Deployment

Module files

Versioning Scheme

Distribution Format

Repository

Runtime Support

Summary

Runtime Support

- Interconnection
 - Version constraints in imports
 - Construction script
- Validation
- Classloading
- Lifecycle

For More Information

- JSR-277: Java Module System
 - <http://www.jcp.org/en/jsr/detail?id=277>
- JSR-294: Improved Modularity Support in the Java Programming Language
 - <http://www.jcp.org/en/jsr/detail?id=294>
- Developing Modules for Development
 - http://blogs.sun.com/roller/page/gbracha?entry=developing_modules_for_development

Summary

- Java SE 7 will:
 - Provide first class modularity, packaging, and deployment support
- Language level module constructs (JSR-294)
- Deployment level module system (JSR-277)

Q&A





the
POWER
of
JAVA™



JavaOne
Part of the Network for Business Success

JSR-277: Java Module System

Stanley M. Ho
JSR-277 Specification Lead
Senior Staff Engineer
Sun Microsystems

Michal Cierniak
Senior Software Engineer
Google

Bryan Atsatt
Consulting Member of
Technical Staff
Oracle

BOF-0684