JSR 294 and Transparency

Alex Buckley
Spec lead, Java Language & VM
Sun Microsystems
Bio

Spec lead, Java Language and VM (901, 924)
Spec lead, Improved Modularity Support in the Java Language (294)
Co-spec lead, Annotations on Java Types (308)
Maintenance lead, JSRs 14 / 175 / 201 / 202

Spec lead, OpenJDK Project Lambda
Occasional observer, OSGi Alliance CPEG
Academic liaison (ECOOP, OOPSLA)
Multiple slots at JavaOne and Devoxx every year
Disclaimer

My JCP experience is with core Java SE JSRs only

*Remember “Quality / Scope / Cost / Time”*

Global impact (Quality)

Infinite supply of opinions, infinite demand for “more” (Scope)

Finite supply of resources (Cost)

Relationships with other bodies e.g. Java EE, OSGi, OpenJDK (Time)

My comments may not apply to SE/EE JSRs further up the stack
The Java modularity landscape

JSR 294

Language and VM features to support module systems
Expert Group includes Sun, IBM, Oracle, Google, P.Kriens, D.Lea

Jigsaw

A module system for JDK7
Not part of a JSR
May use language and VM features from JSR 294

OpenJDK Project Jigsaw

Hosts the Reference Implementation of JSR 294
Hosts the design and implementation of the Jigsaw module system
Hosts discussion on JDK modularization
Module-private accessibility

When a package exposes “too many” public types, other packages are liable to depend on them.

To share types and members across packages but within a module, make them module-private:

```java
module class Foo {
   module String m() { ... }
}
```

Any class in the same module as Foo can access Foo and Foo.m.

Module-private accessibility stored in Foo.class and enforced by the JVM.

In the 294 RI, module membership is inferred from filesystem layout.
**module** is a “restricted keyword”

Adding keywords that break code is frowned upon

You really can write:

```java
package module;
module class module {
    module module module module = new module();
    module module() {}
}
```

(can != should)
How does this help a module system?
Visibility & Accessibility

Visibility determines whether one type can see another type.

“Observability” in the JLS → sourcepath+classpath in the RI (javac)
Class loading in the JVMS → classpath in the RI (application class loader)
Accessibility determines whether a visible type can be used public/protected/package/private in the JLS and JVMS. For a type to be accessible, it must first be visible.
Module systems on the JVM govern visibility via class loaders. But if the underlying types are public, access is permitted. JSR 294 allows a module system to govern accessibility too.
Module-private accessibility

**Runtime module**: an object whose class implements `RuntimeModule`

```java
public interface java.lang.module.RuntimeModule {
    boolean hasModuleAccess(java.lang.module.RuntimeModule other);
}
```

A module system generates runtime modules to tag classes

```java
public abstract class java.lang.ClassLoader {
    protected final Class<?> defineClass(String name, byte[] b,
             int off, int len, ProtectionDomain protectionDomain,
             java.lang.module.RuntimeModule m);
}
```

The JVM allows access to a module-private type/member iff
the accessing class has the same runtime module, or
the runtime module of the accessed class permits access
Today

Visible classes
Class loaders
Module system
Access control
JVM
Bundles
rt.jar
With JSR 294, it is as if ...
Specification v. Implementation

JSR 292 standardizes linkage protocols, not linkage behavior

JSR 294 standardizes module accessibility, not module boundaries

In OSGi, classes in a bundle usually have the same runtime module

But classes in different bundles can have the same runtime module
And classes in a bundle can have different runtime modules

The RI (javac+Hotspot) puts classes in modules based on *modulepath*

Like *classpath*, but with an additional top level for modules
May also consult Jigsaw to search libraries and resolve dependencies
Modularizing the JDK

Concern about the JDK relates to its size, not its encapsulation.

Jigsaw offers various mechanisms to compatibly split up the JDK:

- Statically resolvable dependencies
- Local dependencies (safe split packages)
- Granular optional dependencies
- Reverse dependencies
- Virtual modules
- Multi-dimensional versions

These mechanisms are entirely about better visibility control.

Module-private accessibility is a second-tier technique for JDK code.
Best practices for transparency

What 294 does

“The public should know who is on the EG”
“Publicly readable alias on which EG business is reported”
“JSR schedule should be published and regularly updated”
“Publicly readable discussion forum or Wiki”
“Publicly writable alias for feedback or comments”
“Spec leads should speak at conferences and events”
“Open-source development processes for the RI and TCK are encouraged...”
“Community Update page should point to all other public communications”

What 294 does not do

“Public issue-tracking (spec issues, RI/TCK bugs)”
Mailing lists (Adopted by 292 + 330)

Private EG list

jsr-294-eg@jcp.org (the traditional JCP EG list)

Normal EG list

jsr294-modularity-eg@cs.oswego.edu
Only EG members can read and write
There is one special member: the Observer list

Observer list

jsr294-modularity-observer@cs.oswego.edu
Anyone can subscribe, read, and write the observer list
Receives all traffic from the normal EG list
EG members are not on the observer list (unless they choose to be)
The Observer list

158 non-digest + 46 digest members (1/11/10)

Subscriptions must be approved, but no moderation

IP flow is controlled (welcome mail has a JSPA-like license)

Gives a voice to members of the public
  Comment on EG discussions, sometimes receiving EG member feedback
  Make novel suggestions that other observers may help work through

Helps people at EG member companies to track activity

It's hard to imagine how the JSR could be more open
  EG primacy is not questioned because most content comes from the EG list
  The EG still has a private place to meet, but hardly ever needs it
Challenges in transparency

The science – visibility v. accessibility – is hardly understood

The engineering has many moving parts (lang, VM, libs, tools...)

The clients – OSGi and Jigsaw – are complex communities

Official mailing lists don't get the message across

   Casual observers see too much detail, not enough vision
   Keen observers read unofficial and uninformed sources too

Most Java developers don't care about principled platform extensions

   Would rather have easier expression than stronger rules
   → Closures has 100x the interest of modularity
Personal suggestions

Mandate a formal scoping exercise before a JSR is formed
  Little EG agreement on scope for most of 294's life
  Sun rolled its own questionnaire for OpenJDK *Project Coin*

Mandate publicly readable and writable observer lists

Provide a PR framework for engaging casual observers
  Early Draft Reviews are too detailed
  What the EG isn't doing is as important as what it is doing

http://java7.tumblr.com/search/294
Thank you

alex.buckley@sun.com