Modular Applications
The Need for Modular Applications

- Applications get more complex
- Assembled from pieces
- Developed by distributed teams
- Components have complex dependencies
- Good architecture
  - Know your dependencies
  - Manage your dependencies
The Entropy of Software

- Version 1.0 is cleanly designed...
The Entropy of Software

- Version 1.1...a few expedient hacks...we'll clean those up in 2.0
The Entropy of Software

- Version 2.0...oops...but...it works!
The Entropy of Software

• Version 3.0...Help! Whenever I fix one bug, I create two more!
Version 4.0 is cleanly designed. It's a complete rewrite. It was a year late, but it works...
The Entropy of Software

- Version 4.1...does this look familiar?....
Modular applications

- Discover their components at runtime
- May add/remove/reload components at runtime
- Must satisfy dependencies between components
- Have API contracts between components
- Run inside a runtime container
Modular Runtime Containers Must

- Ensure dependencies are satisfied
- Not allow illegal dependencies
- Allow legal dependencies
- Instantiate components of the system at runtime
- Provide service registration/discovery facility
Architecture

NetBeans IDE
NetBeans Platform
Swing / JDK
Java VM

NetBeans Application
Architecture
Module System

Module Suite (= Deployment Unit)

Module A (= JAR File)
- META-INF/manifest.mf
- layer.xml
- META-INF/services/*
- *.class
- Bundle.properties

Module B

Module C

- Well-defined module dependencies
- Lazy loading / Unloading
- layer.xml for declarative registrations (file system)
- Bundle.properties for internationalization
Public packages are explicitly defined in manifest.mf (project.xml).
Linkage Dependencies

- Each module loaded by own classloader
- Link to classes of modules they explicitly depend on.
- No cyclic dependencies

Diagram:
- Module A
  -→ Module B
  -→ Module C
  -× B to C
  -× C to B
Runtime Dependencies

- Modules can require presence of others during runtime
- Useful for modular libraries
- to require implementations

Module A

Module B

Module C

May require presence
Types of library

- Simple library
  > one impl, put it on classpath and use

- Reference Impl + Vendor Impl
  > You trust that the Vendor impl conforms to the spec

- Modular Library – the API is separate from the implementation
  > Multiple implementations possible
  > Spec conformance is enforced by design
  > API must find its implementation
  > You need a registry of things
The Java Extension Mechanism

- ServiceLoader
  - see lecture 1
Lookup – NetBeans solution

- Small, NetBeans independent library
  > org-openide-util.jar
- Works with any version of Java
- A Lookup is dynamic
  > Can fire changes
- A Lookup is instantiable
- Lookups are composable
Lookup – service discovery

• Global Lookup Patterns
  > Pseudo-singletons:
    ```java
    Lookup.getDefault().lookup(SomeClass.class)
    ```
  > Global services
    ```java
    Lookup.Result<SomeClass> r = Lookup.getDefault().lookupResult(SomeClass.class);
    Collection<SomeClass> c = r.allInstances();
    ```

• Simple registration
  > Using @ServiceProvider annotation
    ```java
    @ServiceProvider(service=SomeClass.class)
    public class SomeClassImpl extends SomeClass {
    }
    ```
Listening for changes

```java
Lookup l = Lookup.getDefault();
Lookup.Result<SomeClass> r =
    l.lookupResult ( SomeClass.class );

r.addLookupListener (new LookupListener() {
    public void resultChanged (LookupEvent e) {
        //handler code here
    }
});
```
Clean Unloading/Reloading

- Get a Lookup.Result from the META-INF/services
  > Lookup (Lookup.getDefault() +/-)
- If a module is uninstalled, it will fire changes
Lookup

• Used “almost” everywhere in NetBeans
  > solution to “almost” everything
Example: Project API

- Associates a directory on disk with a Lookup
- Defines interfaces that may be in that Lookup

```java
public interface Project extends Lookup.Provider {
    FileObject getProjectDirectory();
    Lookup getLookup();
}
```
Example: Selection in NetBeans

- Each main window tab has its own Lookup
  - Some tabs show Nodes, which also have Lookups, and proxy the selected Node's Lookup
- A utility Lookup proxies the Lookup of whatever window tab has focus

Lookup lkp = Utilities.actionsGlobalContext();
Lookup is a place

- A space objects swim into and out of
- You can observe when specific types of object appear and disappear
- You can get a collection all of the instances of a type in a Lookup
Useful utility implementations

- AbstractLookup + InstanceContent
  > Lookup whose contents you can manage
- Lookups.singleton(Object)
  > one item Lookup
- Lookups.fixed(Object[])
  > unchanging Lookup
- Lookups.exclude(Lookup, Class[]);
- ProxyLookup(Lookup[] otherLookups)
  > compose multiple lookups
Defining extension point

- interface to be implemented
  ```
  interface Foo {
  ...
  }
  ```

- extension point
  ```
  Lookup.Result<Foo> rslt =
  Lookup.getDefault().lookupResult(Foo.class);
  for (Foo foo : rslt.allInstances()) {...}
  ```

- in third-party modules
  ```
  @ServiceProvider(service=Foo.class)
  class FooImpl1 implements Foo {
  ...
  }
  ```
Context sensitive actions

- Action observes presence of a type in the global context
  > `xxxCookie`
  > `Utilities.actionsGlobalContext();`
SaveCookie vs. Savable

- Saving of documents in an editor, etc
  - by adding SaveCookie implementation to the global context
    - std Save button listens for the SaveCookie
- From 7.1
  - Savable instead of SaveCookie
    - SaveCookie still exists and implement Savable
  - AbstractSavable abstract implementation of Savable
Communication between windows

- A window puts an object to its lookup
- Another window listens on the global context

```java
InstanceContent ic = new ...
associateLookup(
    new AbstractLookup(this));

ic.set(...)
```

```java
public void componentOpened() {
    result = Utilities.
        actionsGlobalContext().
            lookupResult(Foo.class);
    result.addLookupListener(this);
    result.allItems();
}
```

```java
public void resultChanged(LookupEvent ev) {
    Collection<? extends Foo> words =
        result.allInstances();
    if (!words.isEmpty()) {
        ....
    }
}
```
Quick Search API
Quick Search x API

• A tool – searching box in toolbar
  > Find actions
  > Learn shortcuts
  > Classes by name
  > Help system

• API
  > Custom search content
Configuration in Platform

- Disabled by default, enable
  - Modify XML layer
  - Copy from sample in docs
- Configure set of providers
  - Searching in actions pool and recent searches enabled by default
  - Hide recent searches
    - <folder name="QuickSearch">
    - <folder name="Recent_hidden">
    - </folder>
    - </folder>
Providing search content

- Implement `SearchProvider`
  - Only one method `evaluate(...)`
- Register in XML layer
  - Defines category, prefix, position
- Use Quick Search Wizard
  - New File → Module Development → Quick Search
  - Skeleton `SearchProvider` implementation and XML generated automatically
Implementation notes

- SearchRequest → SearchResponse.addResult(...)  
  > Fill results incrementally in the cycle
- Stop search when needed  
  > Finish when addResult(..) returns false
- Don't replan to other thread  
  > evaluate(..) runs out of EventQueue already
- Category sharing is possible  
  > ...by sharing category String id
Resources

• Javadoc documentation

• Quick Search tutorial
  > http://platform.netbeans.org/tutorials/nbm-quick-search.html