NetBeans Platform

Tom Wheeler

11/11/09

SIUE :: November 2009

Intro: What I Cover in this Section

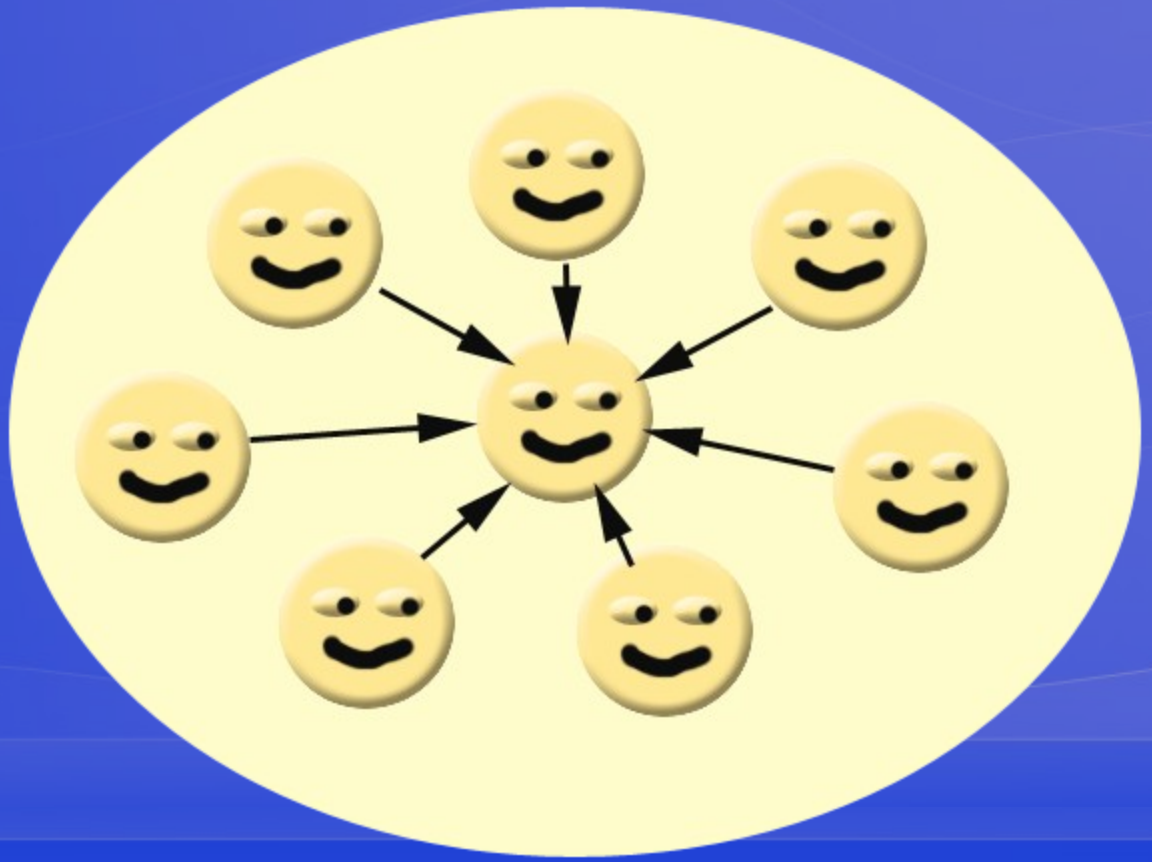
- Why modular applications are needed
- Benefits of modular applications
- NetBeans' Lookup API
- How to separate API from implementation
  - How to register implementations
  - How to find them at runtime
- Common Lookup idioms in NetBeans

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

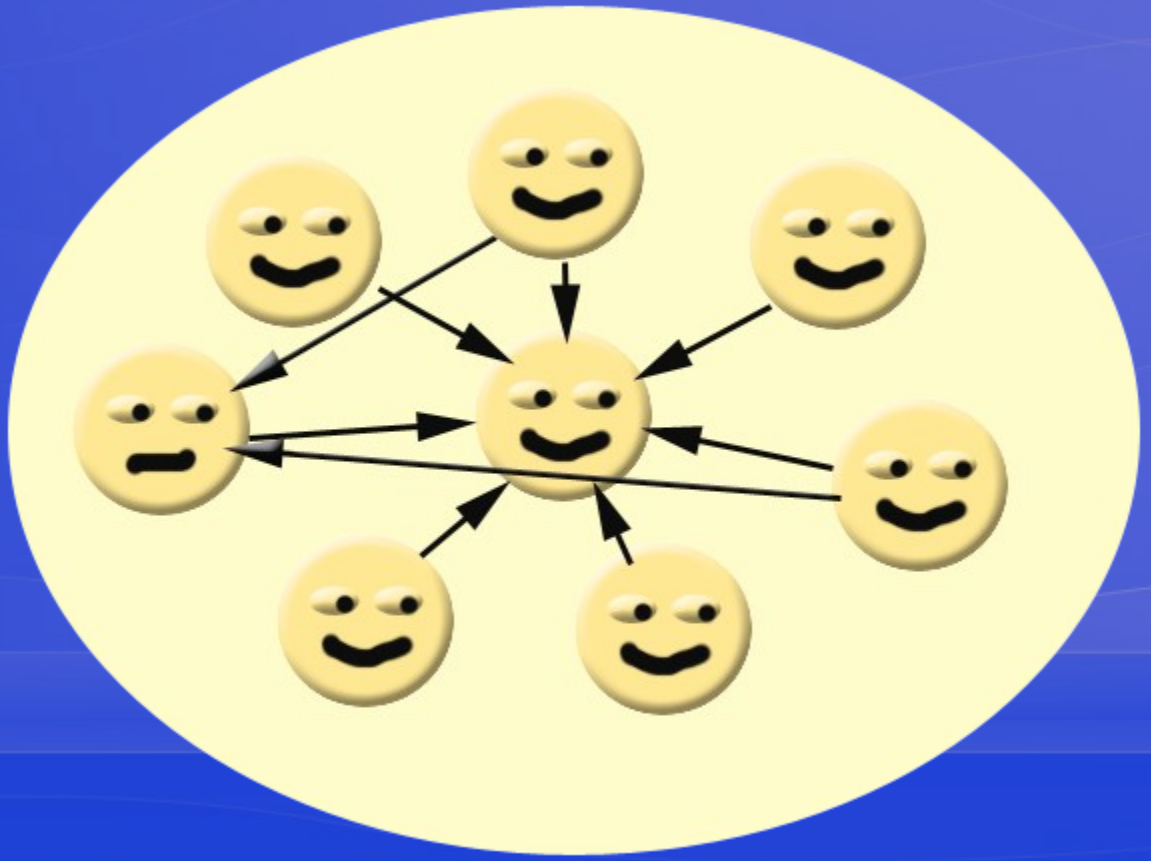
The Entropy of Software

- Version 1.0 is cleanly designed...



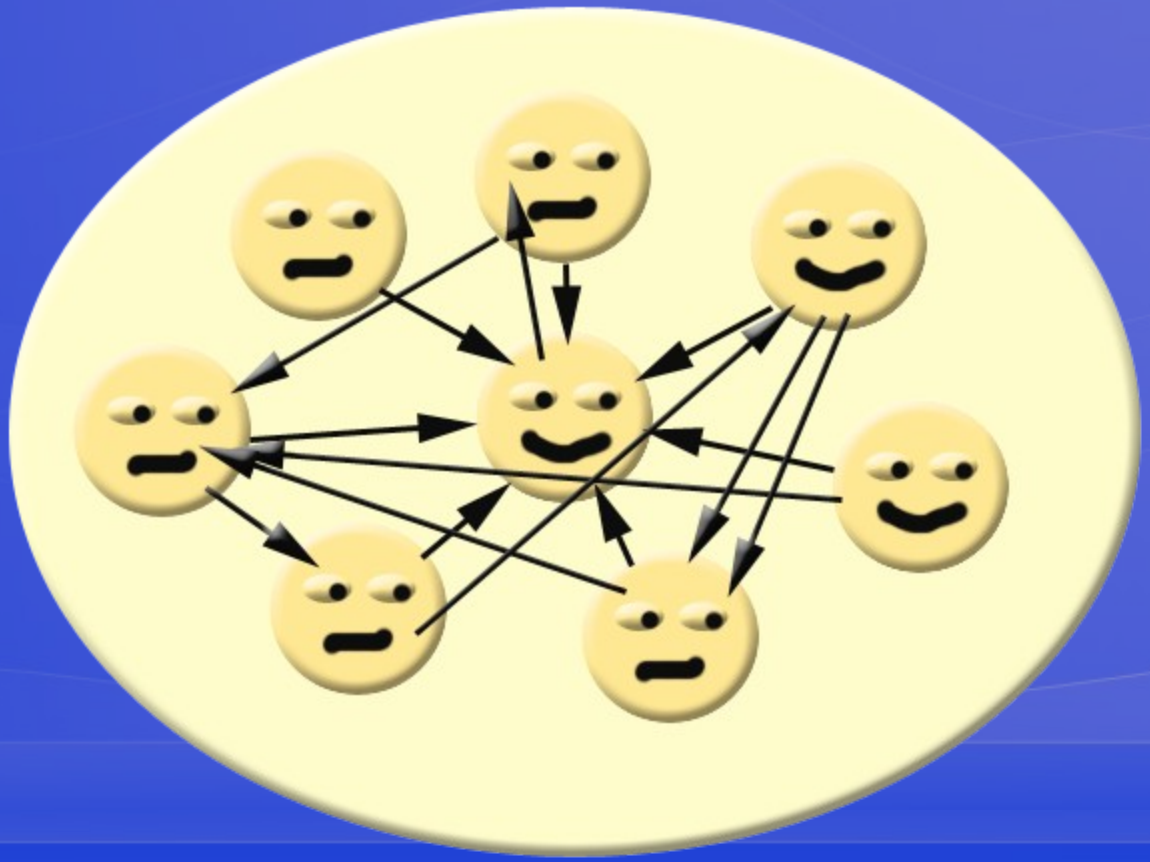
The Entropy of Software

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



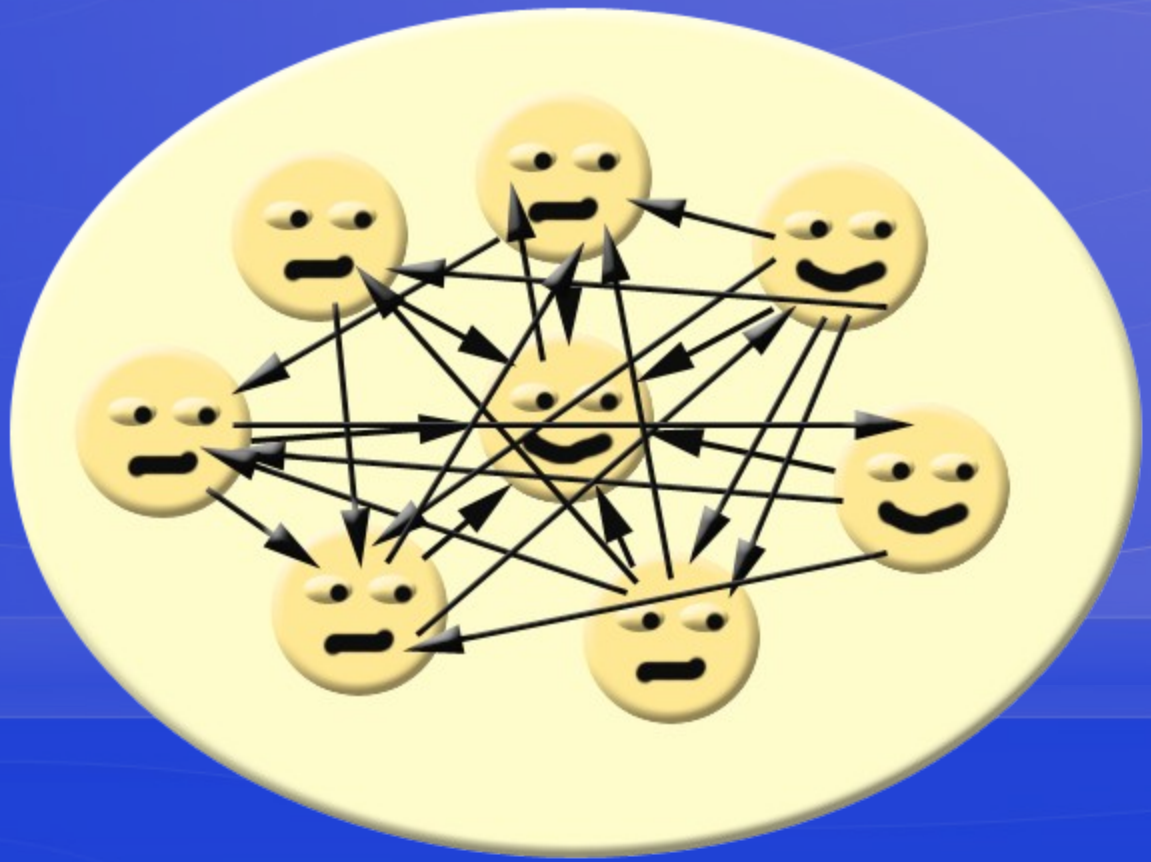
The Entropy of Software

- Version 2.0...oops...but...it works!



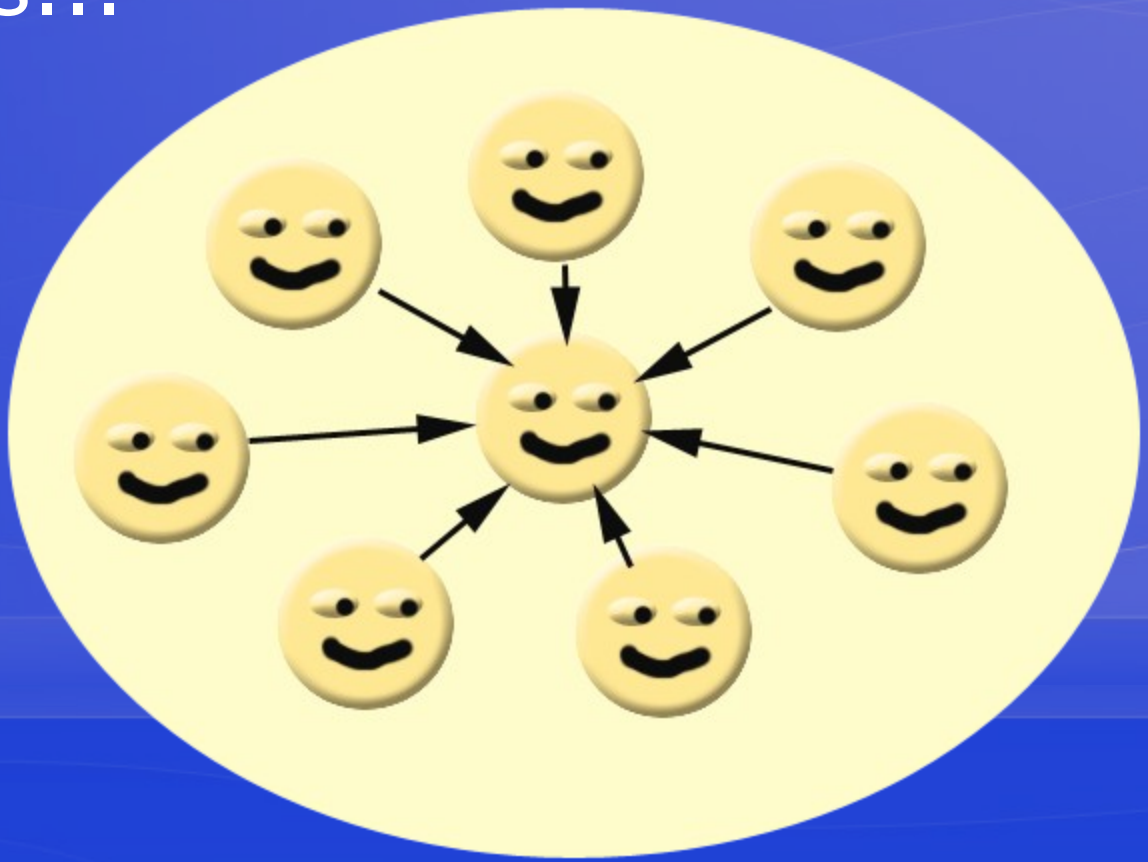
The Entropy of Software

- Version 3.0...Help! Whenever I fix one bug, I create two more!



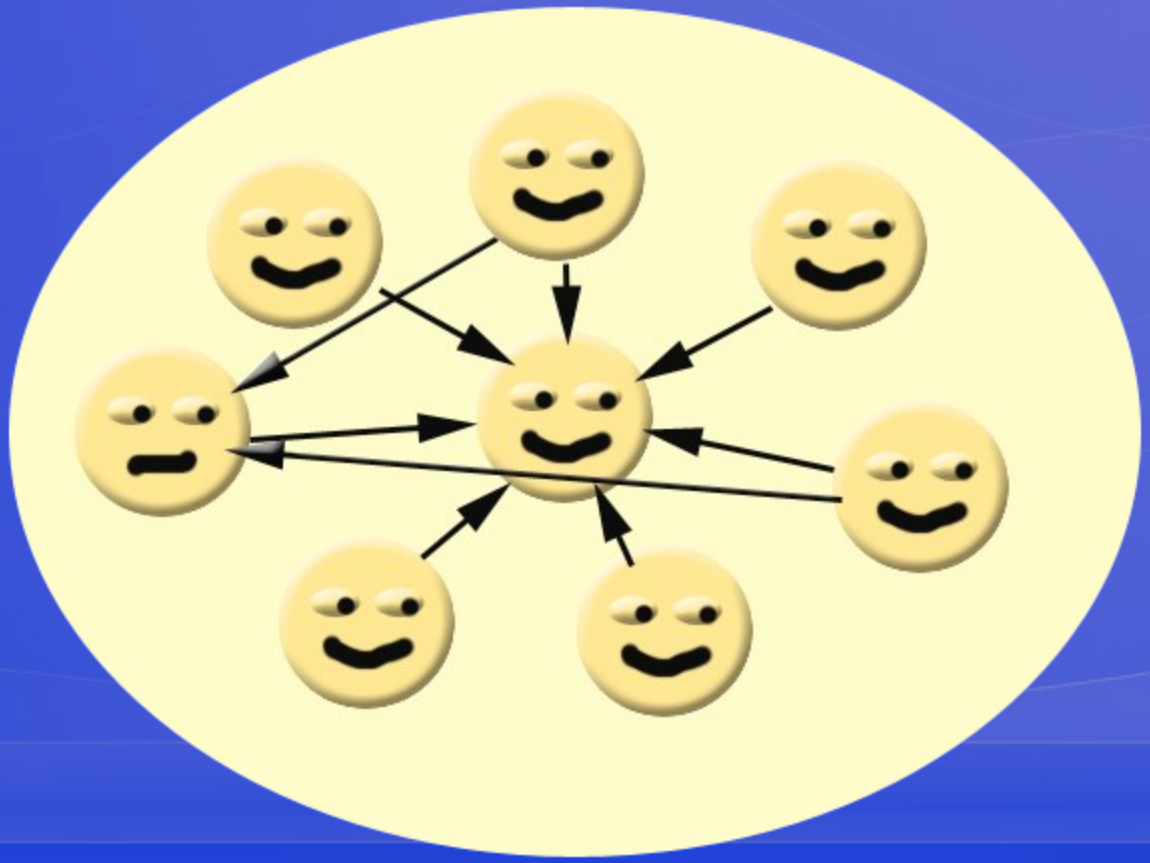
The Entropy of Software

- 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?....



The Entropy of Software

- TO BE CONTINUED….

API, Implementation and Client

- Application Programming Interface
  - Defines behavior of a class library
  - But doesn't typically contain any logic
  - Kind of like a word processor template
- Implementation
  - Fulfills the contract specified by the API
  - Provides actual business logic
- Client: code which calls the API

API and Implementation: Demo

- I'll create a simple Tax Calculator API
- And an implementation of that API

Benefits of Modularization

- Separate API from implementation
- Can easily replace implementation later
  - Create something “quick-and-dirty” now
  - Create something better when time allows
  - Shouldn't require any change to your app.
- Can even plug in new impl. at runtime
- Can have multiple implementations
  - Can allow the user to select one at runtime
  - Handy for file format support

Modular Runtime Containers Must

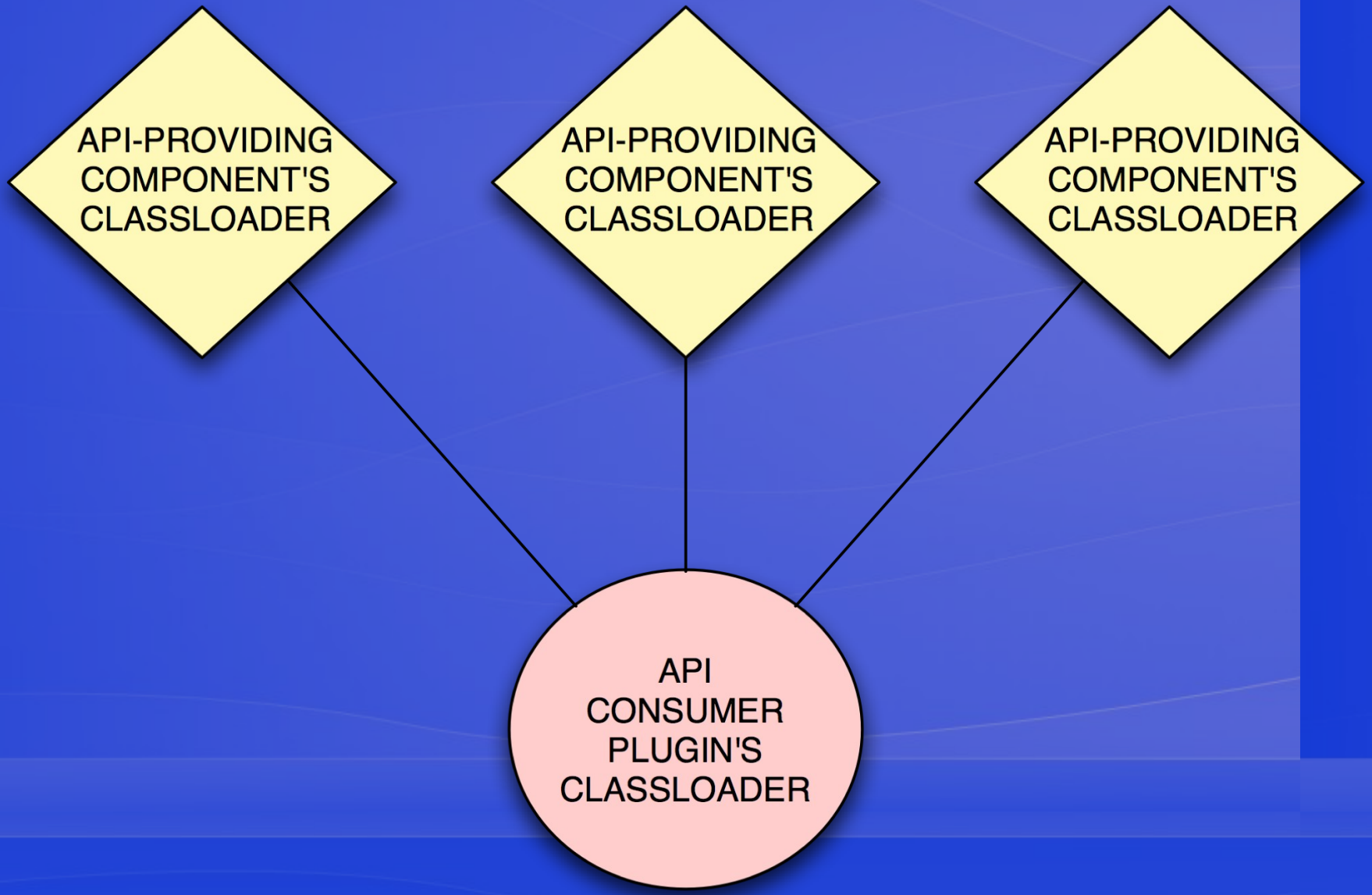
- Ensure dependencies are satisfied
  - In NetBeans, enforced at build and runtime
- Allow only legal dependencies
  - In NetBeans, no circular dependencies
- Instantiate services at runtime
- Allow for service registration
- Allow for service discovery

Use an Existing Runtime Container

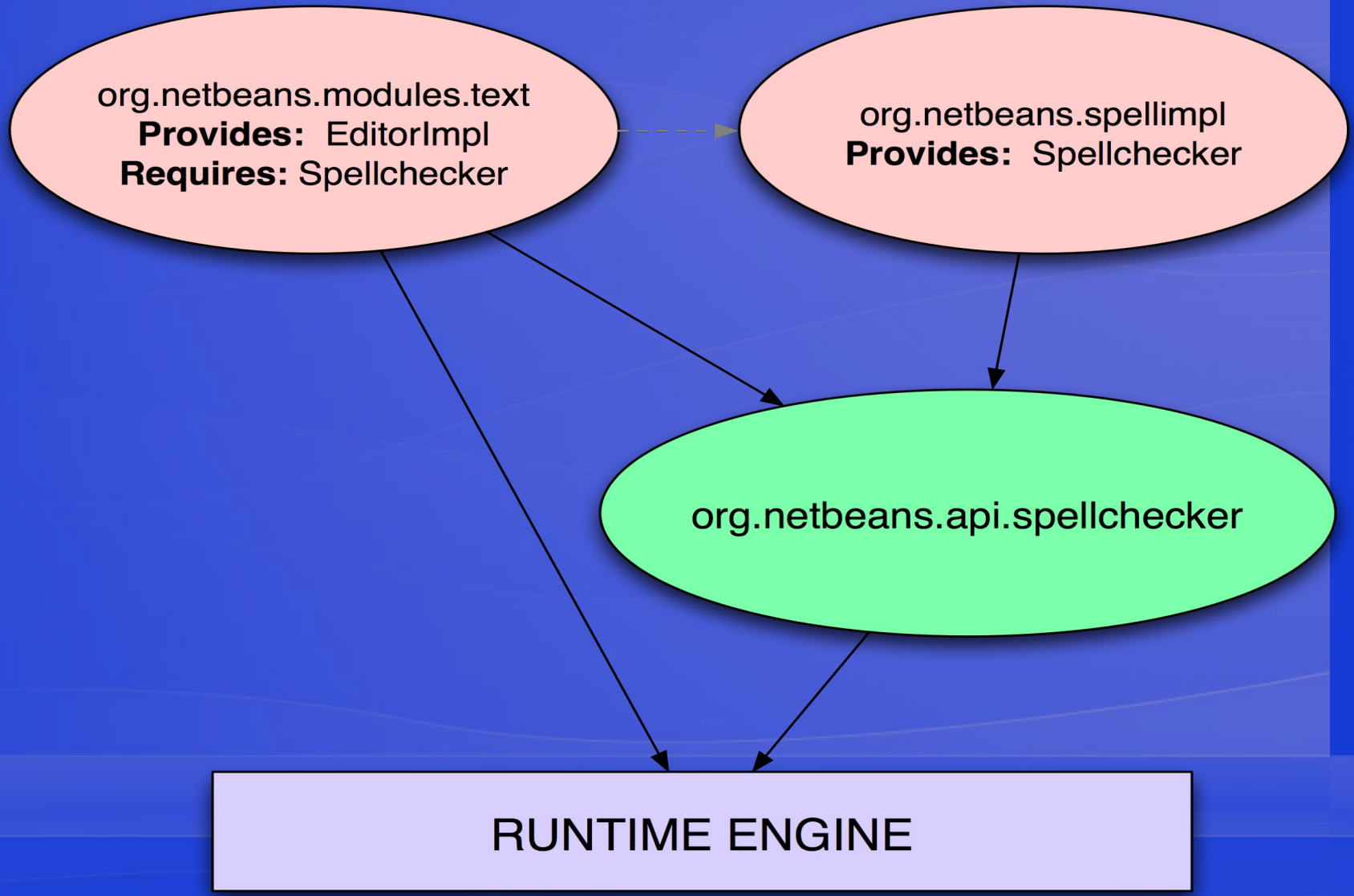
RIP Homemade Frameworks 1995-2005

There is no good reason to create your own!!!

Class Loader Partitioning



Modular Libraries and Discovery



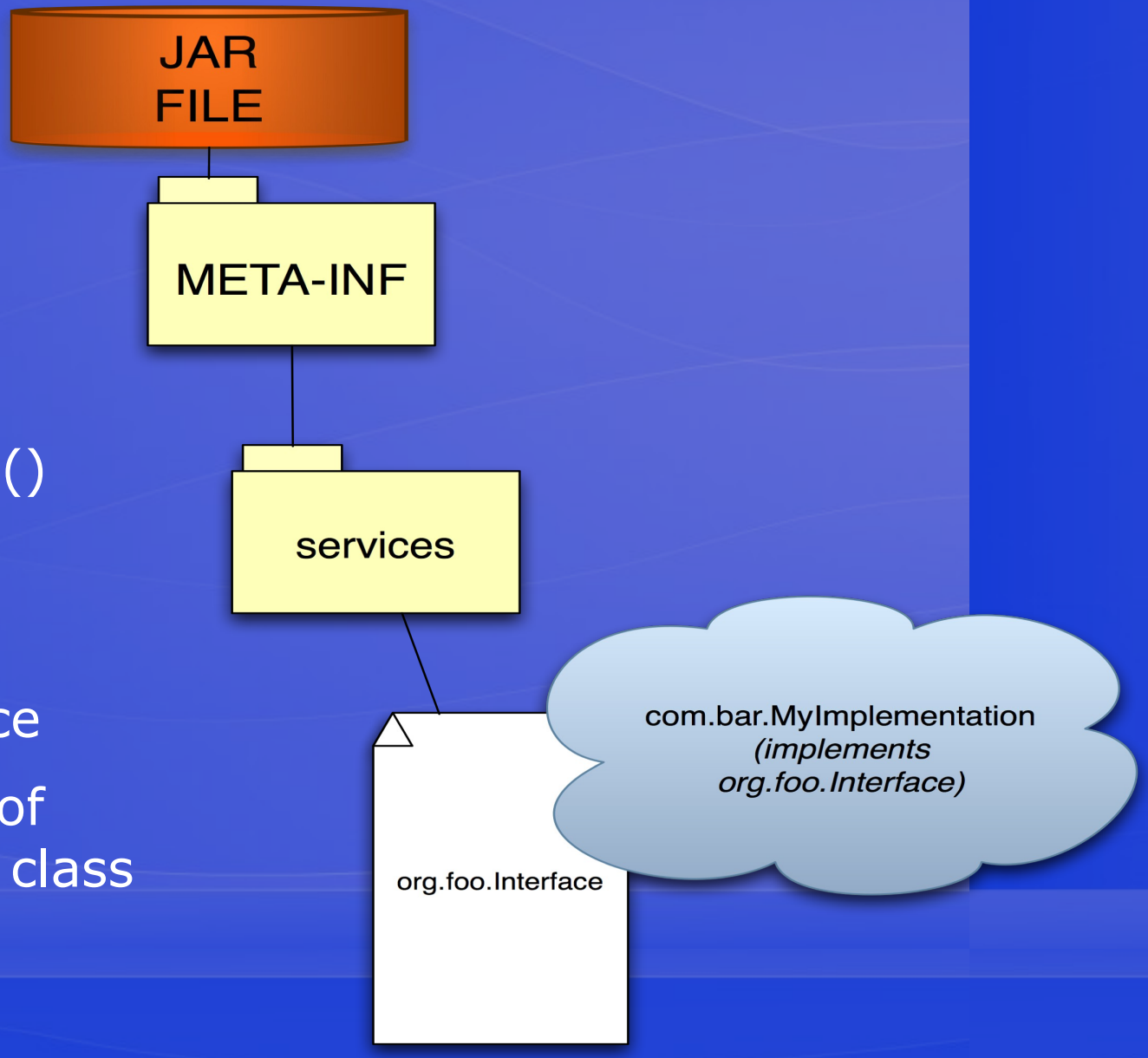
Discovery and Dependencies

?

So how will the SpellChecker API find its implementation?

The Java Extension Mechanism

- In JDK since 1.3
- Easy with JDK 6's ServiceLoader.load()
- Plain-text file in META-INF/services
  - Name is interface
  - Content is FQN of implementation class



Lookup – NetBeans Solution

- Small, NetBeans independent library
  - org-openide-util.jar
- A Lookup is dynamic
  - Its contents can change (and fire events)
  - Interested classes can listen to changes
- A Lookup is instantiable
- Lookups are composable
- Can even use this outside NB Platform!

The Global Lookup

- The “Global” lookup is basically a singleton
  - You can access it easily (example forthcoming)
  - Use it to find reg. implementations of APIs
    - They're registered via META-INF/services
    - Or annotations...
    - Or system filesystem...
    - Such impls are commonly called “services”

A Global Lookup Example

- Suppose you have a SpellChecker API
  - And at least one implementation
  - Registered as described earlier
- Example:

```
SpellChecker sc = Lookup.getDefault().lookup(SpellChecker.class)
```

- Client code only need to know about API
  - Don't need to know name of impl class!
  - Don't even need to know impl module!

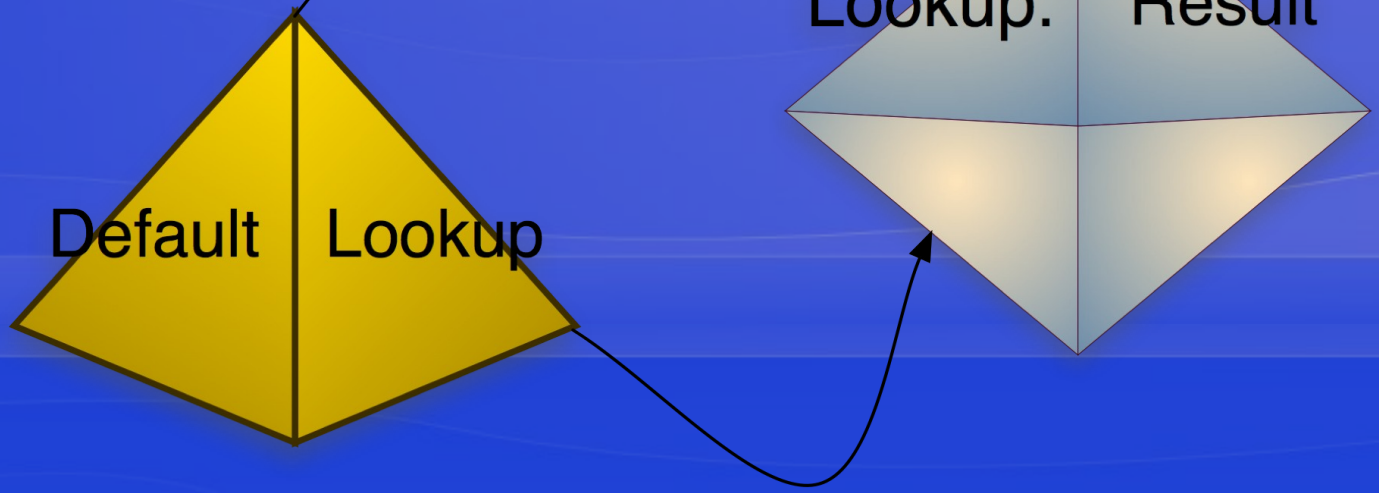
Lookup: Finding All Implementations

- Previous example found just one impl.
  - What if you want to find them all?

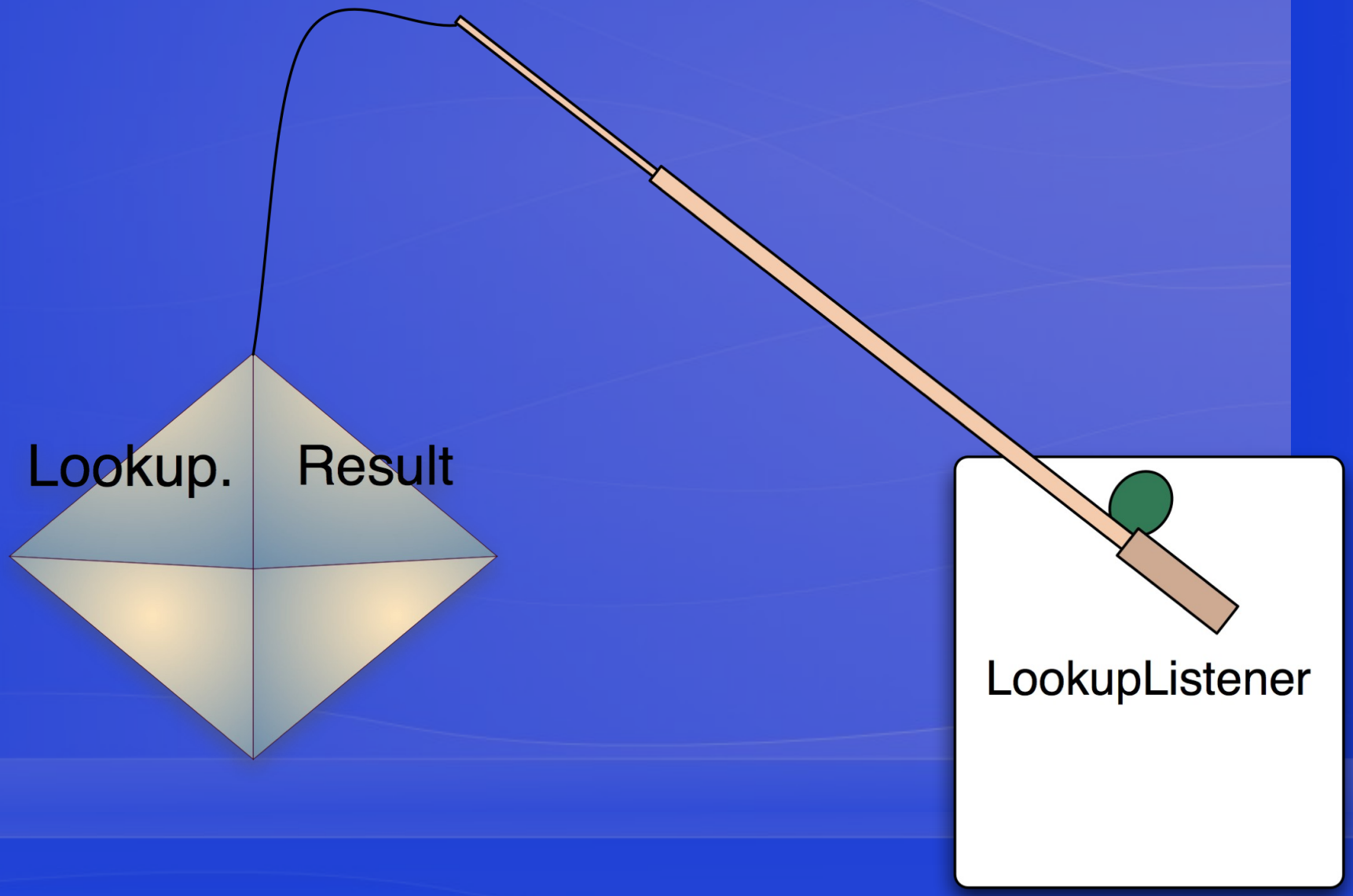
```
Lookup.Result<SpellChecker> r =
  Lookup.getDefault().lookupResult
    (SpellChecker.class);

Collection <SpellChecker> c =
    r.allInstances();
```

Lookup.Result

Lookup.Result result = Lookup.getDefault().lookup (
new Lookup.Template (MyInterface.class))



Why is That Interesting?



Clean Unloading/Reloading

- This is how you get the “Global” lookup

```
Lookup lkp = Lookup.getDefault();
```

- You can add a listener to it (see next slide)
- If a module is uninstalled, it will fire changes

Listening for Changes

```
Lookup.Result<SomeClass> r =
  someLookup.lookupResult ( SomeClass.class );

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

So...What's So Special About This?

?

What if objects had Lookups?

What if Lookups could proxy each other?

A 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

Local Lookups

- In addition to Global Lookup...
  - There are also other lookups
- Anything can provide a Lookup
  - If it implements Lookup.Provider interface
  - Nodes, DataObjects & TopComponents do this
  - Common idiom to find an object's capabilities
  - More details about this later in the course

Local Lookup: Selection in NetBeans

- Each main window tab (TopComponent) 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();
```

Useful Utility Implementations

- AbstractLookup + InstanceContent
  - Lookup whose contents you can manage
- Lookups.singleton(Object)
  - A Lookup which contains exactly one thing
- Lookups.fixed(Object[])
  - A Lookup which does not change

Useful Utility Implementations

- These two proxy to another Lookup
  - ProxyLookup (Lookup[] otherLookups)
    - Compose Lookup from other lookups
  - Lookups.exclude (Lookup, Class[])
    - Allows you to filter out instances based on class

Review Questions

- What are two benefits of modular apps?
- Define “API” and “implementation”
- What is meant by a “service”
- What's a common way of registering a service in the NetBeans Platform?
- Can you listen to changes in a Lookup?

Recap

- Modularity provides many benefits
  - Faster development
  - Easier maintenance
  - More flexibility
- NetBeans' Lookup API
  - Helps make this possible
  - Allows you to separate API from impl
  - Can easily register & find services
  - Listen/react to changes in available services

Exercise (30 minutes)

- We'll work together to create a simple platform app which calculates sales tax.
- It will have three modules
  - API
  - Implementation
  - Client