How to Write APIs That Will Stand the Test of Time

Tim Boudreau and Jaroslav Tulach

Sun Microsystems
http://www.netbeans.org

TS-6218

Design to Last

First Version Is Always Easy

Learn why to strive for good API design and few tricks how to do it from guys who maintain NetBeans™ framework APIs for more than five years

Agenda

Why Create an API at All?

What Is an API?

API Design Patterns

API Design Anti-Patterns

Distributed Development

- There are a lot of Open Source Solutions
  - ant, jalopy, velocity, tomcat, javacc, junit
- Applications are no longer written, but composed
  - Linux distributions, Mac OS X
- Source code spread around the world
- Exact schedule is impossible

Modular Applications

- Composed from smaller chunks
  - Separate teams, schedule, lifecycle
- Dependency management
  - Specification Version 1.34.8
  - Implementation Version Build20050611
  - Dependencies chunk-name1 ≥ 1.32
- RPM packagers
- Execution containers like NetBeans™ technology

http://platform.netbeans.org/

What Is an API?

- API is used for communication
  - Build trust, clearly describe plans
- Evolution is necessary
  - Method and field signatures
  - Files and their content
  - Environment variables
  - Protocols
  - Behaviour
  - L10N messages

http://openide.netbeans.org/tutorial/api-design.html

Preservation of Investments

- Backward compatibility
  - Source vs. binary vs. cooperation
- Knowing your clients is not possible
- Incremental improvements
- First version is never perfect
- Coexistence with other versions

http://openide.netbeans.org/tutorial/api-design.html

Rules for Successful API design

- Use case driven API design
  - Use cases → scenarios → javadoc
- Consistent API design
  - An interface that is predictable serves better than one which is locally optimal but inconsistent across the whole set
- Simple and clean API design
  - Less is more—expose only necessary functionality
- Think about future evolution
  - First version is not going to be perfect

Stability of APIs

- It is all about communication
- APIs can serve different purposes
  - Early adopters
  - Internal communications
  - Framework APIs
- We have stability categories
  - Private, friend
  - Under development, stable, standard
  - Deprecated

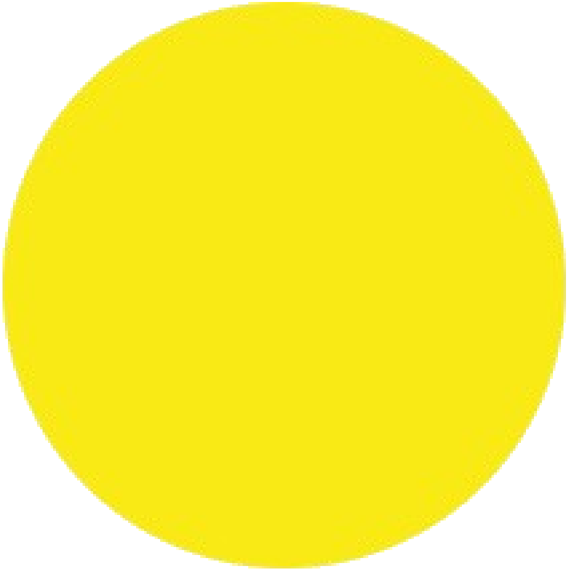
http://openide.netbeans.org/tutorial/api-design.html#life

Evaluation of an API Quality

- Customer-centric—easy to use
- Use cases, scenarios, javadoc
- Future evolution
- Test coverage
- Quality = code Δ specification
- The “amoeba” model

NetBeans API Reviews http://openide.netbeans.org/tutorial/reviews/

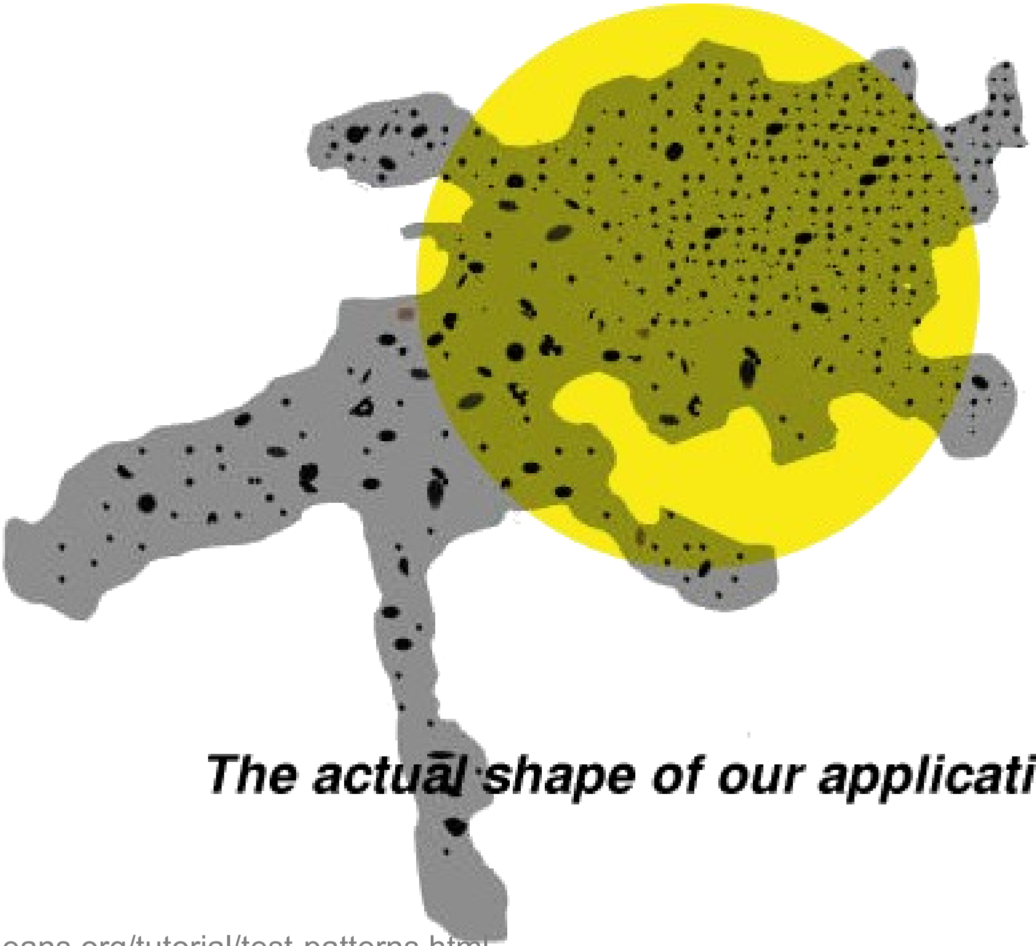
The Amoeba Model



How we think our application looks like

http://openide.netbeans.org/tutorial/test-patterns.html

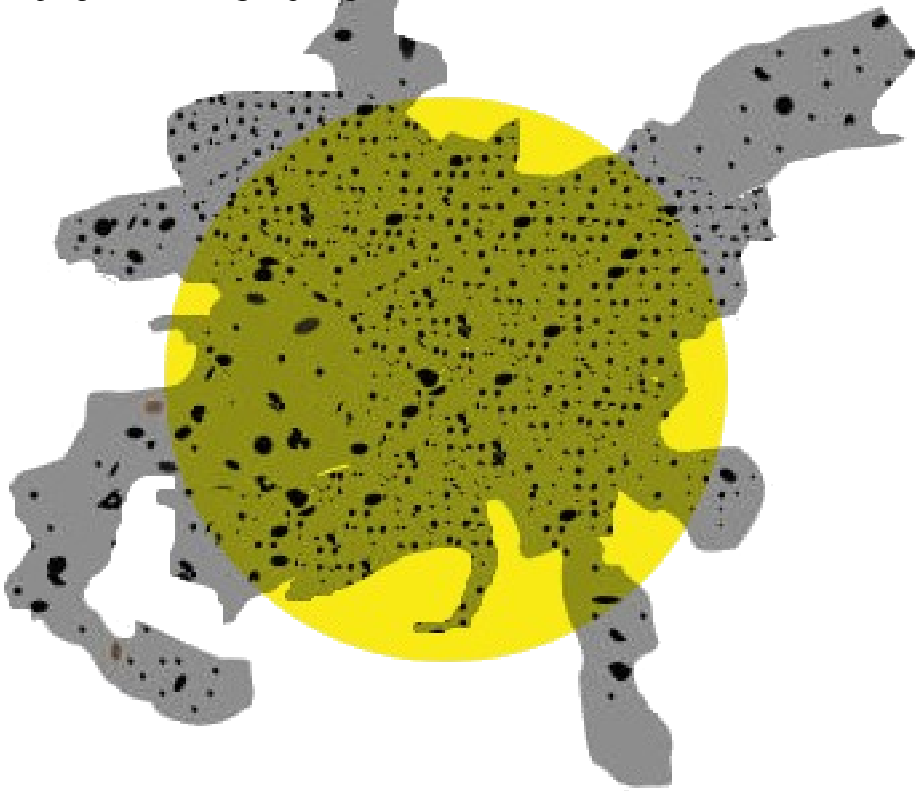
The Amoeba Model



The actual shape of our application

http://openide.netbeans.org/tutorial/test-patterns.html

The Amoeba Model



Shape of amoeba after next release

http://openide.netbeans.org/tutorial/test-patterns.html

Design Patterns

- “Recurring solutions to software design problems”
  - Common name
  - Description of the problem
  - The solution and its consequences
- Simplify description of the architecture

http://openide.netbeans.org/tutorial/api-design.html

API Design Patterns

- Design patterns as well
  - simplify description of the architecture
- API framework vs. internal design
- Main emphasis is on evolution
- First version is never perfect

http://openide.netbeans.org/tutorial/api-design.html

Factory Method Gives More Freedom

Do Not Expose More Than You Have To

```
// exposing constructor of a class like
ThreadPool pool = new GeneralThreadPool();
// gives you less freedom then
ThreadPool pool = ThreadPool.createGeneral();
```

- The actual class can change in future
- One can cache instances
- Synchronization is possible

http://openide.netbeans.org/tutorial/api-design.html

Method Is Better Than Field

Do Not Expose More Than You Have To

```
class Person extends Identifiable {
  String name;
  public void setName(String n) {
    this.name = n;
  }
```

- Synchronization is possible
- Validation of input parameters in setter can be done
- The method can be moved to super class

http://openide.netbeans.org/tutorial/api-design.html

Non-Public Packages

Do Not Expose More Than You Have To

```
OpenIDE-Module-Module: org.your.app/1
OpenIDE-Module-Public-Packages: org.your.api
OpenIDE-Module-Friends: org.your.otherapp/1
```

- NetBeans allows to specify list of public packages
- Enforced on ClassLoader level
- Possible to enumerate modules that can access them
- Split API classes into one package and hide the rest

http://openide.netbeans.org/tutorial/api-design.html

Restrict Access to Friends

Do Not Expose More Than You Have To

- Use package private classes
- Java technology does not have friend packages, but...

```
public final class api.Item {
    /** Friend only constructor */
    Item(int value) { this.value = value; }
    /** API method(s) */
    public int getValue() { return value; }
    /** Friend only method */
    final void addListener(Listener l) { ... }
}
```

http://openide.netbeans.org/tutorial/api-design.html

Restrict Access to Friends (Cont.)

Do Not Expose More Than You Have To

```
/** The friend package defines an accessor
 * interfaces and asks for its implementation
 */
public abstract class impl.Accessor {
  public static Accessor DEFAULT;
  static { Object o = api.Item.class; }

  public abstract Item newItem(int value);
  public abstract void addListener(
        Item item, Listener l);
}
```

http://openide.netbeans.org/tutorial/api-design.html

Restrict Access to Friends (Cont.)

Do Not Expose More Than You Have To

```
class api.AccessorImpl extends impl.Accessor {
  public Item newItem(int value) {
    return new Item(value); }
  public void addListener(Item item, Listener l) {
    return item.addListener(l); }
}
public final class Item {
  static {
   impl.Accessor.DEFAULT = new api.AccessorImpl();
  }
}
```

http://openide.netbeans.org/tutorial/api-design.html

The Difference Between Java Code and C Code

Separate Client and Provider API

- Imagine API for control of media player in C

```
void xmms_pause();
void xmms_add_to_playlist(char *file);
```

- Java version is nearly the same

```
class XMMS {
  public void pause();
  public void addToPlaylist(String file);
}
```

- Adding new methods is possible and beneficial

http://openide.netbeans.org/tutorial/api-design.html

Provider Contract in Java Code and C Code

Separate Client and Provider API

- Now let’s write the interface for playback plugin in C

```
// it takes pointer to a function f(char* data)
void xmms_register_playback((void)(f*)(char*));
```

- Java version much cleaner

```
interface XMMS.Playback {
  public void playback(byte[] data);
}
```

- Adding new methods breaks compatibility!

http://openide.netbeans.org/tutorial/api-design.html

Co-Variance and Contra-Variance

Separate Client and Provider API

- Client API requirements are opposite to Provider API
- Very different and complicated in C
- Simple in object-oriented languages
  - Anything sub-classable is de-facto provider API
- Do not mix client and provider APIs

http://openide.netbeans.org/tutorial/api-design.html

New OutputStream Method

Separate Client and Provider API

- Can you add `write(ByteBuffer)` to OutputStream?

```
public void write(ByteBuffer b) throws IOException {
  throw new IOException(“Not supported”);
}
```

- Previous version complicates clients, but there is a way:

```
public void write(ByteBuffer b) throws IOException {
  byte[] arr = new byte[b.capacity()];
  b.position(0).get(arr);
  write(arr);
}
```

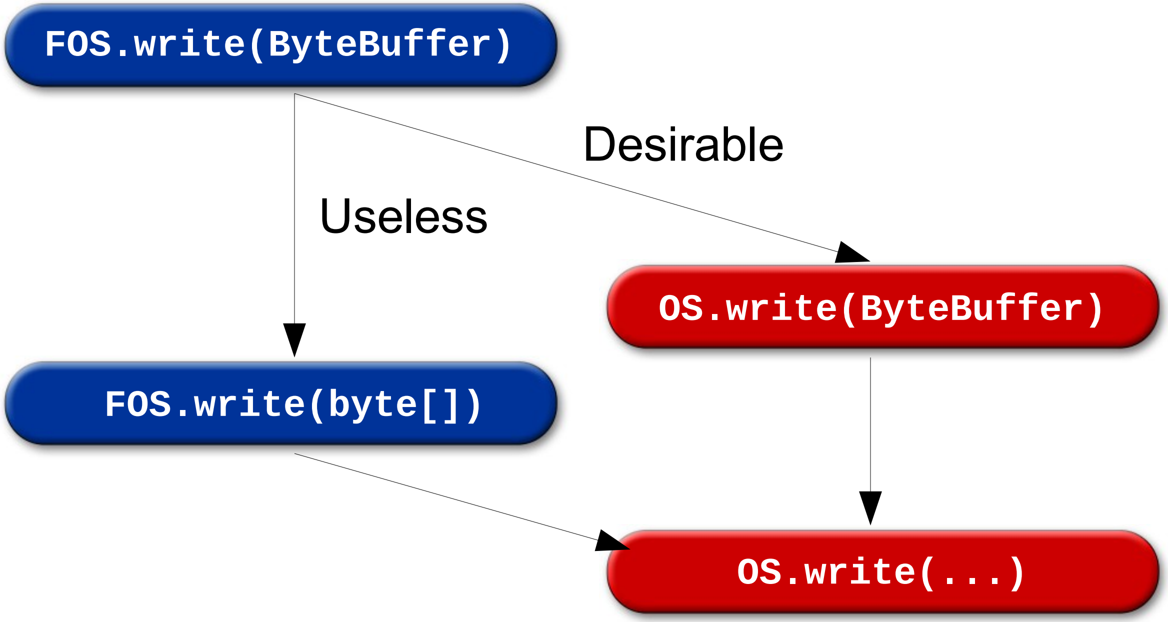
http://openide.netbeans.org/tutorial/api-design.html

The FilterOutputStream Problem

Separate Client and Provider API

- Shall `FilterOutputStream` delegate or call super?

```
public void write(ByteBuffer b) throws IOException {
  out.write(b); // super.write(b);?
}
```



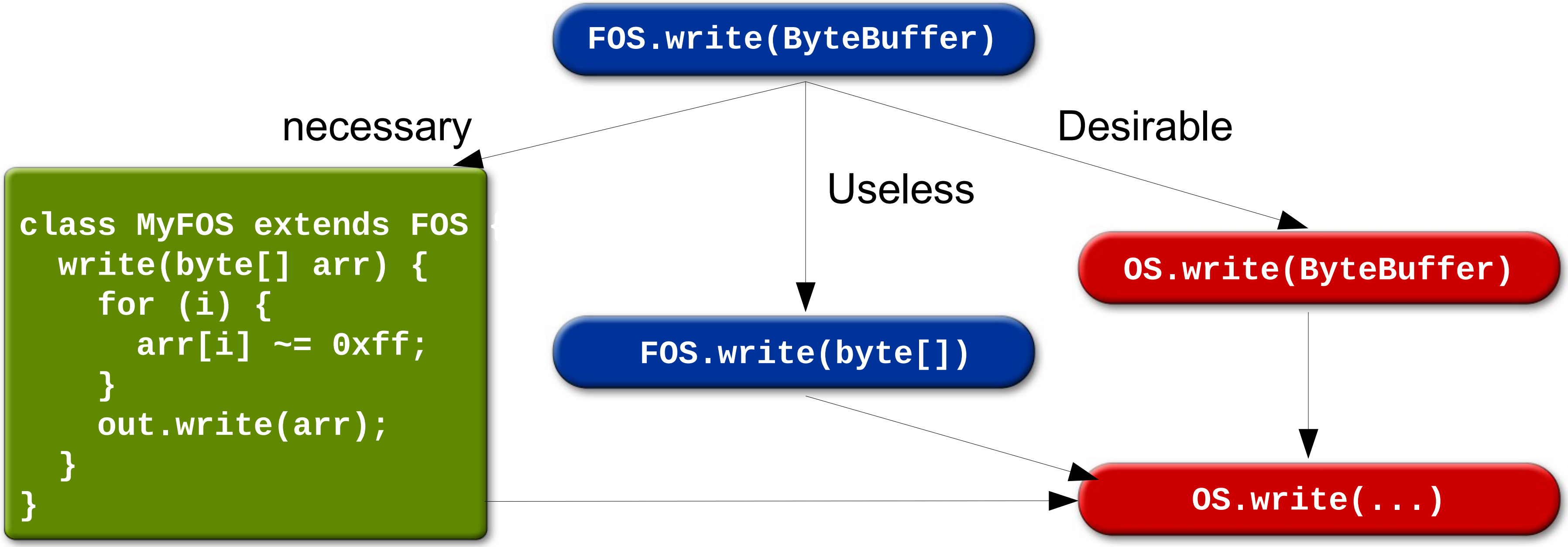
http://openide.netbeans.org/tutorial/api-design.html

The FilterOutputStream Problem

Separate Client and Provider API

- Shall `FilterOutputStream` delegate or call super?

```
public void write(ByteBuffer b) throws IOException {
  out.write(b); // super.write(b);?
}
```



FOS.write(ByteBuffer)

necessary

Useless

Desirable

OS.write(ByteBuffer)

FOS.write(byte[])

OS.write(...)

```
class MyFOS extends FOS {
  write(byte[] arr) {
    for (i) {
      arr[i] ~= 0xff;
    }
    out.write(arr);
  }
}
```

http://openide.netbeans.org/tutorial/api-design.html

Fixing FilterOutputStream Problem

Separate Client and Provider API

- Fixing existing problem
  - Delegate iff `FOS.write(ByteBuffer)` is not overridden
- Think about evolution during API design. For example:

```
public /*final*/ class OutputStream extends Object {
  private Impl impl;
  public OutputStream(Impl i) { impl = i };
  public final void write(byte[] arr) { impl.write(arr); }
  public interface Impl {
    void write(byte[] arr);
  }
  public interface ImplWithBuffer extends Impl {
    void write(ByteBuffer arr);
  }
}
```

http://openide.netbeans.org/tutorial/api-design.html

Separate Interface From Implementation

Modular Applications Are the Future

- Modular applications are not monolithic
- Testability of units
- Communication using well defined interfaces

```
public abstract class LicenseManager {
  public abstract boolean licenseAccepted(URL licese);
}

class DefaultLM extends LicenseManager { ... }
class TestingLM extends LicenseManager { ... }
```

http://www.netbeans.org/download/4_1/javadoc/usecases.html#usecase-Lookup

Lookup Your Implementation

Modular Applications Are the Future

- Inversion of control
  - application code does not care about the implementation
  - specified from outside

```
import org.openide.util.Lookup;
LicenseManager manager;
manager = Lookup.getDefault().lookup(LicenseManager.class);
manager.licenseAccepted(myLicenseURL);
```

- Different setup in tests and in runtime environment

http://www.netbeans.org/download/4_1/javadoc/usecases.html#usecase-Lookup

Foreign Code From Constructor

Anti-Patterns

- Accessing not fully initialized object is dangerous
  - Fields not assigned
  - Virtual methods work
- **`java.awt.Component` calls `updateUI`**
- **`org.openide.loaders.DataObject` calls `register`**
- Wrap with factories, make the constructors lightweight

http://openide.netbeans.org/tutorial/api-design.html

Foreign Code in Critical Section

Anti-Patterns

- Calling foreign code under lock leads to deadlocks
- Sometimes hard to prevent

```
private HashSet allCreated = new HashSet ();
public synchronized JLabel createLabel () {
  JLabel l = new JLabel ();
  allCreated.add (l);
  return l;
}
```

- **`java.awt.Component` grebs AWT tree lock**
- `HashSet.add` **calls** `Object.equals`

http://openide.netbeans.org/tutorial/api-design.html

Verification

- Mistakes happen
- Automatic testing of global aspects
  - Signature tests
  - Files layout
  - List of exported packages
  - Module dependencies
  - Automated tests
- Executed after each daily build

http://openide.netbeans.org/proposals/arch/clusters.html#verify

Summary

- Be client-centric
- Be predictable
- Always think about evolution
- Design to last

Q&A

Tim Boudreau

Jaroslav Tulach

How to Write APIs That Will Stand the Test of Time

Tim Boudreau and Jaroslav Tulach

Sun Microsystems
http://www.netbeans.org

TS-6218