
ORACLE®

JavaOne™

ORACLE®

API Design Checkli

Ten questions to ask before desi

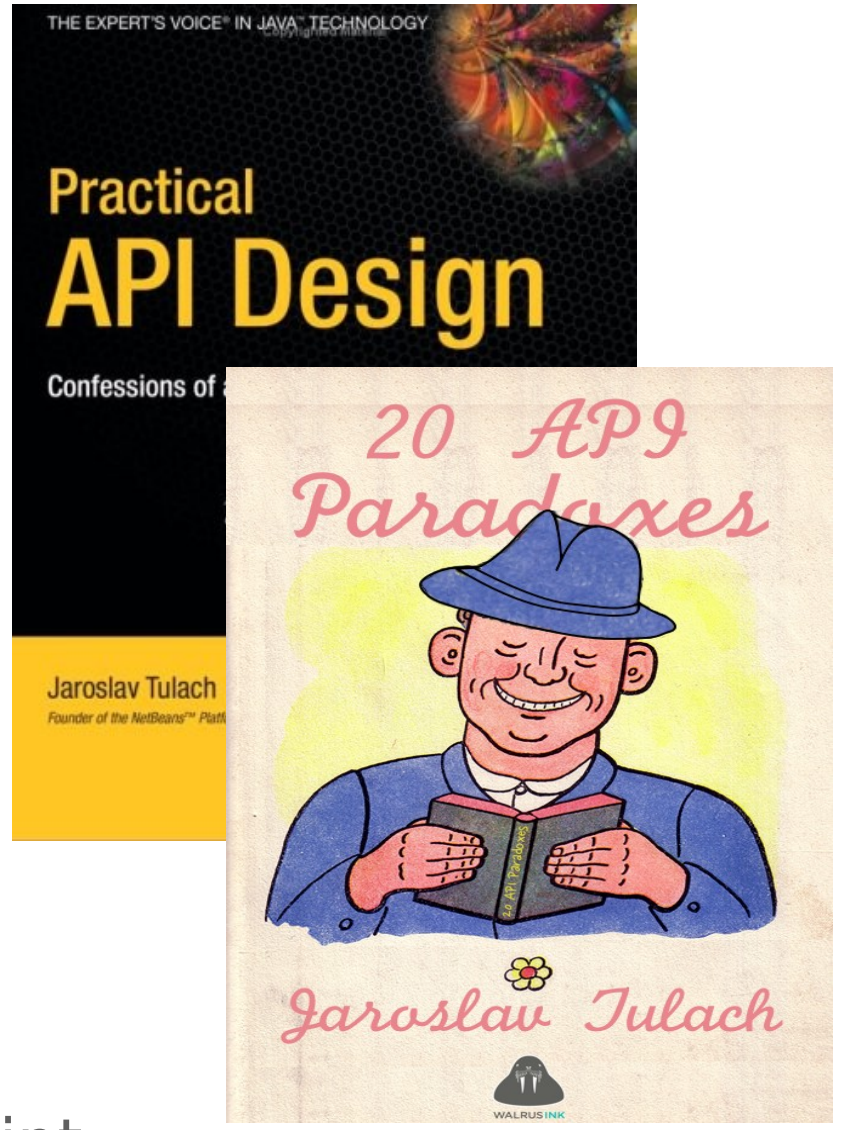
@JaroslavTulach
NetBeans Platform Architect
Oracle

Motto

Just like there is a difference between describing a house and describing a Universe, there is a difference between writing a code and producing an API.

About Me

- 1996 – Xelfi @ MatFyz
- 1997 – Initial NetBeans APIs
- 1999 – Acquired by Sun Microsystems
- 2008 - Practical API Design book
- 2010 – Acquired by Oracle
  - NetBeans & JDeveloper
- 2012 – 20 API Paradoxes book
- now – HTML/Java APIs – e.g. @DukeScript



Program Agenda

1. What is my API?
2. How does a good API look?
3. Is my API correct?
4. Is my API misleading its users?
5. Am I egocentric?

Program Agenda

6. Am I sustainer or developer?
7. Is my API easy to use?
8. How do I accept patches?
9. Do I hide own garbage?
10. Am I ready for future?

What is my API?

- Is API about REST and JSON?
  - Popular these days
  - “API is not enough, protocols are everything!” - upside down
- API is “**everything** somebody else may **depend** on”
  - Javadoc is not the only API you have
  - Files layout, properties, ports, protocols, behavior, memory
  - Private fields? Reflection? sun.misc package?
- Stability categories
  - Stable/Under Development/Friend/Private/Deprecated

How does a good API look?

- Coolness
  - Be able attract attention
- Time to Market
  - Ready for cluelessness
  - Productive Quickly – archetypes, wizards & tools
- Preservation of Investments
  - Backward compatibility
  - Track record important

Demo

Get started with @DukeScript wizard!

Is my API correct?

- It is correct because it exists!
  - Shouldn't the API satisfy some goal?
- Working backwards
  - Press release – high level description
    - http://wiki.apidesign.org/wiki/JerseyFaces
  - Manual – use-cases
  - FAQ – converting use-cases to action
  - The actual classes, methods & Javadoc
- Top-down verification

Demo

JerseyFaces press release

Is my API misleading its users?

- Things should have a single meaning
- Clarity of access modifiers
  - **public final**
  - **protected abstract**
  - **protected final**
- Clarity of types
  - Client API – use **final class**
  - Service Provider Interfaces – use **interface**
  - Evolution story

Demo

HTML/Java Audio API & SPI

Am I egocentric?

- Optimize for API writer or user?
  - One write vs. millions of users
- Does beauty matter?
  - API use should lead to beautiful code
- API needs to be stable & its internals reliable
  - Avoid visible refactorings
  - Be ready to hack
    - Bytecode patching
    - Dependency rewrites

Am I sustainer or developer?

- API design is art
  - No, it is engineering
- No repeating release cycles
- Creativity while working on first version
  - Prepare your evolution story
- Switch to sustaining mode
  - Compatibility #1 constraint
- Testing for compatibility
  - Binary: sigtest

Is my API easy to use?

- Usability study @apiusabilitytst
  - http://wiki.netbeans.org/Html4JavaUXStudy2014
  - Newcomer experience only
  - Can optimize “time to market”
- Comparing complexity of code
  - @DukeScript Java code shorter than original JavaScript
- Testability
  - @DukeScript application logic fully unit testable

Demo

Short and testable DukeScript code

How do I accept patches?

- Reaching limits of one's API
  - Force forks? Encourage hacks? Accept patches?
- I don't like your patch – artistic approach
  - Sure you don't as it is mine! But why wouldn't you accept it?
    - “Does not fit in the spirit”, “Would need careful review”, “Busy now”, etc.
- Your patch is not good enough – public API review process
  - Backward compatible
  - Documented
  - Tested
  - Ready for evolution

Do I hide my garbage?

- Javadoc for everything?
  - No, just for API packages
- Minimize conceptual surface
  - API classes should not expose implementation ones
  - Client API should not expose SPI interface
- Modules
  - Versioning
    - Semantic versioning – range dependencies
  - Enforcing public/private packages

Am I ready for the future?

- Your API is wrong!
  - Your API is definitely wrong!
- Get your evolution story right!
  - Client API needs new methods
  - Service provider interfaces cannot change
- There will be deprecations
  - Define reasonable end of life policy
    - Transfer API into different stability category
- Patching bytecode, classloading tricks & co.

The Checklist

- API Stability categories assigned
- Optimized for Time to Market
- Future press release written
- Criteria for accepting patches published
- API elements have a single meaning
- Ready to sacrifice myself not users
- API evolution story defined
- Implementation classes/packages hidden
- Testing signature compatibility
- End of life policy