
Documenting APIs: Your First Week on the Job

SF-STC

Jim Bisso, Bitzone LLC

jbisso@bitzone.com, 510.234.4046

Introduction

- Jim Bisso
 - technical communicator since 1988
 - specializing in developer documentation
 - co-author of *Documenting APIs: Writing Developer Documentation for Java APIs and SDKs*
 - instructor
 - computer science
 - technical writing
 - software engineer at Sun Microsystems

Overview

- User docs versus developer docs
- Developing a framework for developer docs
 - skills and domain knowledge —what you know
 - documentation deliverables—what you produce
 - resources—what you gather and discover
- Some examples

User Docs vs. Developer Docs

- Tech writer new to developer docs may feel lost when asked to document an API even though she or he
 - has years of experience in technical communication
 - has a methodology for writing user guides

Typical Methodology for Writing User Docs

- Learn the software and while learning
 - define audience/purpose for doc
 - develop user task list for the defined audience
 - note any problems in UI, etc., for further attention
- Define the document deliverables
 - write a doc plan with outline, etc., and circulate for review

Methodology for Writing User Docs (continued)

- Gather source documents
 - templates
 - earlier versions of doc deliverables
 - related docs, such as ppt. slides, competitor, etc.
- Write the alpha draft
 - applying best practices, write alpha draft
 - circulate for peer/technical review
- Revise, review, release, and maintain

What You Know

- Set of skills
 - the writer is an SME, too
 - how to
- Domain knowledge
 - “Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information on it.” -- Samuel Johnson

Skills

- More a set of skills than a list of procedures
 - problem-solving skills
 - heuristics, described in 1945 in Pólya's *How to Solve It*
 - top-down vs bottom-up
 - executive overview vs down in the trenches
 - breadth first vs depth first
 - big picture vs deep diving

Heuristics

- Pertaining to how something is discovered; Pólya described the steps in discovery as:
 - understanding the problem
 - making a plan
 - executing the plan
 - evaluating the solution
 - looking back
 - how could it be better?

Domain Knowledge

- Foundational
 - programming language(s)
 - technologies
- Transferable
- Category-generic
- Implementation-specific

An Example

- Foundational
 - Java
 - RDBMS
- Relational-object mapping
- JDBC API
- JDBC DB-specific driver

What you produce

- Developer doc plan
- Developer documentation
 - reference guide
 - programmer's guide
- Audience
 - developers
 - others

Developer Doc Planning

- Coverage
 - packages (or namespaces), classes, members
- Milestones
 - coordination with product
- Deliverables
 - paper
 - help
 - web

What you gather and discover

- Product: the API or SDK
- Subject matter experts (SMEs)
- Your own domain knowledge

Product

- Get to know your product
 - earlier versions
 - documentation
 - functional specifications
 - existing manuals
 - competing products
 - third party documentation

SMEs

- High level overview
 - architect
 - product manager, marketing & technical
- Details
 - engineering, deltas from functional spec
- Auxiliary
 - QA, testing plans
 - support: FAQs, forums

Another Example of Leveraging Domain Knowledge

- Design patterns
 - “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use the solution a million times over, without ever doing it the same way twice.” [Christopher Alexander]

Design Patterns

- Design patterns are a kind of foundational domain knowledge
 - helpful for both the writer and the audience (programmer) in understanding the API
- Easily expressed as a UML diagram that can be incorporated into the developer docs

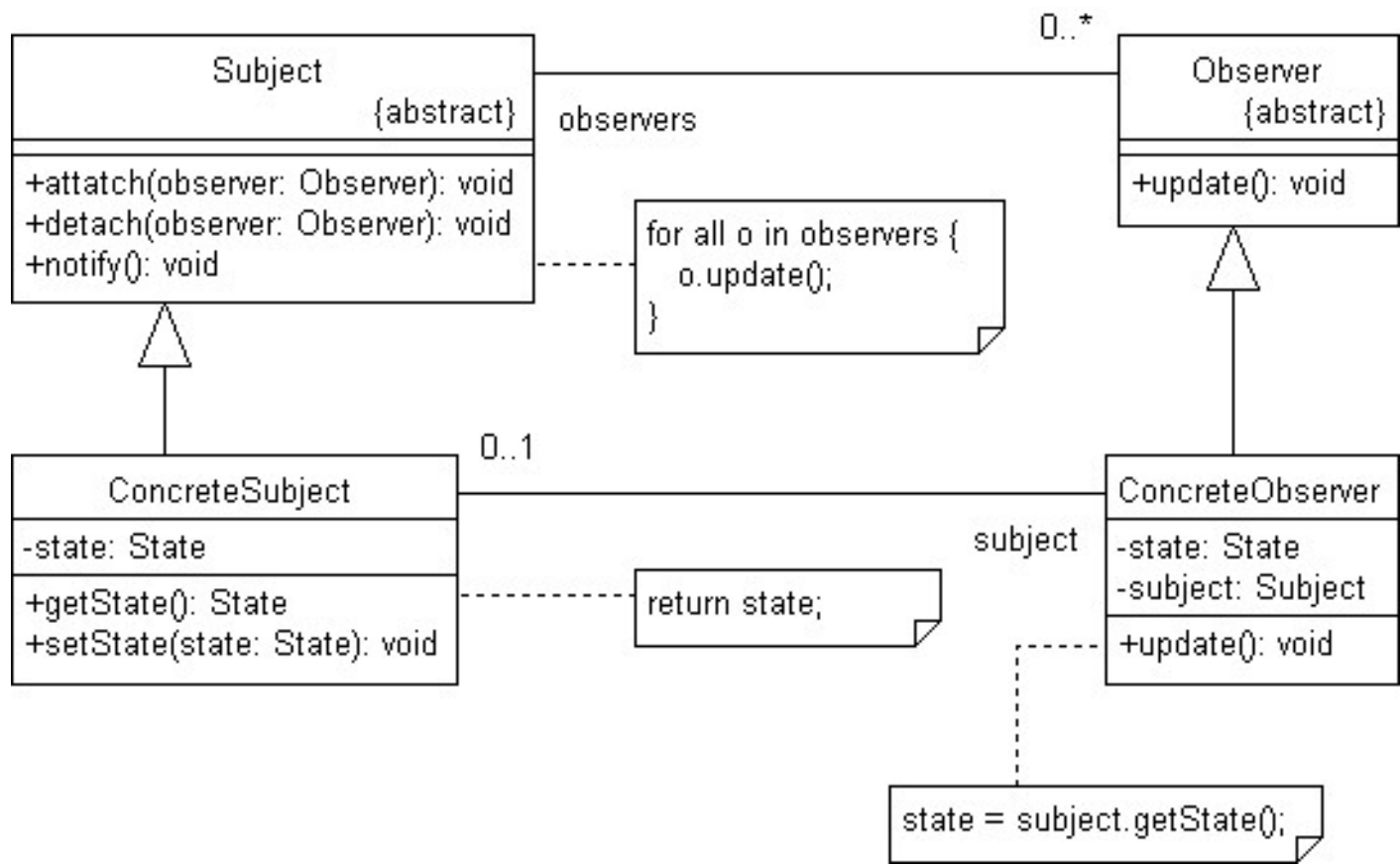
The Observer Design Pattern

- What is the Observer pattern for?
- The generic GoF design pattern
- Java-specific listener interfaces (in `java.awt.event`, et al.)

Example: Purpose

- Programs must respond to user input
 - Polling
 - object A wants to know when the state of object B has changed, so object A asks (polls) object B every so often - (does not scale)
 - Observer pattern
 - aka callback, publish/subscribe
 - object A registers with object B to call back object A when its (B's) state has changed - (scales well)

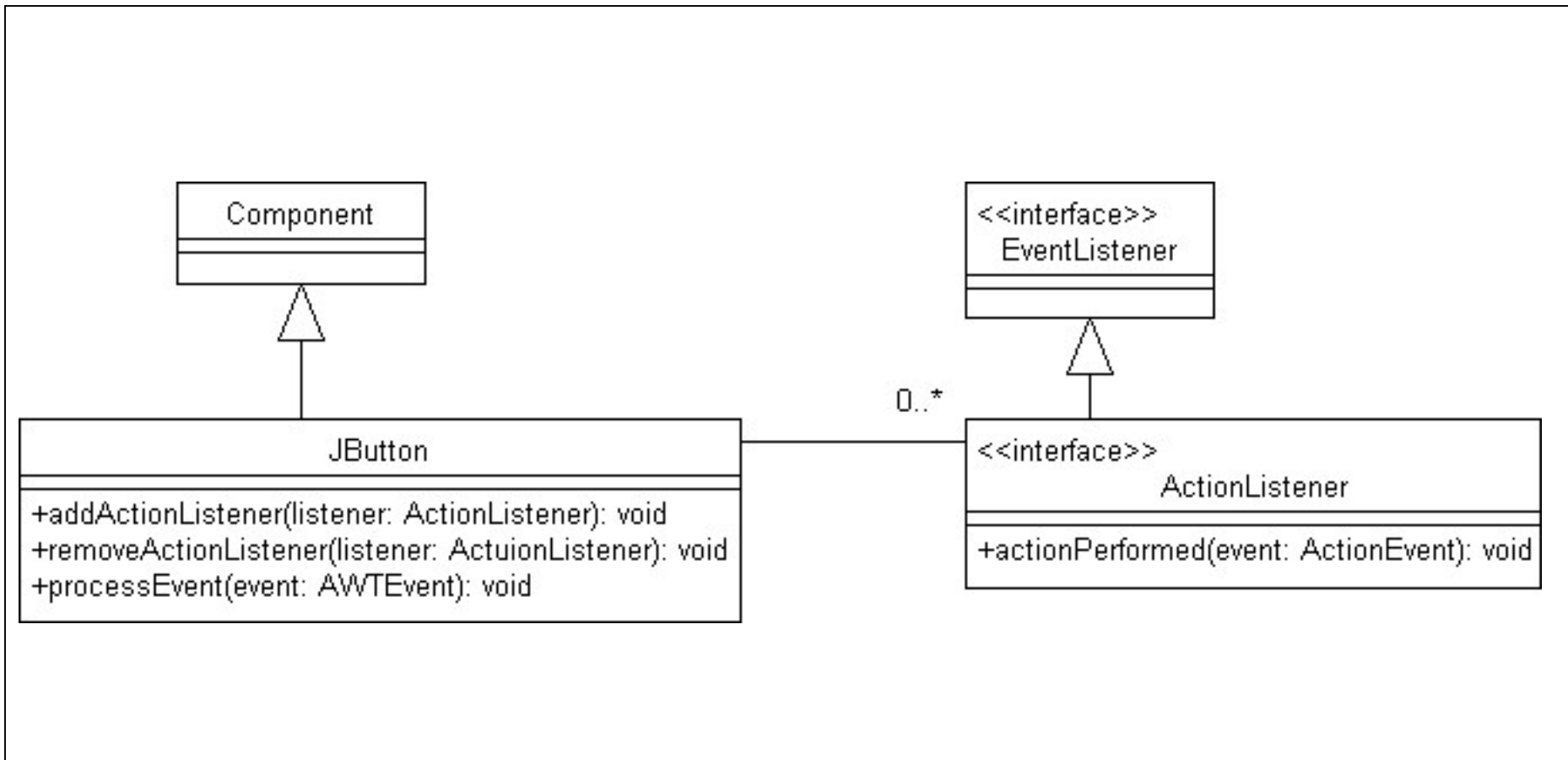
Example: GoF Pattern



Example: Listeners

- `java.awt.event` package
 - `ActionListener`
 - `KeyListener`
 - `MouseListener`
 - `MouseMotionListener`
 - (and a host of others)

Example: AWT Pattern



References

- Gang of Four, pp.293-303.
- <http://java.sun.com/j2se/1.5.0/docs/api/>
- <http://java.sun.com/docs/books/tutorial/uiswing/events/actionlistener.html>
- <http://c2.com/cgi/wiki?ObserverPattern>

Q & A

- Questions and comments