Documentation, Reusable Code, and Tooling

June 30, 2017

Reading Quiz

What is javadoc?

A. Improved, multilingual documentation in Java8

B. A system for generating documentation based on code comments

C. A Java 8 feature to access your documentation at runtime

D. A system for generating code based on documentation

Which is a reason for using Javadoc?

A. It makes it easier to keep your documentation in sync with your code comments

B. It makes it easier to keep your code comments in sync with your program's functionality

C. It makes it easier to type your code comments

D. It ensures users of your code will read your documentation

Which is **not** an issue when building reusable, cross platform code?

A. Different conventions for line endings

B. Using `Runtime.exec` to call subprocesses / other programs

C. Architectural differences in the JVM

D. Hardcoded path names in programs

Using Java naming conventions, what is a good method name?

A. THIS_ONE_RIGHT_HERE()

B. this_second_option_here()

C. thirdOptionCouldBeRight()

D. FourthOptionOfCourseDuh()

Using Java naming conventions, what is **THIS_THING_HERE** probably?

A. a "static final" property (ie a constant)

B. a private method

C. a nested class

D. a static constructor

Done!

The Plan

- One more note on Lambda
- Writing reusable libraries
  - Javadoc
  - Checkstyle
  - Pmd
  - Rules for libraries
- Homework 4

Javadoc

The Problem

- Two types of documentation
  - Implementation description (ex, code comments)
  - Usage description (ex PDF, HTML, etc)
- Problematic
  - Typing documentation is tedious (so less documenting)
  - More things to keep in sync (can introduce and security hazards)

Javadoc Solution

- A standard for commenting Javacode
- A tool to convert those comments into easer-to-read comments
- Write comments once, read wherever

Javadoc Example

```
/**
 * Class for demonstrating javadoc
 */
public class JavadocExample {

  /**
   * An example of a method name
   *
   * <p>Here is a longer description of what this does.
   *
   * @param input boolean a true/false input to the method.
   *
   * @return String A string version of the boolean value.
   *
   * @throws ExceptionName An exception that gets thrown
   *
   * @see Some other method or thing to look at
   */
  public String example(boolean input) {
  }
}
```

javadoc/RamonesFacts.java –>

CheckStyle

Problem

- Code is read more than written
- English (and other languages) have standards to ease readability
- We can do the same in Java to reduce errors

Code Standard Example

THEROMANSWROTELATIN
ALLINUPPERCASEWITH
NOWORDBREAKS
ORPUNCTUATION

checkstyle/Style.java –>

```c
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

Coding Standards Help

- Google's
  https://google.github.io/styleguide/javaguide.html

- Oracle's (outdated)
  http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Example Rules

- Always indent with the block
- Spaces over tabs (and just two of them!)
- Curlies go on the same line
- Spaces around operators
- Variable naming conventions
- Don't fall through in "switch" statements

checkstyle/Style.java –>

PMD

- Checks more than formatting
  - Complicated functions
  - Unused variables
  - Unused imports
  - Dead code

Writing Libraries

Libraries are "different"

- Provide functionality instead of using it
- Shared across projects
- General intent, instead of specific
- Main - Application, Library - Everything else

Goals for Libraries

- Be general
- Each library should have a single theme
- Each class should have a specific purpose
- Inform application about errors
- Never die / crash / dump stack / etc

Goals Continued

- Hide as much as possible from the application
- Private properties
- Minimally visible methods, classes

library/{Emojifier, Main}.java –>

Where Should it Go?
Application or Library?

- Error messages
- System.exit / status codes
- Documentation
- Defining exceptions
- User interface definition
- Text for user

JSON?

A. I've used it a lot

B. I know what it is, but haven't used it

C. I've heard of it, but don't really know what it is

D. I've never heard of it

YAML?

A. I've used it a lot

B. I know what it is, but haven't used it

C. I've heard of it, but don't really know what it is

D. I've never heard of it

XML?

A. I've used it a lot

B. I know what it is, but haven't used it

C. I've heard of it, but don't really know what it is

D. I've never heard of it

CSV?

A. I've used it a lot

B. I know what it is, but haven't used it

C. I've heard of it, but don't really know what it is

D. I've never heard of it

Homework 4

- Comandline tool for playing with JSON data
- Data from city of Chicago https://data.cityofchicago.org/Transportation/Red-Light-Camera-Violations/spqx-js37
- No loops!