Professional Issues II
Unit 2.2: code commenting
Lab preparation

Markus Roggenbach

February 2015

Recap: two different kinds of comments

Documentary comments:

- To provide basic information on the file.

Program structure descriptions:

- To explain how the program works
  up to now: method declarations
  MR's rule book requires 4–5 different comments

You will learn

How to realize the concepts of

- documentary comments and
- program structure descriptions

with the constructs provided by JavaDoc and HTML.

This will be a first example that concepts and constructs are not necessarily congruent to each other.

Javadoc

What is it?

Javadoc is a documentation generator [. . .] [originally] from Sun Microsystems for generating documentation in HTML format from Java source code.

The "doc comments" format used by Javadoc is the de facto industry standard for documenting Java classes.

from Wikipedia

The link: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

A first sample run

```
import java.io.*;
/**
* @author Markus Roggenbach
* - no copyright
* @version 1.0
*/
    public class Hugo {
    /**
    * Erna does nothing
    * @param i is pointless
    * @return we don't compute anything
    */
    public static boolean Erna (int i){
        return true;
    }

    /**
    * This program writes "hugo".
    */
    public static void main (String args[]) {
        System.out.println("hugo");
    }
}
```

The javadoc command

- `javadoc -d <directory-name> -version -author <name-of-the-java-program>`

Available Tags

`@author`, `{@code}`, `{@docRoot}`, `@deprecated`, `@exception`, `{@inheritDoc}`, `{@link}`, `{@linkplain}`, `{@literal}`, `@param`, `@return`, `@see`, `@serial`, `@serialData`, `@serialField`, `@since`, `@throws`, `{@value}`, `@version`

Comments are written in HTML and can use HTML tags.

Important tags

`@author name-text`

Adds an "Author" entry with the specified name-text to the generated docs when the -author option is used.

`@version version-text`

Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.

`{@code text}`

Displays text in code font without interpreting the text as HTML markup or nested javadoc tags. This enables you to use regular angle brackets (< and >) instead of the HTML entities (< and >)

`@param parameter-name description`

Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section.

`@return description`

Adds a "Returns" section with the description text. This text should describe the return type and permissible range of values. This tag is valid only in a doc comment for a method.

HTML

What it is

HTML, an initialism of HyperText Markup Language, is the predominant markup language for Web pages. It provides a means to describe the structure of text-based information in a document.

`from Wikipedia`

Some HTML tags

- Text Formatting

```
<p>This is a paragraph</p>
<br> (line break)
<hr> (horizontal rule)
```

- Unordered list

```
<ul>
<li>First item</li>
<li>Next item</li>
</ul>
```


- Ordered list

```
<ol>
<li>First item</li>
<li>Next item</li>
</ol>
```

from http://www.w3schools.com/html/default.asp

Example: Stacks

What is it?

How to manipulate it

- put something on top – “push”
- look what is on top – “top”
- take the top item away – “pop”

Application:
Main implementation strategy in compilers for method calls

Stacks are "generated" by `push` and `empty`

There is an empty stack: `empty`.

Pushing an element on a stack leads to a "bigger" stack:

```
                          empty
                 push(1, empty)
        push(2, push(1, empty))
push(3, push(2, push(1, empty)))
...
```

If there is not enough memory left:
`push` leads to an ``error'' message.

Stacks are "observed" by top

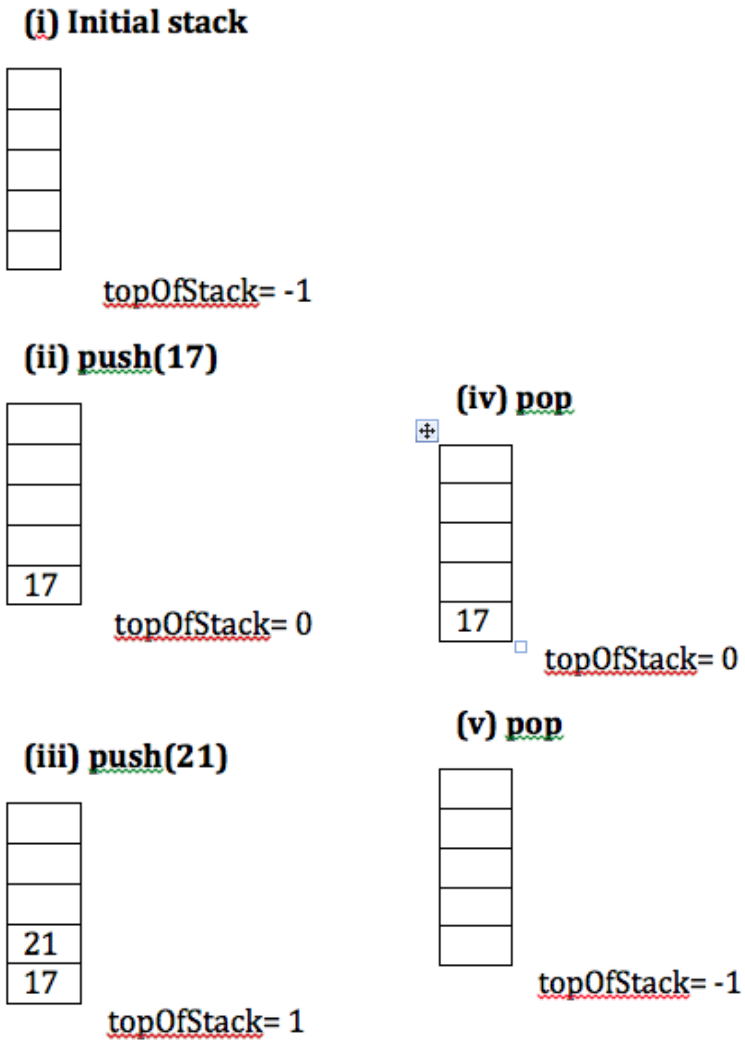
```
top(                              empty)    = "error"
top(                   push(1, empty)       = 1
top(          push(2, push(1, empty))       = 2
top(push(3, push(2, push(1, empty))))       = 3
...
```

Stacks are "observed" by pop

pop takes off one element from a stack:

```
pop(                              empty)    = "error"
pop(                    push(1, empty)    = empty
pop(          push(2, push(1, empty))   = push(1, empty)
pop(push(3, push(2, push(1, empty)))) = push(2, push(1, empty))
...
```

Implementation with array & topOfStack



Avoiding use of a 'broken' stack

Use a stack of limited size can lead to mistakes:

- push on a full stack
- pop on an empty stack

One can use a boolean variable `errorFree` to signal that these 'abuses' have not appeared in a stack's history:

- In the beginning: `errorFree` is set to `true`.
- Should an error occur: set `errorFree` to `false`.

Lab-classes

Working Pattern

- Lectures: *Preparation for the lab-class*
  - Comments (documentary, commenting method declarations)
  - Technology (Javadoc, Html)
  - Example: Stack of integers
- Labclass: *Playing with tools* (still to come)
  - Understanding the stack implementation
  - Write comments using Javadoc
- Lectures: *Reflection* (still to come)
  - Discussion of common mistakes (if any)
  - Model solution

What you have learned in this unit

Definitions

- JavaDoc
- Html

You should be able to explain by example

- What JavaDoc tags exist.
- The data type stack.
- How to implement a stack with three variables: an array, a `topOfStack` pointer, an `errorFree` flag.