

LECTURER: JOHN DOE

DATA STRUCTURES AND JAVA CLASS LIBRARY

INTRODUCTORY ROUND

Who are you?

- Name
- Employer
- Position/responsibilities
- Fun Fact
- Previous knowledge? Expectations?



TOPIC OUTLINE

Programming Style

1

Working with Objects

2

External Packages and Libraries

3

Data Structures

4

Strings and Calendar

5

File System and Data Streams

6

UNIT 1

PROGRAMMING STYLE

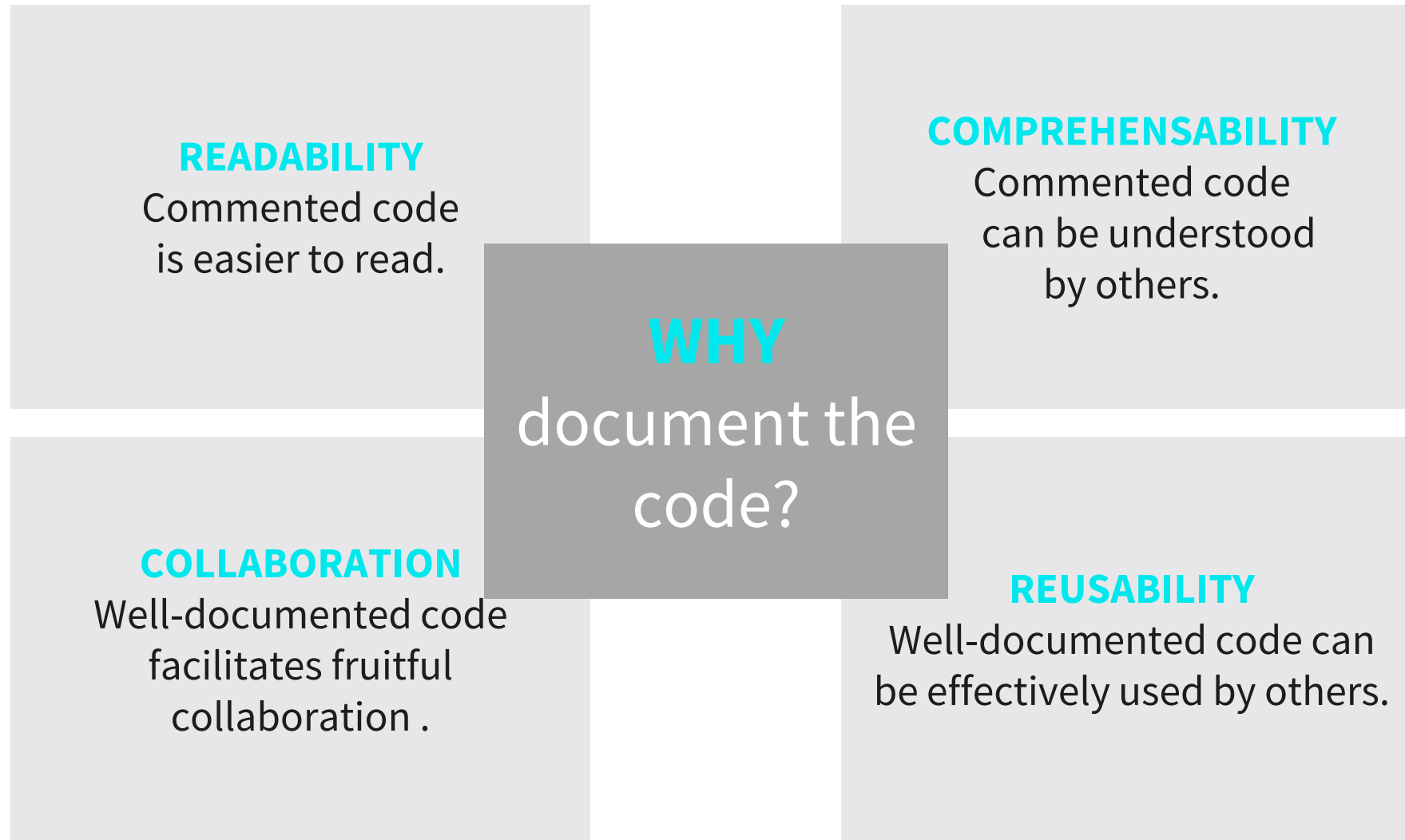


- Learn to use comments to document code.
- Learn to use **Javadoc** to automatically generate code documentation.
- Learn to use **Java** annotations for providing the compiler with useful additional code information.
- Understand guidelines and best practices for structuring, formatting and naming classes, methods and attributes.



1. How can comments and Javadoc be used to document and describe Java code?
2. Describe the most common Javadoc tags and their purpose of use.
3. Why are code conventions important?

WHY COMMENT AND DOCUMENT THE CODE?



HOW TO COMMENT AND DOCUMENT CODE

WRITE DOCUMENTATION IN COMMENTS

Use comments to provide explanations directly in the source code.

USE JAVADOC

Use standardized commenting for automatic generation code documentation.

PROVIDE CODE ANNOTATIONS

Add programmatically useful additional information through code annotations .

USE CODE CONVENTIONS EFFECTIVELY

Rely on best practices to structure and format the code effectively.

WRITE DOCUMENTATION IN COMMENTS

- **Comments** are used to describe and document source code.
- They are special characters informing the compiler that the following words should not be interpreted as code.
- Java allows for different ways to denote comments.



Single-line comments denoted by a double slash (“//”).



Multi-line comments start with a forward slash and an asterisk (“/*”) and end with an asterisk and forward slash (“*/”).



Bracketed comments ([...]) allow longer explanations directly in the code (often used to document classes, methods and attributes).

WRITE DOCUMENTATION IN COMMENTS

An example of single and multi-line comments describing a class, its attributes and methods:

```
[...]
/*
 * This class is the introduction point to the program.
 * It contains the <code>main</code> method, which is used for
 * starting the program. The <code>main</code> method creates
 * automatically all necessary objects and executes all tasks.
 */

public class Main {
    /* The used Online Shop. */
    public OnlineShop onlineShop = null;

    /*
     * Creates a new book with author, title, manufacturer
     * and stock, informs the administration
     * objects and returns the created book.
     */
    public Book newBook(String author, String title,
                        String manufacturer, int stock)
    {
        [...]
    }
}
```

Class comments describe classes

Attribute comments describe attributes

Method comments describe methods

- **Javadoc** allows to automatically generate documentation from code that is commented following some standard notation.
- Comments must be introduced with a slash and two subsequent asterisks (“/**”). Javadoc uses special tags (starting with ‘@’) to generate HTML documentation pages.

1

@author denotes the author of a file

2

@param describes a method's parameter

3

@return describes what the method returns

4

@throws lists possible error sources and thrown exceptions

DOCUMENTING CODE USING JAVADOX

- An example of Javadoc-style commenting which leads to automatic code documentation as HTML pages:
- Javadoc commenting and documentation can be done inside the programming environment.

```
[...]
/**
 * This class is the introduction point to the program.
 *
 * It contains the <code>main</code> method, which is used for
 * starting the program. The <code>main</code> method creates
 * automatically all necessary objects and executes all tasks.
 *
 * @author Christian Coder
 * @version 1.0
 */
public class Main {

    /**
     * The used Online Shop.
     */
    public OnlineShop onlineShop = null;

    /**
     * Creates a new book and informs the
     * administration objects.
     *
     * @param author Name of Authos of the new book
     * @param title Title of the new book
     * @param manufacturer Manufacturer of the new book
     * @param stock Quantity of the new book in stock
     *
     * @return new book instance
     *
     * @throws RuntimeException if updating the stock fails
     */
    public Book newBook(String author, String title,
                        String manufacturer, int stock)
    {
        [...]
    }
}
```

Javadoc comments start with a slash and two subsequent asterisks

Class comment

Javadoc tags to specify author and version of the class

Method comment

Javadoc tags to describe parameter, return values, and possible exceptions

CODE ANNOTATIONS

- Code **annotations** allow to add additional information to Java source code which can be programmatically useful.
- Annotations start with the **@** sign and can be used to annotate classes, methods and attributes.
 - **@Override** is a common annotation used to mark methods that override the methods of a superclass.
 - **@Deprecated** allows to mark classes, methods and attributes that are obsolete, thus, should not be used in new code.
- Annotations help developers to recognize errors more quickly.

DOCUMENTING CODE USING JAVADOX

- Inherited method
getDescription() is overridden,
as denoted by the respective
annotation.
- Method setPages() is replaced
by method
setNumberOfPages(), thus
becoming obsolete.

```
public abstract class Book extends Product {  
    [...]  
    @Override  
    public String getDescription() {  
        return super.getDescription()  
            + " of " + author;  
    }  
  
    @Deprecated  
    public void setPages (int pages) {  
        this.pages = pages;  
    }  
  
    public void setNumberOfPages (int pages) {  
        this.pages = pages;  
    }  
    [...]  
}
```

Explicit information
that a method was overridden

Annotation that a method is outdated

CODE CONVENTIONS

- Code **conventions** provide a standardized way of structuring and formatting the code, thus increasing its readability.
- Common Java code conventions include:



writing package names in lowercase letters

Name classes and interfaces as nouns starting with a capital letter.

Use **camel case** when the name consists of multiple words.

When possible, name methods as verbs. Write them in lowercase or camel case.

Fully capitalize constants. Name attributes and variables meaningfully.

CODE CONVENTIONS

- When structuring code, items that belong together should also appear together. A common order includes:



package declarations and imports (alphabetically sorted)

class variables and constants

instance variables

constructors

getters and setters and after them other additional methods

CODE CONVENTIONS

- Separating code blocks from one another increases readability.
- Other conventions include:
 - Declaring variables at the beginning of a code block.
 - Write each declaration on a separate line.
 - Use blank lines to group semantically relevant sections.
 - Indent the code properly through the usage of curly brackets ({ }).

The diagram illustrates Java code conventions with the following annotations:

- Package names are always written in lower case:** points to `package onlineshop.products;`
- Class names in Camel case:** points to `public class NonfictionBook`
- For method names, the Camel case is also used:** points to `public void setSubject`
- Class constants are completely capitalized:** points to `CLASS_PREFIX = "NF";`
- Method names are formulated as verbs and written in lower case:** points to `public String summarize`

```
package onlineshop.products;

public class NonfictionBook extends Book {
    private static final String CLASS_PREFIX = "NF";
    private String subject;
    public void setSubject(String subject) {
        this.subject = subject;
    }

    public String summarize() {
        [...]
    }
}
```



- Learn to use comments to document code.
- Learn to use **Javadoc** to automatically generate code documentation.
- Learn to use **Java** annotations for providing the compiler with useful additional code information.
- Understand guidelines and best practices for structuring, formatting and naming classes, methods and attributes.

SESSION 1

TRANSFER TASK

TRANSFER TASK

- 1) Consider the provided code defining an Employee class containing.

Comment the class accordingly using Javadoc style and using special tags such as `@author`, `@param`, `@return` and others.

- 2) **Literature research:** review literature to identify a set of the most used Javadoc tags. Discuss your findings with a classmate.

TRANSFER TASK
PRESENTATION OF RESULTS

Please present your
results.

The results will be
discussed in
plenary.





1. What is the purpose of comments?
 - a) Comments document the requirements for the system to be developed.
 - b) Comments summarize the meaning and purpose of lines of code and facilitate understanding of the program code.
 - c) Comments control the execution of a program.
 - d) Comments give the compiler additional information for the compilation process.



2. Javadoc is...

- a) ... the online documentation of the Java class library.
- b) ... used to describe maintenance work on the source code.
- c) ... a program and commenting convention.
- d) ... a debugging tool that automatically detects errors in the source code.



3. In which order are the elements of a class typically arranged?

- a) static variables and methods, instance variables, constructors, getters and setters, instance methods
- b) static variables and methods, instance variables, instance methods, constructors, getters and setters
- c) getters and setters, static variables and methods, instance variables, instance methods, constructors
- d) instance variables, getters and setters, constructors, instance methods, static variables and methods

LIST OF SOURCES

Oracle. (2019a). *How to write doc comments for the Javadoc tool*. Oracle. Retrieved from <https://www.oracle.com/technetwork/java/javase/tech/index-137868.html>

© 2022 IU Internationale Hochschule GmbH

This content is protected by copyright. All rights reserved.

This content may not be reproduced and/or electronically edited, duplicated, or distributed in any kind of form without written permission by the IU Internationale Hochschule GmbH.