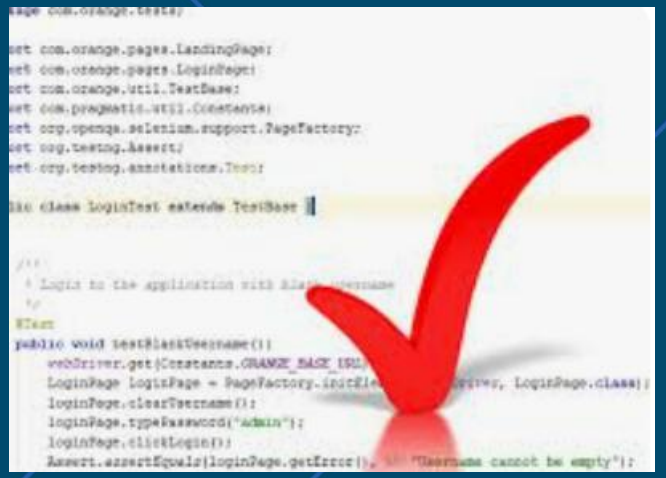
Féidearthachtaí as Cuimse
Infinite Possibilities



# JavDoc and Coding Standards Week6

Object Oriented programming

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

# 80% of the cost of software

- Is on s/w maintenance

- Your code needs to be
  - readable
  - understandable
  - follow the language naming standards

# Documentation

JavaDoc is a  documentation tool built into the JDK

Converts special comments into HTML documentation.

Build API documentation

Uses tags like @param, @return,  @author. @version

Example:

```
/**
 * Calculates the area of a circle.
 * @param radius the radius of the circle
 * @return the area as a double
 */
```

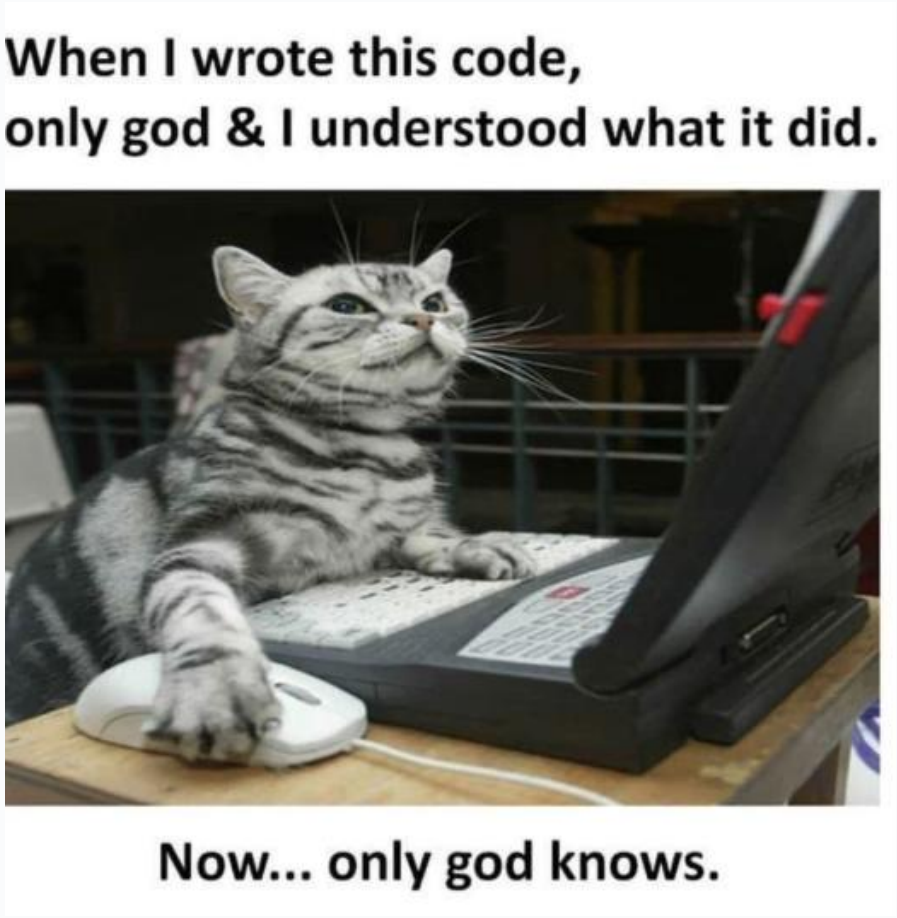
# Similar to other comments

- // This is a single line comment
- /* This is a regular multi-line comment */
- /** This is a Javadoc */

# Where Javadoc Is Used

The Javadoc comment must be placed **immediately before** the declaration of the class, interface, method, constructor, or public/protected field you are documenting.

# Why Use It ?




https://www.linkedin.com/feed/update/urn:li:activity:7381784396204285953?utm_source=share&utm_medium=member_desktop&rcm=ACoAABZa8jgB7OVZKMgcz_La1vF0ObnmNd3YOqg

# Anatomy

- 2 parts – Description and Block Tags

**Short Summary:** The first sentence (used in index summaries). It should be a clear, concise verb phrase for methods.
**Longer Body (Optional):** Detailed explanation, behavior, side effects, or constraints. Use HTML tags like `<p>` for paragraphs.

Structured metadata starting with @ (e.g., @param, @return). They must follow the description.

# Javadoc tags

| Tag | Description |
| --- | --- |
| `@param` | Describes a parameter |
| `@return` | Describes what the method returns |
| `@throws` | Lists exceptions thrown |
| `@author` | Who wrote the class |
| `@version` | Version information |
| `@see` | Reference to related code |

## Example format

```
/**
 * Short, one-sentence description of the class/method.
 * <p>
 * This is the optional longer description. You can include more
 * detail here about how the code works, what its side effects are,
 * or any important context. You can use standard HTML tags like
 * <p> for new paragraphs, <ul> for lists, and <code> for code.
 *
 * @param parameterName description of the parameter.
 * @return description of the return value.
 * @throws ExceptionName if a specific error condition occurs.
 */
public int calculateValue(String parameterName) throws ExceptionName {
    // ... method implementation
}
```

# Writing Javadoc in Eclipse

1. Place cursor above a class or method
2. Type /** then press **Enter**
   → Eclipse generates the Javadoc template automatically

3. Or Source → Generate Element Comment

```
/**
 *
 * @param name
 * @return
 */
```

But you have to fill in the descriptions manually

# What to write? Class

| Section | Focus | Javadoc Tags Used |
| --- | --- | --- |
| **Short Summary** | **The main purpose.** What does this class represent or provide? | N/A (First sentence) |
| **Longer Body** | **The concept and design.** Explain the object's lifecycle, threading concerns, immutability, or how it relates to other classes. | HTML tags, `{@link}` |
| **Metadata** | **Who, when, and version.** Standard housekeeping information. | `@author`, `@version`, `@since`<br>`@see` |

# What to write?

- Class level Comments:
- **Javadoc Rules to Live By**
- **Be Concise:** The first sentence is the summary—make it count!
- **Focus on the Contract:** Describe *what* the element does, not *how* you implemented it.
- **Document Public API:** Only document public and protected classes, methods, and fields. Leave implementation details to standard comments.

# Make sure

- Follow Java naming standards.
- Oracle has their (archived) conventions

https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

- AND - basics
  - Indentation
  - aligned { }
  - enough, but not too many comments  (think JavaDocs
  - Comment headers
  - /*******************************
  - This class was developed.. etc
  - *******************************/

# Generation

- Demo
- Eclipse  Project → Generate JavaDoc


Coding
Standards

# Specifics – and use in your assignment

| Name | Convention |
| --- | --- |
| **class name** | **should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.** |
| **interface name** | **should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.** |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| **variable name** | **should start with lowercase letter e.g. firstName, orderNumber etc.** |
| **package name** | **should be in lowercase letter e.g. java, lang, sql, util etc.** |
| **constants name** | **should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc** |

# Coding Principles/Standards

- 🔹 **Clarity Over Cleverness** – Use names that clearly express purpose; avoid vague or generic ones.
  - 🔹 **Be Consistent** – Follow the same naming patterns throughout your codebase.
  - 🔹 **Reveal Intent** – Good names explain *why* code exists, often removing the need for comments.
  - 🔹 **Keep It Concise** – Be descriptive but brief; avoid long, cluttered names.
  - 🔹 **Limit Abbreviations** – Use only well-known ones (e.g., HTML, API).
  - 🔹 **Follow Conventions** – Use camelCase or snake_case as per your language or team standard.
  - 🔹 **Get Feedback** – Ask teammates when unsure; clarity benefits everyone.
  - 🔹 **Avoid Collisions** – Don't reuse names with different meanings.
- 💡 **Remember:** Code is read far more often than it's written.

https://www.linkedin.com/feed/update/urn:li:activity:7381784396204285953?utm_source=share&utm_medium=member_desktop&rcm=ACoAABZa8jgB7OVZKMgcz_La1vF0ObnmNd3YOqg

# Self documenting code

```
public void rcalc(int a){
        r = a / 2;
        while ( abs( r - (a/r) ) > t ) {
              r = 0.5 * ( r + (a/r) );
        }
        System.out.println( "r = " + r );
}
```

```
public void calSquareRoot(int num)
{
        root = num/ 2;
        while ( abs(root - (num/ root) ) > t )
       {
              r = 0.5 * (root + (num/ root));
        }
           System.out.println( " root = " +
             root );
}
```