Documentation of Software

- Types of documentation
- The problems related to documentation
- The purpose of the JavaDoc tool (or utility)
  - Limitation
- Guidelines for JavaDoc from SunSoft
- Non standard JavaDoc comments
- Summary

Types of Documentation

- Analysis and design (high-level and low-level) documentation
  - What you are doing in object-oriented analysis and design documents
  - Used to provide an overview of the system

- System documentation
  - Provides all the (low-level) details
  - Reference manual type of documentation
  - This is what we emphasizing in today's lecture

- Test documentation
  - High-level descriptions of the test approach
  - Low-level documentation JavaDoc-like

Problems Related to Documentation

- Documentation is competing hard with test to be the part most often missing from a software project.
  - „Use the source, Luke!“ [The motion picture Star Wars]
- Programmers are often reluctant to write documentation.
  - The job of the most recently hired programmer
- Documentation is often the last thing done in a project
  - If it is done at all, „it takes too long to write documentation!“
- Documentation gets outdated
  - Documentation is for version 1.0 of the system which is currently now release in version 5.2.
  - Source code in one file, documentation in another file
- Documentation is incomplete
  - Not all parts of the system are documented

Problems Related to Documentation, cont.

- Documentation is incorrect
- It looks ugly
  - Text file
  - Multiple different Word or HTML templates
- The documentation is not uniform
  - Where should I start to look for information?
  - Some document method, others also the data, and a final group only constructors. They all forget to document the return value of functions.
- Does not provide an overview
  - Find documentation for that single method your are currently having problems understanding

JavaDoc Purposes

- Makes the documentation look nice
  - Use HTML
- Makes the documentation look uniform across all programs
  - If you know the structure of the documentation for one program, you know it for all programs!
- Makes the documentation fast to write
  - Auto generated from source files
- Makes it easy to browse the documentation
  - Extract overview and use HTML, e.g., hyper links to different parts
- Solves the problem that documentation written last
  - Written concurrently with the program
- Solves the problem that documentation gets outdated
  - Documentation is embedded into the source code

JavaDoc Purposes, cont.

- Solves documentation not complete
  - Can check e.g., all method documented
- Solves (partly) documentation is incorrect
  - Simple checks that, e.g., all parameters and return value have documentation strings

Limitations of JavaDoc

- Only system documentation, i.e., reference manual type
- No-magic, the programmer still have to write the documentation strings that are extracted via the JavaDoc tool
  - Development environments will help you provide JavaDoc skeletons
- Only partly check of consistency between source and documentation
  - check documentation string for all parameters and return value
  - check all exception documented

JavaDoc, Example

```
import com.mycompany.misc.*;
/** Implements a garage of cars and trucks */
public class Garage {
     /** Comment on instance variable */
     Car car1;
     Truck truck1;

     /** Default constructor, create a car and a truck */
     public Garage(){
          /** Comment on method internals */
          car1 = new Car();
          truck1 = new Truck();
     }

     /** Prints the vehicles made in the constructor
      * @see Garage#Garage()
      */
     public void toPrint(){
          System.out.println ("A garage: " + car1 + truck1);
     }
}
```

JavaDoc Tags

- A tag is a special keyword within a comment that JavaDoc can process (note case sensitive)
- Types
  - Block tags, must appear at beginning of line, e.g., @author
  - Inline tags, anywhere must be within curly bracket, e.g., {@link}

- User-defined cascading-style-sheet for JavaDoc can be provided
  - To give a company-specific look to the documentation
- Tags can be user-defined via Doclets
  - Java program using the doclet API
    - e.g., @todo, @extension, @bug, @workaround, @pre, @post
  - Overkill for your projects

Standard Tags

- @author *text*
- @deprecated *text*
- @exception *class-name description*
- @param *parameter-name description*
- @return *description*
- @see *reference*
- @serial *description* | include | exclude
- @serialData *description*
- @serialField *field-name field-type*
- @since *text*
- @throws (same as @exception)
- @version *text*

Standard Tags, cont

- {@docRoot}
  - Example: See Copyright.
- {@inheritDoc}
  - Copies documentation from the „nearest“ inheritable class
  - Nice when methods are overridden in subclasses
- {@link} *package-name.class#member label*
- {@linkplain} *package-name.class#member label*
  - Same as {@link} but plain text
- {@value}
  - Show the value of a constant

Class Documentation Example

```
/**
 * The SQL keywords. This are defined here to avoid having literals
 * spread all over the code.
 * The ant logo.
 * <center>
 * <img src="{@docRoot}/resources/ant.gif" alt="The Ant Logo">
 * </center>
 *
 * @author Kristian Torp, torp (at) cs (dot) aau (dot) dk
 * @author Viggo Pedersen vig (at) cs (dot) aau (dot) dk
 * @version 1.20
 * @since 1.00
 * <p/>
 * History
 * <pre>
 * Version      How          What changed
 * ----------   ----------   ------------------------------------
 * 1.0          kt           Made first running version
 * 1.1          vig          Made to inherit from DatabaseObject
 * 1.2          kt           Implemented Comparable interface
 * </pre>
 */
```

The Output

The @see Tag

- Current Class
  - @see #field
  - @see #method(Type, Type,...)
  - @see #method(Type argname, Type argname,...)
  - @see #constructor(Type, Type,...)
  - @see #constructor(Type argname, Type argname,...)
- Referencing an element in another package (fully qualified)
  - @see package.Class#field
  - @see package.Class#method(Type, Type,...)
  - @see package.Class#method(Type argname, Type argname,...)
  - @see package.Class#constructor(Type, Type,...)
  - @see package.Class#constructor(Type argname, Type argname,...)
  - @see package.Class.NestedClass
  - @see package.Class
  - @see package

@see, Example

```
/**
   * Checks if two NumberColumns are equal.
   * @param o Object
   * @see java.lang.Object#equals(java.lang.Object)
   */
public boolean equals(Object o) {  // snip }
/**
   * The visitor method.
   * @param v the visitor
   * @link dk.auc.cs.sql.visitor
   */
public void accept(Visitor v) {
        v.visitNumberColumn(this);
}
/**
   * Gets the default value for the column.
   * @return the default value for a column
   * @linkplain Column
   */
public double getDefaultValue() {
      return defaultValue;
}
```

The Output

JavaDoc Guidelines

- Document all public, package, and protected fields and methods.
  - classes, (including inner classes)
  - interface
- Document all constructors
- Document all checked exceptions thrown by methods
- Make a general comment on each package
  - Via package.html file in directory
- Add @auther, @version and @since to each class and interface
  - To document changes
- Add HTML code to documentation where it makes sense
  - Overviews
  - Detailed history
  - Examples

Running the JavaDoc Tool

- **`javadoc`**
  - The Java documentation generator
  - **`javadoc [options] <filename.class> | <package>`**
- Examples
  - **`javadoc dk.auc.sql.ast`**
    - Cannot handle **`assert`** keyword private not covered
    - Must be at right location
  - **`javadoc -private -source 1.4 dk.auc.cs.sql.ast`**
    - Outputs in same directory as source files
  - **`javadoc -private -source 1.4 -d c:\temp dk.auc.cs.sql.ast`**
    - specify the output directory with -d
  - **`javadoc -private -source 1.4 -d c:\temp -windowtitle "Hello JavaDoc" dk.auc.cs.sql.ast`**
    - Make a nice title on main window

Non-JavaDoc Comments

- JavaDoc is for the client programmers and partly for class developers.
- Good implementation comments are still needed for class maintainers
  - For internal use in a company
  - For open source, for any one else to be able to understand the code

Examples of Non-JavaDoc Comments

- @default
  - When a default value is supplied for a system variable
- @design
  - To indicate and communicate important design decision
- @extension
  - Nice-to-have feature that can be implemented in the next release
- @grant
  - Database grant or network grant to be able to use a service
- @limit
  - Know limitation of the system, e.g., max. 10 user can be handled
- @performance
  - Correct but slow implementation, to be looked at in next release
- @todo
  - Should be done before system is release to customers

Rules-of-Thumb

- Saying variable and method names are the best comments
- Make comments where there are surprises
- Make comments where you cannot remember what your own code does
- Make comments clear and correct
- Make comments that focus on *why* rather than *how*!
- Make comments to prepare reader for the code to follow
- Make comments that summarizes what the code does

- Do not comment the obvious

Summary

- Documentation is the piece of information most likely to be missing for a complete software program.
- JavaDoc has set new standards for system documentation
  - Code and documentation stored in the same file
  - Many similar products to other languages
- For the MIP exam you must provide JavaDoc system documentation.

Not documenting your software
is unprofessional!