Documentation of Sotfware

- Types of documentation
- The problems related to documentation
- Purpose JavaDoc
    - Limitation
- Guidelines for JavaDoc from SunSoft
- Summary

Types of Documentation

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

- System documentation
  - Provides all the details
  - Reference manual type of documentation
  - This is what we emphaizise in today's lecture

- Test documentation
  - High-level descrptions of the 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 StarWars]
- Programmers are often relucante 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 build it
- Documenation is outdated
  - Documetation is for version 1.0 of the system and the program is currently in version 5.2
  - Source code in one file, documentation is another file
- Documentation is complete
  - Not all parts of the program is 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. The 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

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- Solve the problem that documentation gets outdated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

JavaDoc Purposes, cont.

- Solve documentation not complete
  - can check e.g., all method documented
- Solve (partly) documentation is incorrect

Limitations of JavaDoc

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

Javadoc, Example

```java
import com.mycompany.misc.*;
/** Implements a garage consisting 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
- Types
  - Block tags, must appear at beginning of line, e.g., @author
  - Inline tags, anywhere must be within curly bracket, e.g., {@link}

Standard Tags

- @author *text*
- @deprecated *text*
- @exception *class-name description*
- @param *paramater-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}**
  - See the Copyright.

- **{@inheritDoc}**
  - Copies documentation from the "nearest" inheritable class

- **{@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) auc (dot) dk
 * @author Viggo Pedersen vig (at) cs (dot) auc (dot) dk
 * @version 1.00
 * @since 1.20
 * <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

```java
/**
 * 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 integer
 * @linkplain Column
 */
public double getDefaultValue() {
        return defaultValue;
}
```

The Output

Running the JavaDoc Tool

- **`javadoc`**
  - The Java documentation generator
  - **`javadoc [options] <filename.class> | <package>`**

- Examples
  - javadoc dk.auc.sql.ast
    - ◆ Cannot handel 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 souce files
  - javadoc -private -source 1.4 -d c:\temp dk.auc.cs.sql.ast
    - ◆ specify the output dir 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

Rules-of-Thumb

- Saying variable and method names are the best comments
- Make comment where there are supprises
- Make comment 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 obious

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!