
Program Documentation

Audience of Program Documentation

- Documentation is targeted at different readers with different information needs:
 1. programmers *working on* the code
 - documentation of code
 - documentation of relationship to design
 2. client programmers *working with* the code
 - documentation of the API (interfaces, contracts)
 - documentation of responsibilities
 3. end-users having the code work (using the final program)
 - manuals
 - online help

Kinds of Program Documentation

- Documentation for code developers
 - Analysis documents
 - for design
 - for tests
 - Design documents; as blueprints for code
 - Source code comments
 - Relationships between documents, e.g.,
 - design decisions: how are requirements reflected in design?
 - decisions made during implementation (patterns, ...)
- Documentation for developers using code
 - API documentation
- Documentation for end-users
 - manuals
 - online help
 - tutorials, guided tours, ...

Source Code Comments

- Comments explain the source code.
 - semantics is not obvious from code alone
 - remember: pre-, post-, and side-conditions
 - pragmatics: why at all?
- Java has two kinds of comments for programmers working *on* code
 - single-line comments

```
// this is a single line comment
```
 - multi-line comments

```
/*  
    this is a block comment  
*/
```

Useful Source Code Comments (1)

- ❑ Make sure that comments are helpful!
- ❑ Bad example:

```
boolean check (int [] a) {  
    // iterate counting in i  
    for (int i=1 ; i<a.length ; i++)  
        // compare a [i-1] and a [i]  
        if (a [i-1] > a [i])  
            return false ;  
    // return true  
    return true ;  
}
```

Useful Source Code Comments (2)

- Good example:

```
// check whether array is sorted in ascending order  
boolean check (int [] a) {  
    // iterate 2nd...last element  
    for (int i = 1 ; i < a.length ; i++)  
        // compare to predecessor; if order is violated, fail  
        if (a [i-1] > a [i])  
            return false ;  
    // no violation found => report order ok  
    return true ;  
} // check
```

- Note: identifier names also contribute!

```
boolean isInAscendingOrder  
    (int [] arrayToCheck) { ... }
```

Documentation Generated from Code

- API documentation can be generated from comments included in source code for programmers *working with* code.
 - in Java: Javadoc
 - special comment:
`/** this goes into the API documentation */`
 - can be attached to classes, fields, methods, and constructors
 - the Javadoc tool generates documentation from given source files
- Usage
 - from command line, e.g.:
`javadoc -d output-directory -use -version -author
-windowtitle "title" -doctitle "title"
-sourcepath path -classpath path list-of-packages`
 - interactively from IDEs

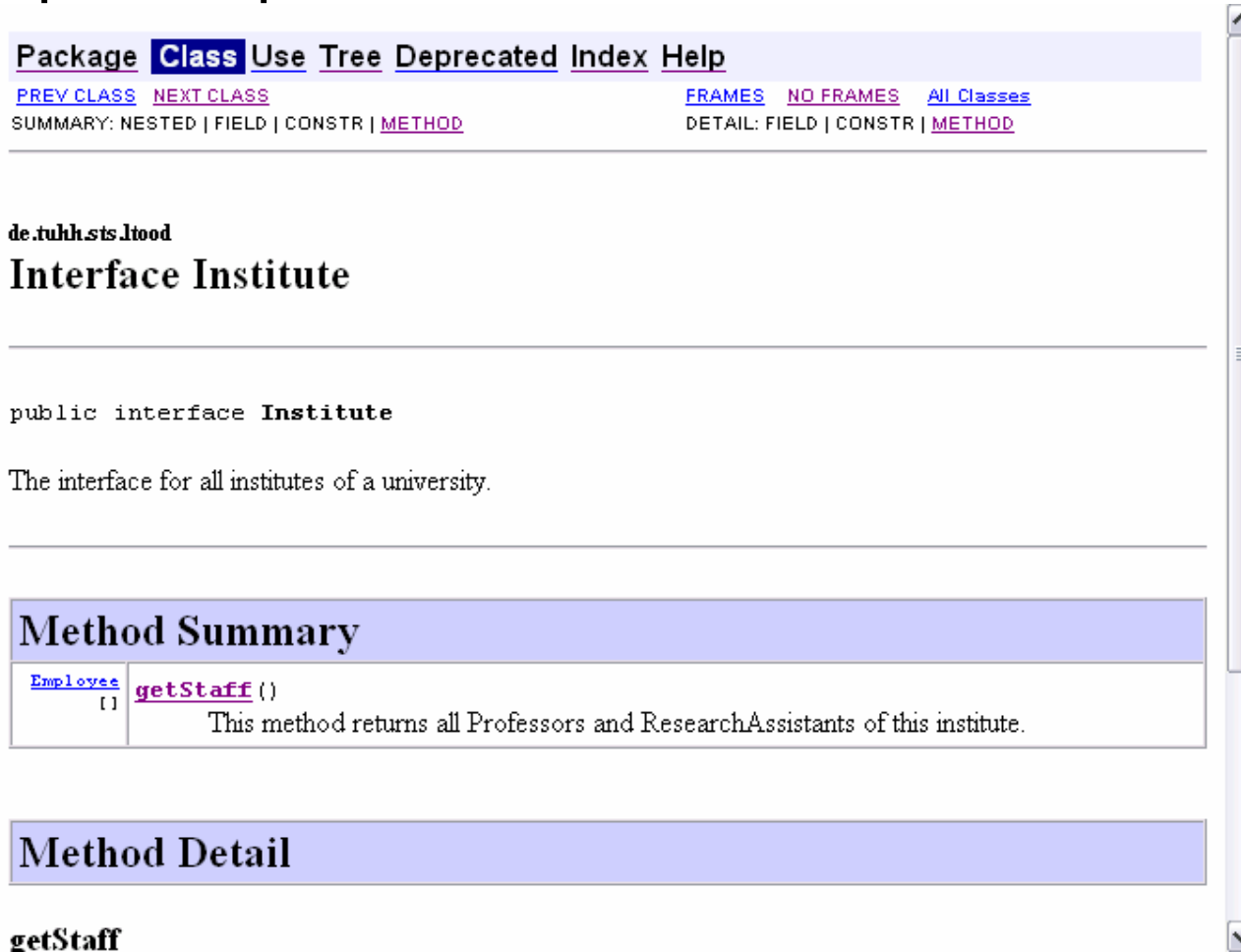
JavaDoc (1)

□ Example input:

```
package de.sts.tuhh.ltood ;  
/**  
 * The interface for all institutes of a university.  
 */  
public interface Institute {  
 /**  
  * This method returns all  
  * Professors and  
  * ResearchAssistants of  
  * this institute.  
  */  
  public Employee [] getStaff () ;  
} // interface Institute
```

JavaDoc (2)

□ Example output:



The screenshot shows a JavaDoc page for the `de.tuhh.sts.ltood` package. The page title is **Interface Institute**. The code defines a public interface `Institute` with a description: "The interface for all institutes of a university." The **Method Summary** section lists a method `getStaff()` that returns an array of `Employee` objects. The description for this method is "This method returns all Professors and ResearchAssistants of this institute." The **Method Detail** section is partially visible, showing the signature `getStaff`.

Package **Class** Use Tree Deprecated Index Help

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#) DETAIL: FIELD | CONSTR | [METHOD](#)

`de.tuhh.sts.ltood`

Interface Institute

`public interface Institute`

The interface for all institutes of a university.

Method Summary

Employee []	getStaff () This method returns all Professors and ResearchAssistants of this institute.
--------------------------------	---

Method Detail

`getStaff`

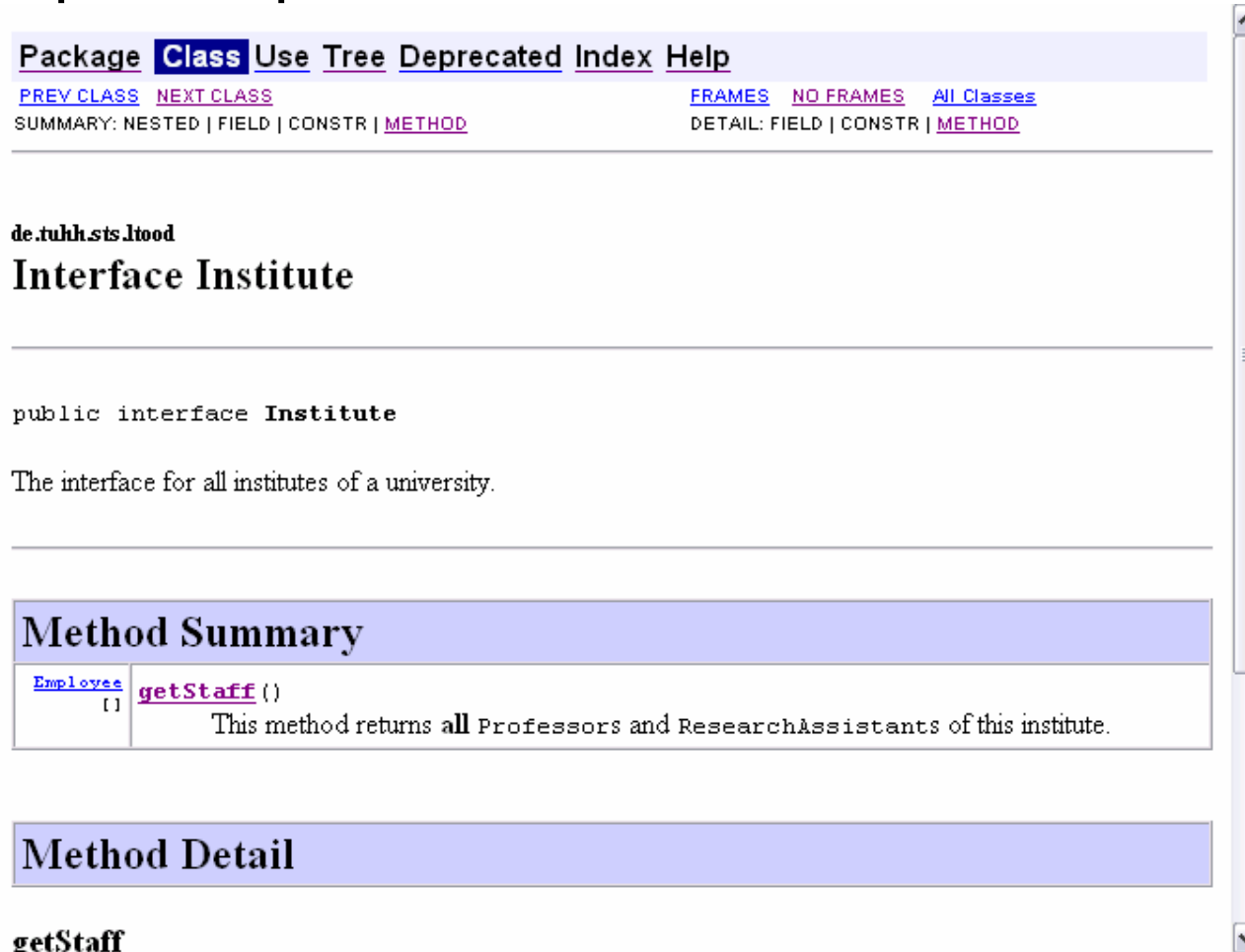
JavaDoc (3)

- By default, HTML pages are generated.
- Therefore, HTML markup can be included in comments:

```
public interface Institute {  
    /**  
     * This method returns <b>all</b>  
     * <code>Professor</code>s and  
     * <code>ResearchAssistant</code>s of  
     * this institute.  
     */  
    public Employee [] getStaff () ;  
} // interface Institute
```

JavaDoc (4)

□ Example output:



The screenshot shows a JavaDoc page for the `de.tuhh.sts.ltood` package. The page title is `Interface Institute`. The code snippet shows a public interface `Institute` with a description: "The interface for all institutes of a university." Below this, there is a "Method Summary" section with a table listing the `getStaff()` method. The method signature is `Employee[] getStaff()` and its description is "This method returns all Professors and ResearchAssistants of this institute." Below the method summary is a "Method Detail" section, which is currently empty. The page includes navigation links for package, class, use tree, deprecated, index, help, previous class, next class, frames, no frames, all classes, and summary/detail options.

Package [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [All Classes](#)
SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#) DETAIL: FIELD | CONSTR | [METHOD](#)

`de.tuhh.sts.ltood`
Interface Institute

`public interface Institute`

The interface for all institutes of a university.

Method Summary

Employee []	getStaff () This method returns all Professors and ResearchAssistants of this institute.
--------------------------------	--

Method Detail

getStaff

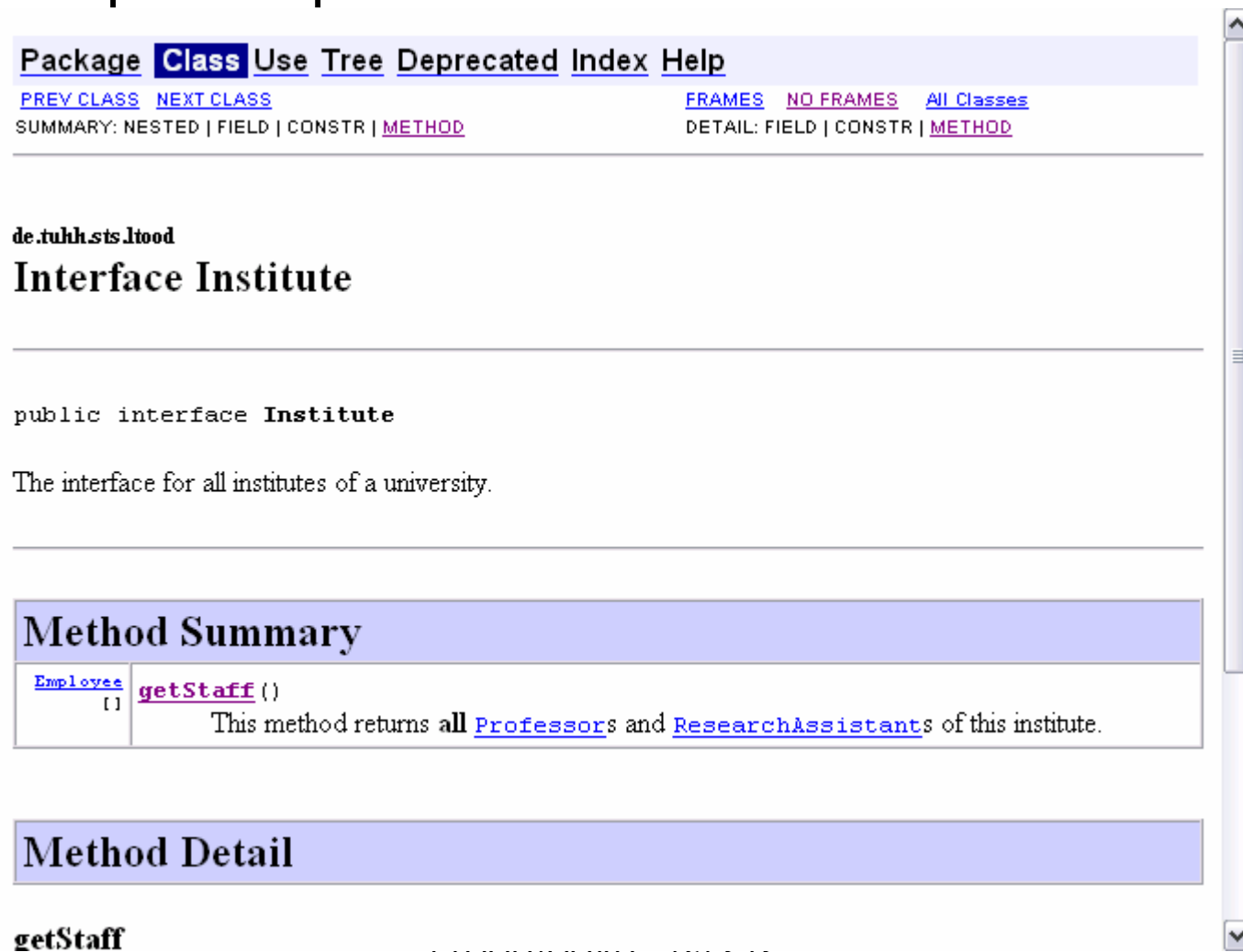
JavaDoc (5)

- There are special JavaDoc tags:
 - standalone tags
 - inline tags
- Example of an inline tag: `{@link}`

```
public interface Institute {  
    /**  
     * This method returns <b>all</b>  
     * {@link de.tuhh.sts.ltood.Professor}s and  
     * {@link de.tuhh.sts.ltood.ResearchAssistant}s of  
     * this institute.  
     */  
    public Employee [] getStaff () ;  
} // interface Institute
```

JavaDoc (6)

□ Example output:



The screenshot shows a JavaDoc page for the `de.tuhh.sts.llood` package. The page title is `Interface Institute`. The page content includes a navigation bar with links for `Package`, `Class`, `Use`, `Tree`, `Deprecated`, `Index`, and `Help`. Below the navigation bar, there are links for `PREV CLASS`, `NEXT CLASS`, `FRAMES`, `NO FRAMES`, `All Classes`, `SUMMARY: NESTED | FIELD | CONSTR | METHOD`, and `DETAIL: FIELD | CONSTR | METHOD`. The main content area shows the package name `de.tuhh.sts.llood` and the interface name `Interface Institute`. Below this, the code `public interface Institute` is displayed. A description follows: "The interface for all institutes of a university." The `Method Summary` section shows a table with one entry: `Employee` (with a small icon) and `getStaff ()`. The description for `getStaff ()` is "This method returns all `Professors` and `ResearchAssistants` of this institute." The `Method Detail` section is also visible. The footer of the page contains the text `getStaff` and `LLOOD/OUAD - (C) SIS`.

Package **Class** Use Tree Deprecated Index Help

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#) DETAIL: FIELD | CONSTR | [METHOD](#)

de.tuhh.sts.llood

Interface Institute

```
public interface Institute
```

The interface for all institutes of a university.

Method Summary

Employee [1]	getStaff () This method returns all Professors and ResearchAssistants of this institute.
---------------------------------	---

Method Detail

getStaff

LLOOD/OUAD - (C) SIS

JavaDoc (7)

- Examples of standalone tags: @author, @version, @see

```
package de.sts.tuhh.ltood ;  
/**  
 * The interface for all institutes of a university.  
 * @author Hans-Werner Sehring  
 * @version 1.0  
 */  
public interface Institute {  
 /**  
  * This method returns <b>all</b>  
  * <code>Professor</code>s and  
  * <code>ResearchAssistant</code>s of  
  * this institute.  
  * @see de.tuhh.sts.ltood.Professor  
  * @see de.tuhh.sts.ltood.ResearchAssistant  
  */  
  public Employee [] getStaff () ;  
 } // interface Institute
```

JavaDoc (8)

□ Example output:

```
public interface Institute
```

The interface for all institutes of a university.

Version:

1.0

Author:

Hans-Werner Sehring

Method Summary

[Employee](#)

[1]

[getStaff\(\)](#)

This method returns **all** Professors and ResearchAssistants of this institute.

Method Detail

getStaff

[Employee](#)[] **getStaff()**

This method returns **all** Professors and ResearchAssistants of this institute.

See Also:

[Professor](#), [ResearchAssistant](#)

JavaDoc (9)

- Examples of standalone tags for methods:

```
@param, @return, @throws
package de.sts.tuhh.ltood ;
public interface Institute {
    public Employee [] getStaff () ;
    /**
     * Create a new <code>Professor</code> in
     * this institute. The newly created professor is
     * initialized with the given name.
     * @param name the name of the new professor
     * @return the newly created professor
     * @throws IllegalArgumentException when <code>name</code>
     * is not valid
     */
    public Professor createProfessor (String name)
        throws IllegalArgumentException ;
} // interface Institute
```

JavaDoc (10)

□ Example output:

This method returns **all** Professors and ResearchAssistants of this institute.

See Also:

[Professor](#), [ResearchAssistant](#)

createProfessor

[Professor](#) **createProfessor**(java.lang.String name)
throws [IllegalNameException](#)

Create a new Professor in this institute. The newly created professor is initialized with the given name.

Parameters:

name - the name of the new professor

Returns:

the newly created professor

Throws:

[IllegalNameException](#) - when name is not valid

Package **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

JavaDoc Extensions

- Often the need arises to extend the set of available tags.
- E.g., a “to do” tag to note parts of code which need rework
 - tag `@todo` in JBuilder
 - special comment with `TODO:` in Eclipse
- Often, such tags are evaluated by the special environment which introduced them (JBuilder, Eclipse, ...).
- Still, JavaDoc is open to such additions.

JavaDoc Components

- The JavaDoc tool is based on two kinds of components:
 - doclets
 - taglets
- For both individual implementations can be given.
- References:
 - JavaDoc documentation by Sun [1]
 - Lisa Friendly: **The Design of Distributed Hyperlinked Programming Documentation.** [2]

Standard Doclet

- In the JDK a standard doclet with standard taglets is provided
 - DocRootTaglet for `@docRoot`
 - InheritDocTaglet for `@inheritDoc`
 - ParamTaglet for `@param`
 - ReturnTaglet for `@return`
 - SeeTaglet for `@see`
 - SimpleTaglet for standalone tags like `@author` and `@version`
 - ThrowsTaglet for `@throws`
 - ValueTaglet for `@value`

Custom Doclets

- ❑ Custom doclets can be implemented using the doclet API.
- ❑ Simple example: print out all class names

```
import com.sun.javadoc.*;

public class ListClass {
    static public boolean start (RootDoc root) {
        ClassDoc [] classes = root.classes();
        for (int i=0; i < classes.length; i++)
            System.out.println(classes[i]);
        return true;
    }
} // class ListClass
```

Custom Tags

- Easy introduction of tag using the `tags()` method defined for classes, methods, ...
- Example: `@todo` tag of a doclet printing out open tasks

```
import com.sun.javadoc.* ;
public class ListToDo {
    static public boolean start (RootDoc root) {
        ClassDoc [] classes = root.classes () ;
        for (int i=0 ; i<classes.length ; i++) {
            Tag [] tags = classes [i].tags ("todo") ;
            for (int j=0 ; j<tags.length ; j++)
                System.out.println ("todo for " +
                                     classes [i] + ": " +
                                     tags [j].text ()) ;

            } // for i
        return true ;
    }
} // class ListToDo
```

Custom Taglets (1)

- Writing complete doclets is a tedious task. Therefore, one prefers to
 - subclass an existing doclet
 - introduce custom taglets
- The taglet API consists of one interface:
`com.sun.tools.doclets.Taglet`
- Example: “to do” tag
 - usage in JavaDoc comments:
`@todo complete method body`
 - output in documentation page:
To do: *complete method body*

Custom Taglets (2)

- A taglet has to implement methods
 - for registration,
 - to describe its name and applicability, and
 - to print out information given by tags.
- Sample code:

```
import com.sun.tools.doclets.Taglet ;
import com.sun.javadoc.* ;
import java.util.Map ;
public class ToDoTaglet implements Taglet {
    public static void register (Map tagletMap) {
        ToDoTaglet tag = new ToDoTaglet () ;
        Taglet t=(Taglet)tagletMap.get (tag.getName ()) ;
        if (t != null)
            tagletMap.remove (tag.getName ()) ;
        tagletMap.put (tag.getName (), tag) ;
    } // register
    public String getName () { return "todo" ; }
}
```

(continued on next slide)

Custom Taglets (3)

(continued from previous slide)

```
// tag works for fields
public boolean inField () { return true ; }
// tag works for constructors
public boolean inConstructor () { return true ; }
// tag works for methods
public boolean inMethod () { return true; }
// tag works for overview
public boolean inOverview () { return true ; }
// tag works for packages
public boolean inPackage () { return true ; }
// tag works for classes and interfaces
public boolean inType () { return true ; }
// tag is of type "standalone" (not an inline tag)
public boolean isInlineTag () { return false ; }
```

(continued on next slide)

Custom Taglets (4)

(continued from previous slide)

```
public String toString (Tag tag) {
    return "<p><b>To do:</b> <i>" + tag.text() + "</i></p>\n";
} // toString

public String toString (Tag [] tags) {
    if (tags.length == 0)
        return null ;
    String result = "\n<p><b>To do:</b> <i>" ;
    for (int i = 0 ; i < tags.length ; i++)
        result += (i == 0 ? "" : ", ") + tags [i].text();
    return result + "</i></p>\n";
} // toString
} // class ToDoTaglet
```

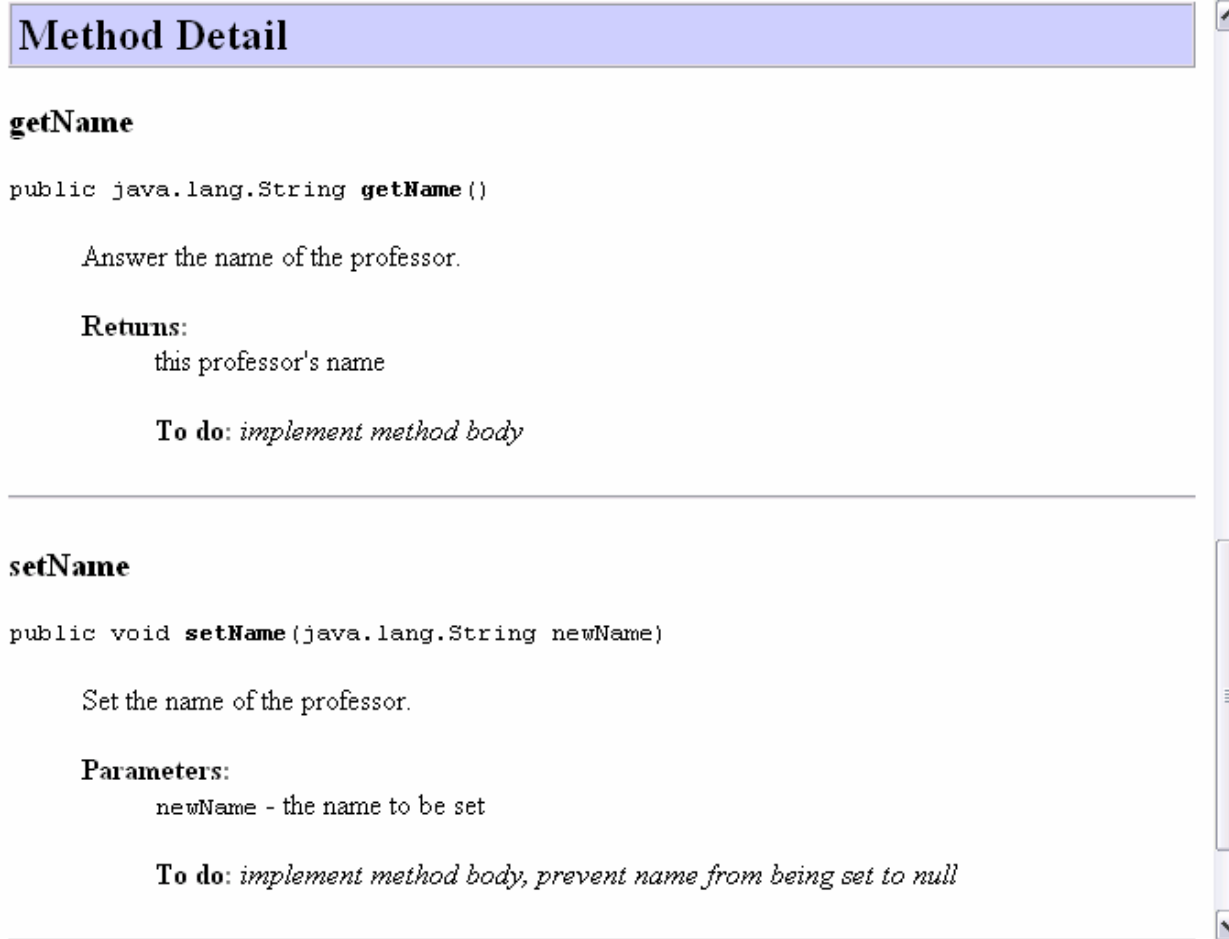
Custom Taglet Example (1)

- Sample input:

```
public class Professor {  
    /**  
     * The name of a professor.  
     * @todo make field private  
     */  
    String name ;  
    /**  
     * Answer the name of the professor.  
     * @return this professor's name  
     * @todo implement method body  
     */  
    public String getName () { return null ; }  
    /**  
     * Set the name of the professor.  
     * @param the name to be set  
     * @todo implement method body  
     * @todo prevent name from being set to null  
     */  
    public void setName (String newName) { ... }  
} // class Professor
```

Custom Taglet Example (2)

□ Sample output:



The screenshot shows a scrollable window titled "Method Detail" with a light blue header. It contains two method entries. The first entry is for the `getName` method, which is a public `java.lang.String` method. It includes a description "Answer the name of the professor.", a "Returns:" section stating "this professor's name", and a "To do:" note "implement method body". The second entry is for the `setName` method, which is a public void method taking a `java.lang.String` parameter named `newName`. It includes a description "Set the name of the professor.", a "Parameters:" section stating "newName - the name to be set", and a "To do:" note "implement method body, prevent name from being set to null". A vertical scrollbar is on the right side of the window.

Method Detail

getName

```
public java.lang.String getName ()
```

Answer the name of the professor.

Returns:
this professor's name

To do: *implement method body*

setName

```
public void setName (java.lang.String newName)
```

Set the name of the professor.

Parameters:
newName - the name to be set

To do: *implement method body, prevent name from being set to null*

References

- [1] JavaDoc documentation by Sun:
<http://java.sun.com/j2se/javadoc/>
- [2] Lisa Friendly: *The Design of Distributed Hyperlinked Programming Documentation*. In: Proceedings of the International Workshop on Hypermedia Design (IWHD'95), Montpellier, France, 1-2 June 1995