
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

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

Source Code Comments

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

Useful Source Code Comments (1)

- o Make sure that comments are helpful!
- o 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

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

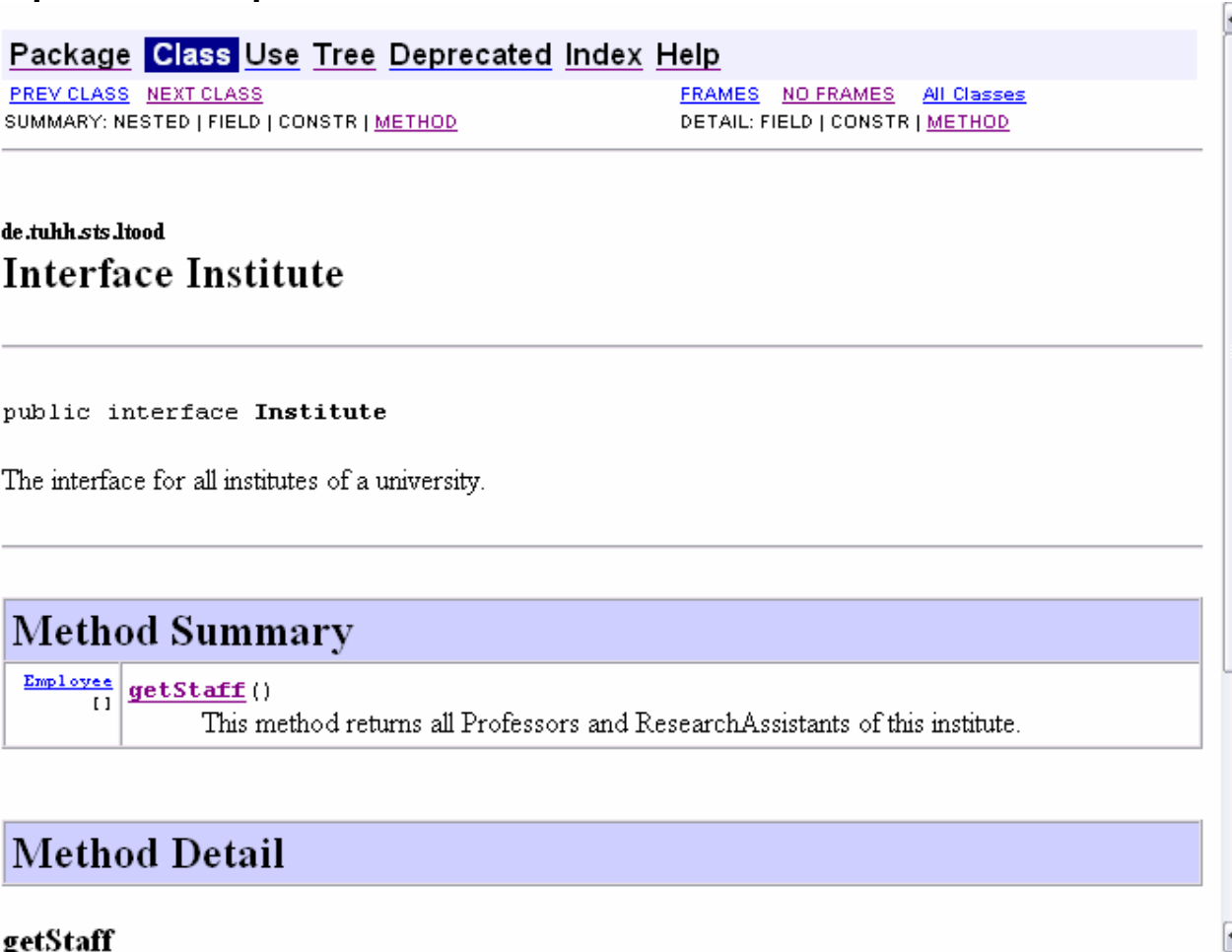
JavaDoc (1)

- 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)

- o Example output:



The screenshot shows a JavaDoc page for the `de.tuhh.sts.ltood` package. The page title is `Interface Institute`. The package name is `de.tuhh.sts.ltood`. The interface is defined as `public interface Institute`. The description is "The interface for all institutes of a university." The `Method Summary` section shows a table with one entry: `getStaff ()` with a return type of `Employee []`. The description for `getStaff ()` is "This method returns all Professors and ResearchAssistants of this institute." The `Method Detail` section shows 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`

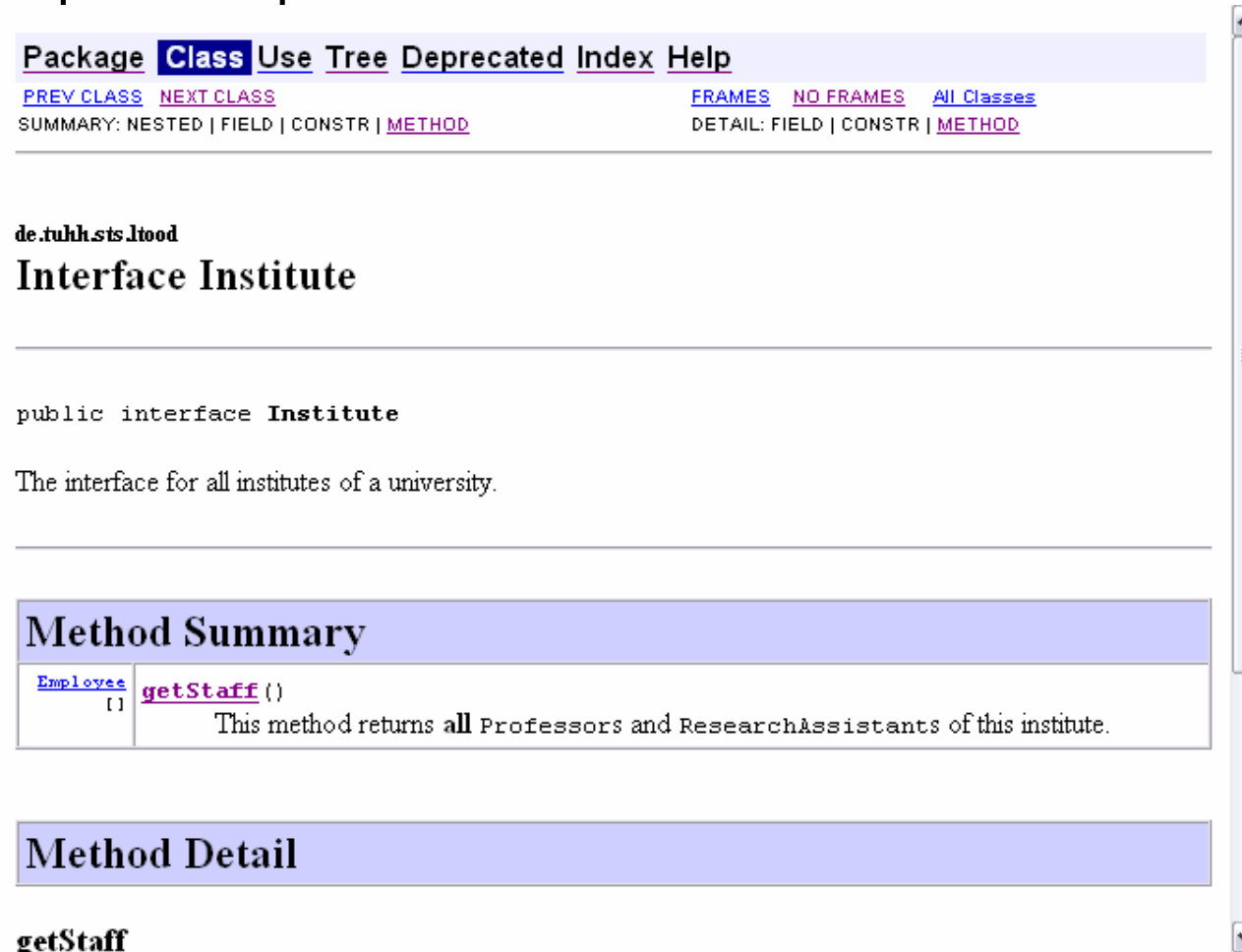
JavaDoc (3)

- 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)

- o 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 `public interface Institute`. The description states: "The interface for all institutes of a university." The **Method Summary** section contains a table with one entry: `Employee [] getStaff ()` with the description "This method returns all Professors and ResearchAssistants of this institute." The **Method Detail** section shows 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

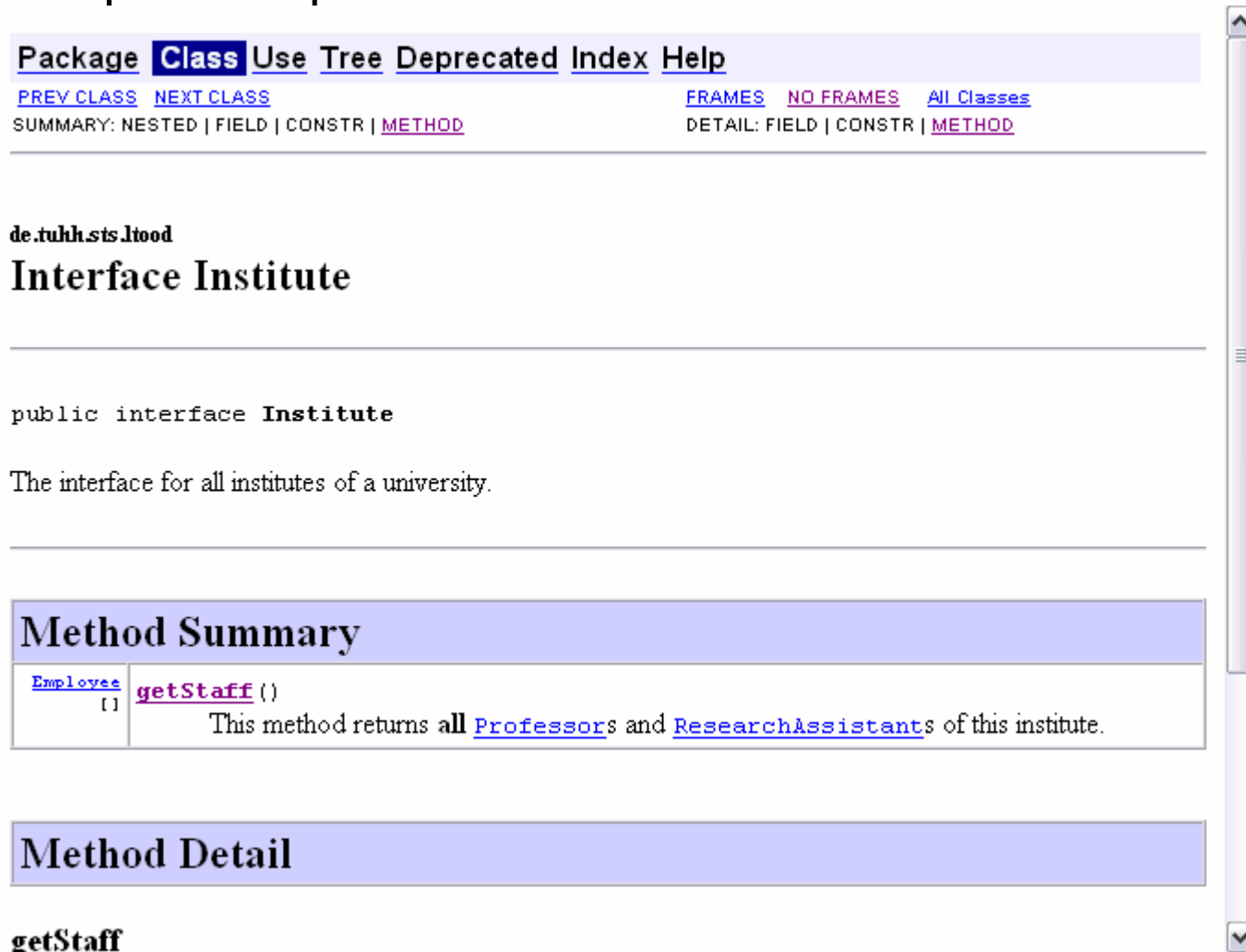
JavaDoc (5)

- o There are special JavaDoc tags:
 - n standalone tags
 - n inline tags
- o 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)

- o Example output:



The screenshot shows a JavaDoc page for the `de.tuhh.sts.llood` 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 `getStaff()` states: "This method returns all `Professors` and `ResearchAssistants` of this institute." The `Method Detail` section is also visible.

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

Method Detail

getStaff

JavaDoc (7)

- o 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)

- o Example output:

```
public interface Institute
```

The interface for all institutes of a university.

Version:

1.0

Author:

Hans-Werner Sehring

Method Summary

Employee []	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)

- o 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)

- o 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

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

JavaDoc Components

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

Standard Doclet

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

Custom Doclets

- o Custom doclets can be implemented using the doclet API.
- o 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)

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

Custom Taglets (2)

- o A taglet has to implement methods
 - n for registration,
 - n to describe its name and applicability, and
 - n to print out information given by tags.
- o 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
```

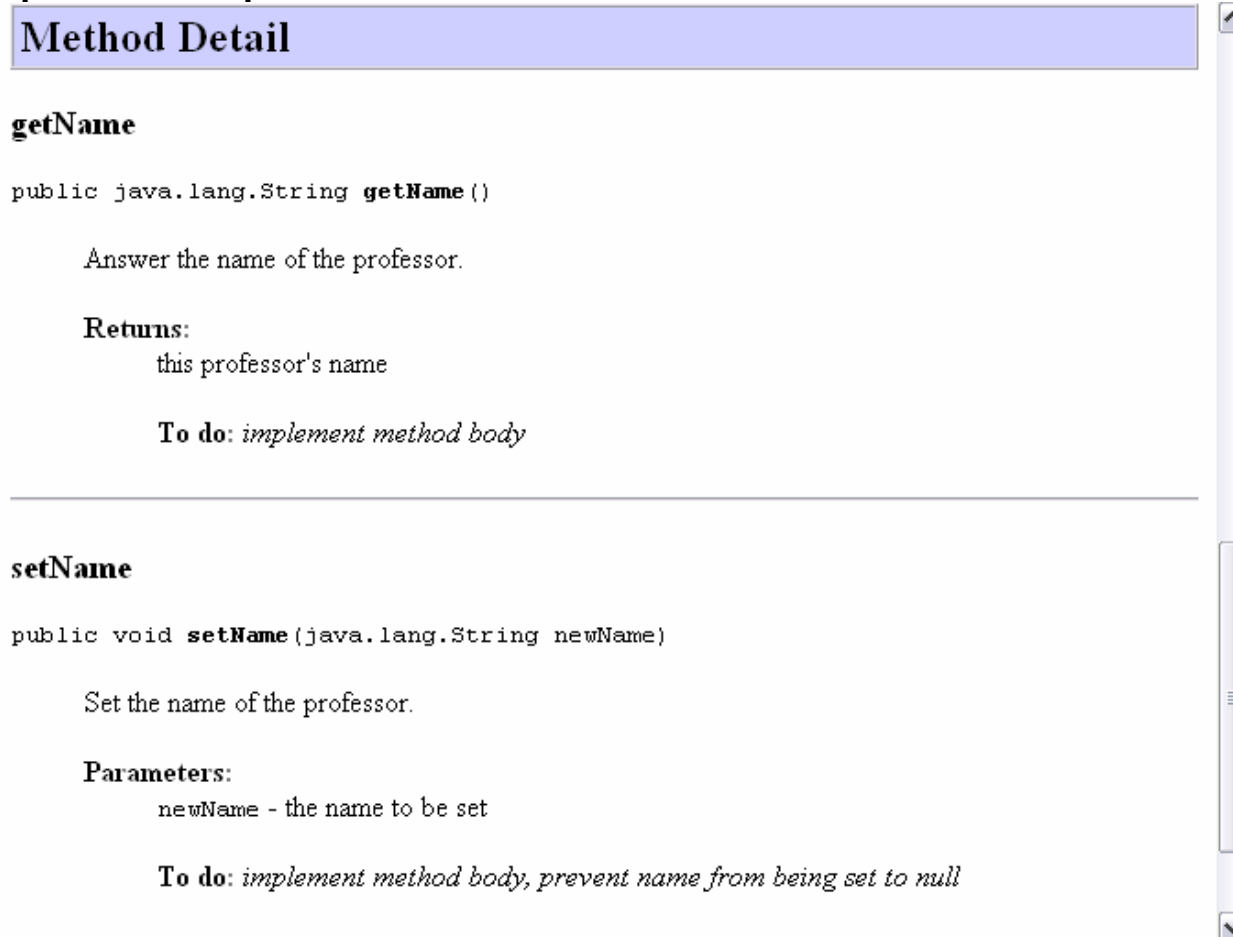
Custom Taglet Example (1)

- o 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)

- o Sample output:



The screenshot shows a Java IDE window titled "Method Detail". It displays the details for two methods: `getName` and `setName`. The `getName` method is a public `String` method that returns the name of the professor. The `setName` method is a public `void` method that sets the name of the professor, with a note to prevent the name from being set to null.

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