Using `javadoc` Tool

Using the `javadoc` Tool

Documentation of your code is important to the success of future maintenance efforts for standalone applications and it critical to the use of APIs

- **`javadoc`** tool
  - Extract information from
    - packages, interfaces, classes and fields
- Documentation comment tags
- How to use the tool

Usage

- Java 2 SDK tool generates HTML documentation pages
- Usage:

```
javadoc [optionss] [packages|files]
```

Option	Value	Decription
-d	Output path	The directory in which the generated HTML files should placed
-sourcepath	Directory path	The root directory where the source file package tree
-public		Specify that only public declarations be included (default)
-private		Specify that all declareations be included

Common Documentation Tags

- Comments starting with /** and ending with */ are parsed by the `javadoc` tool; *free-form text (HTML modifiers)* followed by *tags*
- These comments should immediately precede the declaration they reflect

Tag	Purpose	Class/interface	Constructor	Method	Attribute
@see	To create a link to another declaration	✓	✓	✓	✓
@author	The author of the class or interface	✓			
@param	Documents a parameter		✓	✓	
@return	Documents the return		✓	✓	

Example

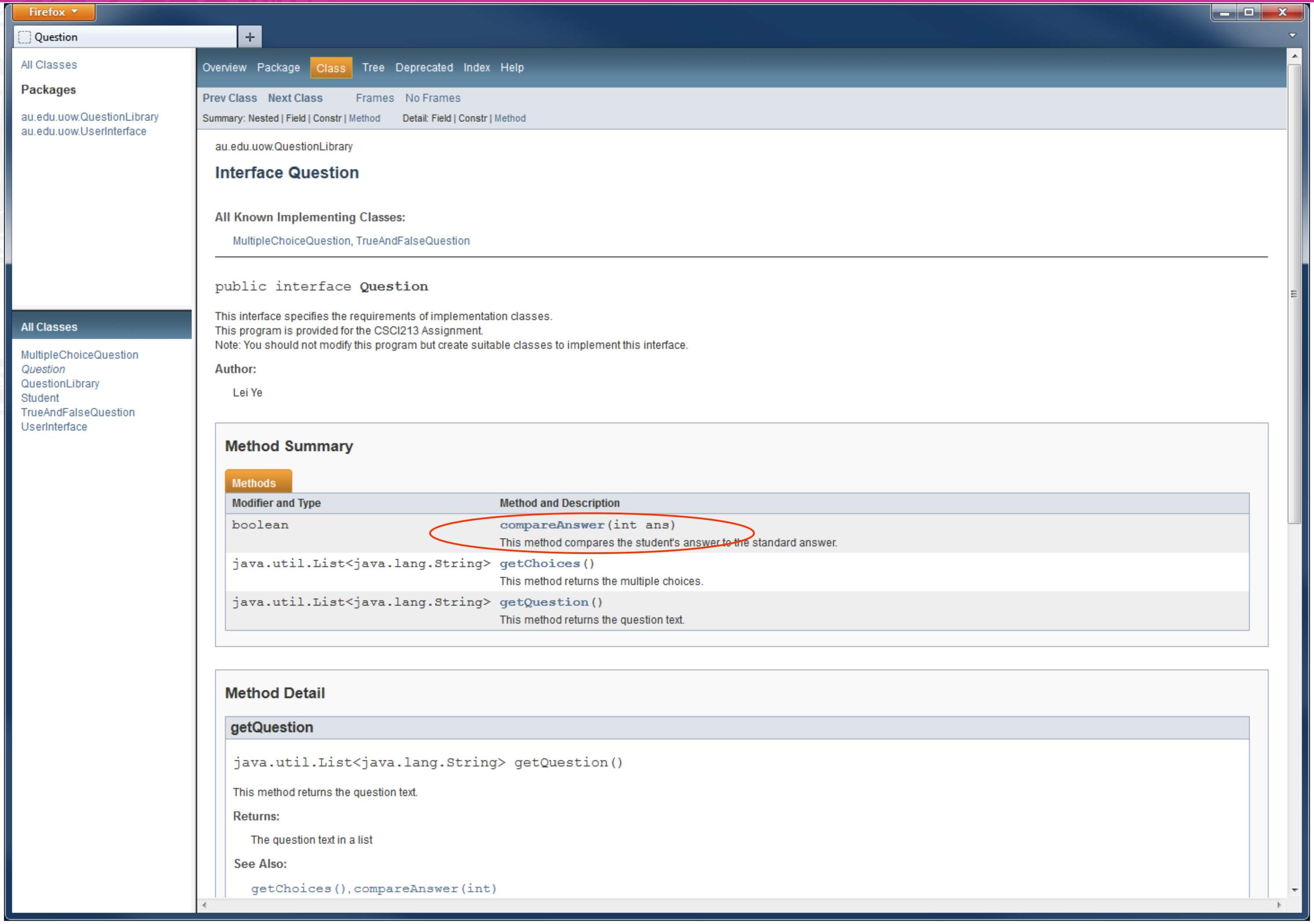
```
/**
 * This interface specifies the requirements of implementation classes.<br>
 *
 * This program is provided for the CSCI213 Assignment. <br>
 * Note: You should not modify this program but create suitable
 *       classes to implement this interface.
 *
 * @author Lei Ye
 *
 */
public interface Question {
       /**
        * This method returns the question text.
        * @return The question text in a list
        * @see #getChoices()
        * @see #compareAnswer(int)
        */
       List<String> getQuestion();

       /**
        * This method returns the multiple choices.
        * @return The list of choices
        * @see #getQuestion()
        * @see #compareAnswer(int)
        */
       List<String> getChoices();

       /**
        * This method compares the student's answer to the standard answer.
        * @see #getQuestion()
        * @see #getChoices()
        * @param ans The student's answer
        * @return True for the correct answer; false for incorrect answers.
        */
       boolean compareAnswer(int ans);
}
```

```
>javadoc -author -d doc Question.java
```

Example (Cont.)


Firefox
Question
All Classes
Packages
au.edu.uow.QuestionLibrary
au.edu.uow.UserInterface
All Classes
MultipleChoiceQuestion
Question
QuestionLibrary
Student
TrueAndFalseQuestion
UserInterface
Overview Package Class Tree Deprecated Index Help
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

au.edu.uow.QuestionLibrary

Interface Question

All Known Implementing Classes:

MultipleChoiceQuestion, TrueAndFalseQuestion

```
public interface Question
```

This interface specifies the requirements of implementation classes.
This program is provided for the CSCI213 Assignment.
Note: You should not modify this program but create suitable classes to implement this interface.

Author:

Lei Ye

Method Summary

Methods

Modifier and Type	Method and Description
`boolean`	`compareAnswer(int ans)` This method compares the student's answer to the standard answer.
`java.util.List<java.lang.String>`	`getChoices()` This method returns the multiple choices.
`java.util.List<java.lang.String>`	`getQuestion()` This method returns the question text.

Method Detail

getQuestion

```
java.util.List<java.lang.String> getQuestion()
```

This method returns the question text.

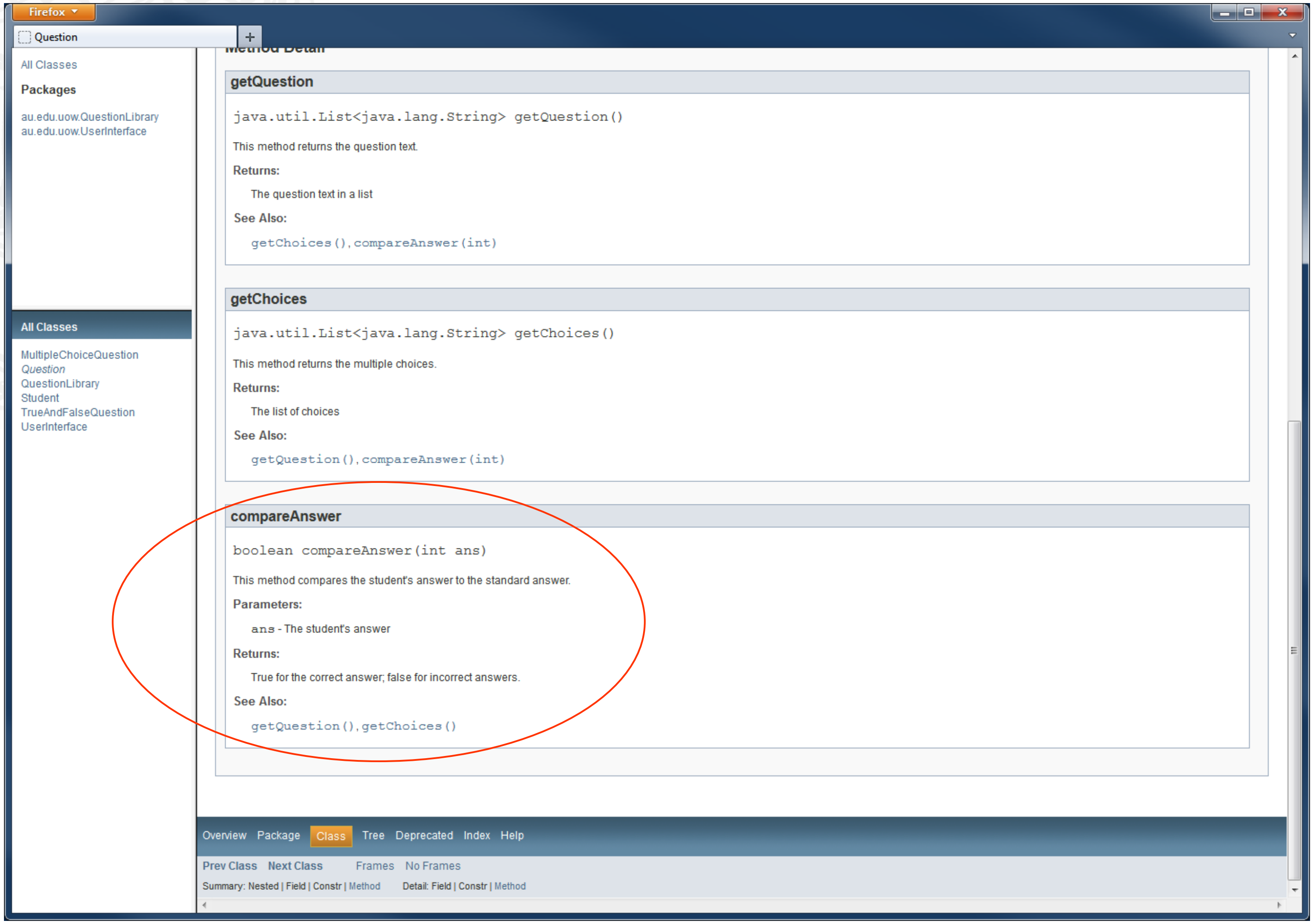
Returns:

The question text in a list

See Also:

`getChoices()`, `compareAnswer(int)`

Example (Cont.)



Firefox

Question

All Classes

Packages

au.edu.uow.QuestionLibrary
au.edu.uow.UserInterface

All Classes

MultipleChoiceQuestion
Question
QuestionLibrary
Student
TrueAndFalseQuestion
UserInterface

getQuestion

```
java.util.List<java.lang.String> getQuestion()
```

This method returns the question text.

Returns:

The question text in a list

See Also:

`getChoices()`, `compareAnswer(int)`

getChoices

```
java.util.List<java.lang.String> getChoices()
```

This method returns the multiple choices.

Returns:

The list of choices

See Also:

`getQuestion()`, `compareAnswer(int)`

compareAnswer

```
boolean compareAnswer(int ans)
```

This method compares the student's answer to the standard answer.

Parameters:

`ans` - The student's answer

Returns:

True for the correct answer; false for incorrect answers.

See Also:

`getQuestion()`, `getChoices()`

Overview Package Class Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

UNIVERSITY OF WOLLONGONG

Resources on Documentation

- How to write doc comments for **`javadoc`**

  http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Java Code Conventions

From Programmer to Developer

Reasons for Code Conventions

- **80%** of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is **maintain**ed for its whole life by the original author.
- Code conventions improve the **readability** of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Code Conventions

- File Organization

```
Class/interface documentation comment
    (/** … */)
Class/interface statement
Class/interface implementation comment
    (/* … */)
Class (static) variables
Instance variables
Constructors
Methods
```

Code Conventions

- Indentation
  - Line length
    - < 80 char
  - Wrapping lines
    - Break after a comma.
    - Break before an operator.
    - Prefer higher-level breaks to lower-level breaks.
    - Align the new line with the beginning of the expression at the same level on the previous line.
    - If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Code Conventions

- Implementation Comments /* … */ -- about the particular implementation
  - Block comments

```
/*
*  Here is a block comment.
*/
```

  - Single-line comments

```
if (condition) {
    /* Handle the condition. */
    ...
}
```

  - Trailing Comments

```
if (a == 2) {
    return TRUE;            /* special case */
} else {
    return isPrime(a);      /* works only for odd a */
}
```

- Documentation Comments /** … */ -- the specification of the code (omitted)

Code Conventions

- Declaration
  - Number per line

    ```
    int level, size;
    ```

  - Initialization
    - Initialize local variables where they are declared
  - Placement
    - Put declarations only at the beginning of blocks
  - Class and Interface Declarations
    - No space between a method name and the parenthesis "(" starting its parameter list
    - Open brace "{" appears at the end of the same line as the declaration statement
    - Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"

Code Conventions

- Naming Conventions

Identifier Type	Rules for Naming
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names
Classes/interfaces	Class names should be ***nouns***, in mixed case with the first letter of each internal word capitalized
Methods	Methods should be ***verbs***, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized
Variables	Variables are in mixed case with the first letter lowercase, with the first letter of each internal word capitalized
Constants	Names of constants should be all uppercase with words separated by underscores ("_")

Resources on Code Conventions

- Code Conventions for the Java Programming Language

  `http://www.oracle.com/technetwork/java/codeconv-138413.html`