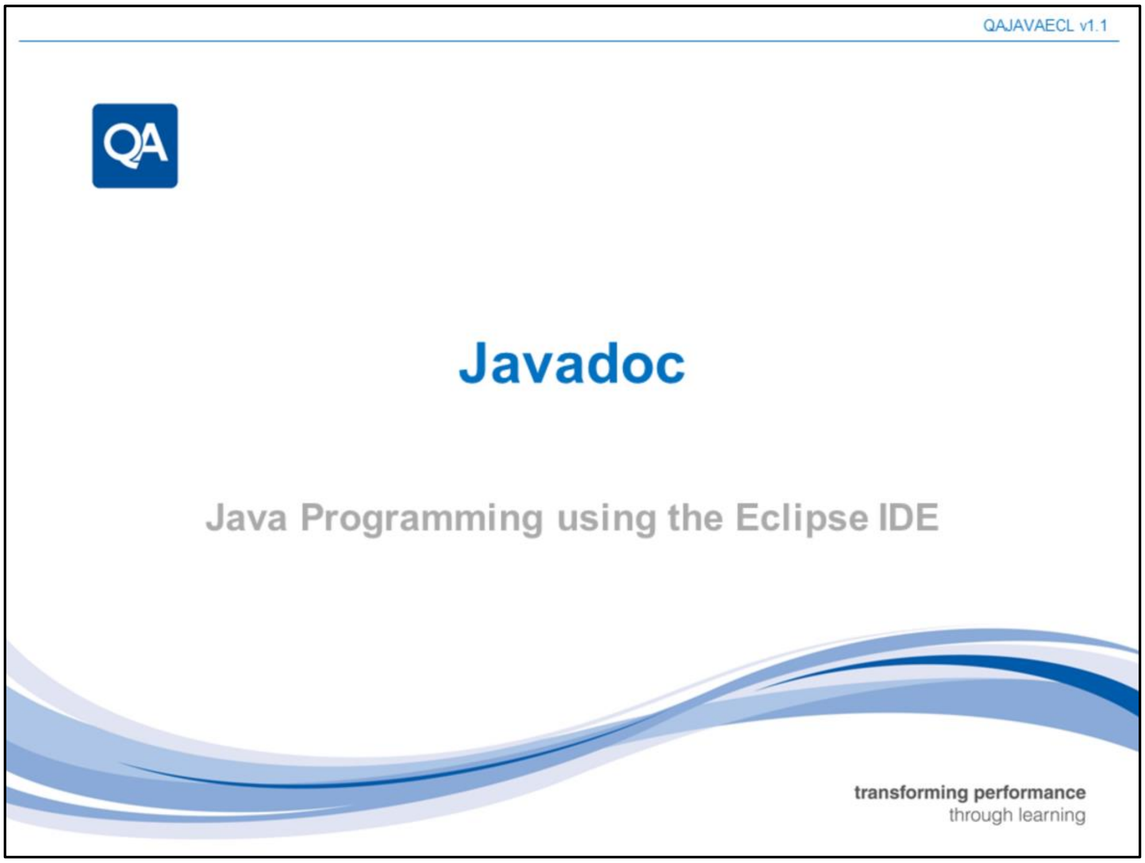
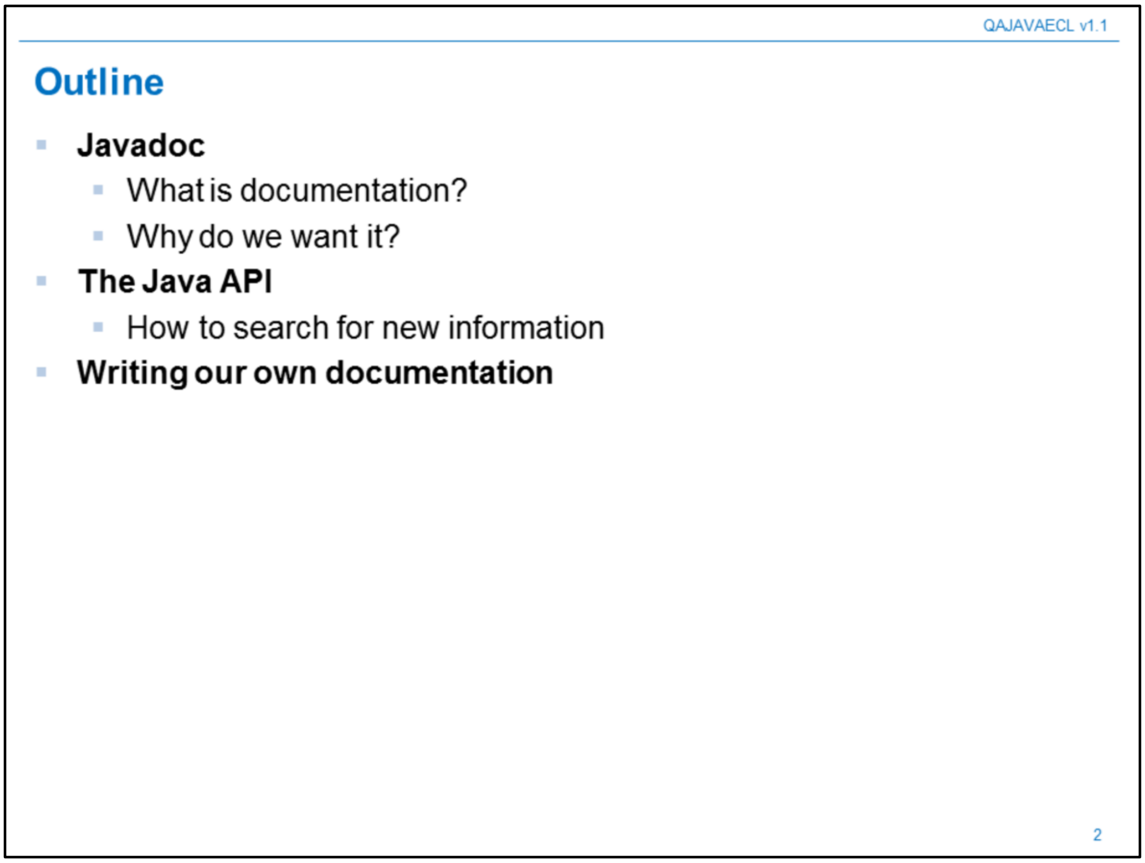


QAJAVAECL v1.1

Javadoc

Java Programming using the Eclipse IDE

transforming performance
through learning

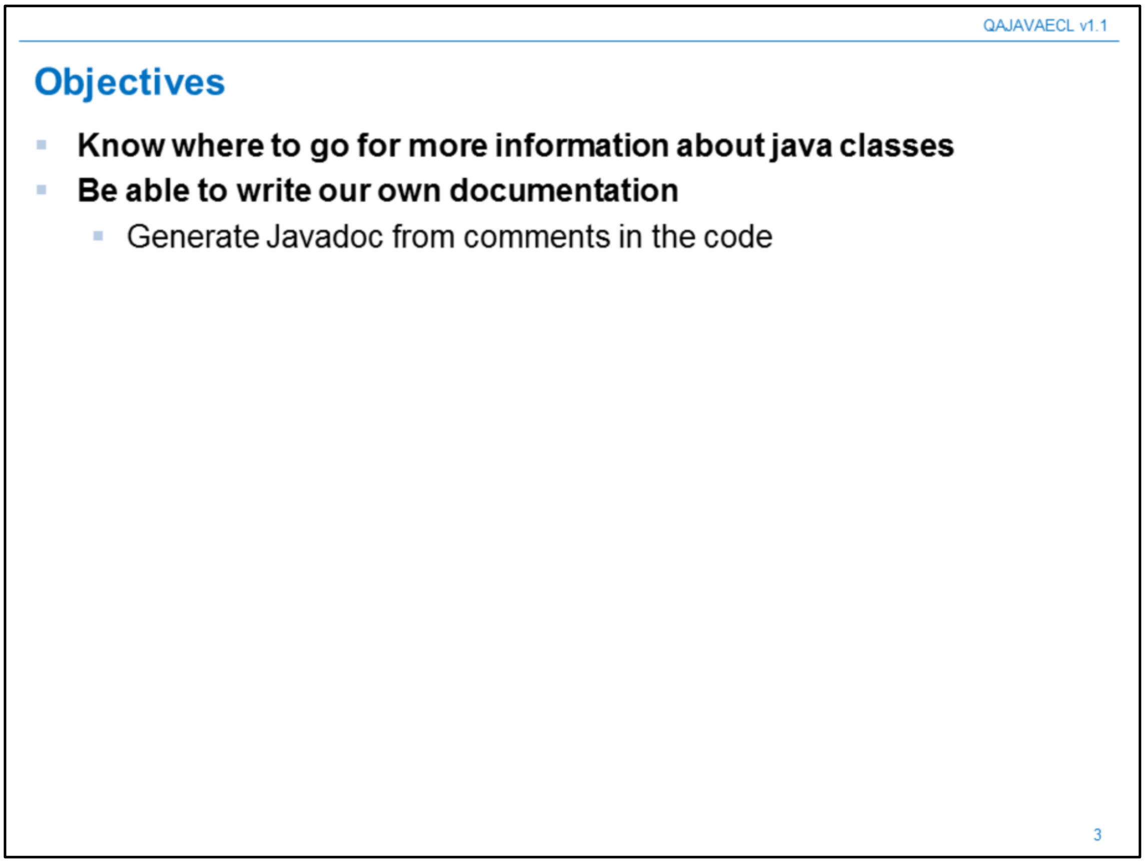


QAJAVAECL v1.1

Outline

- **Javadoc**
  - What is documentation?
  - Why do we want it?
- **The Java API**
  - How to search for new information
- **Writing our own documentation**

2

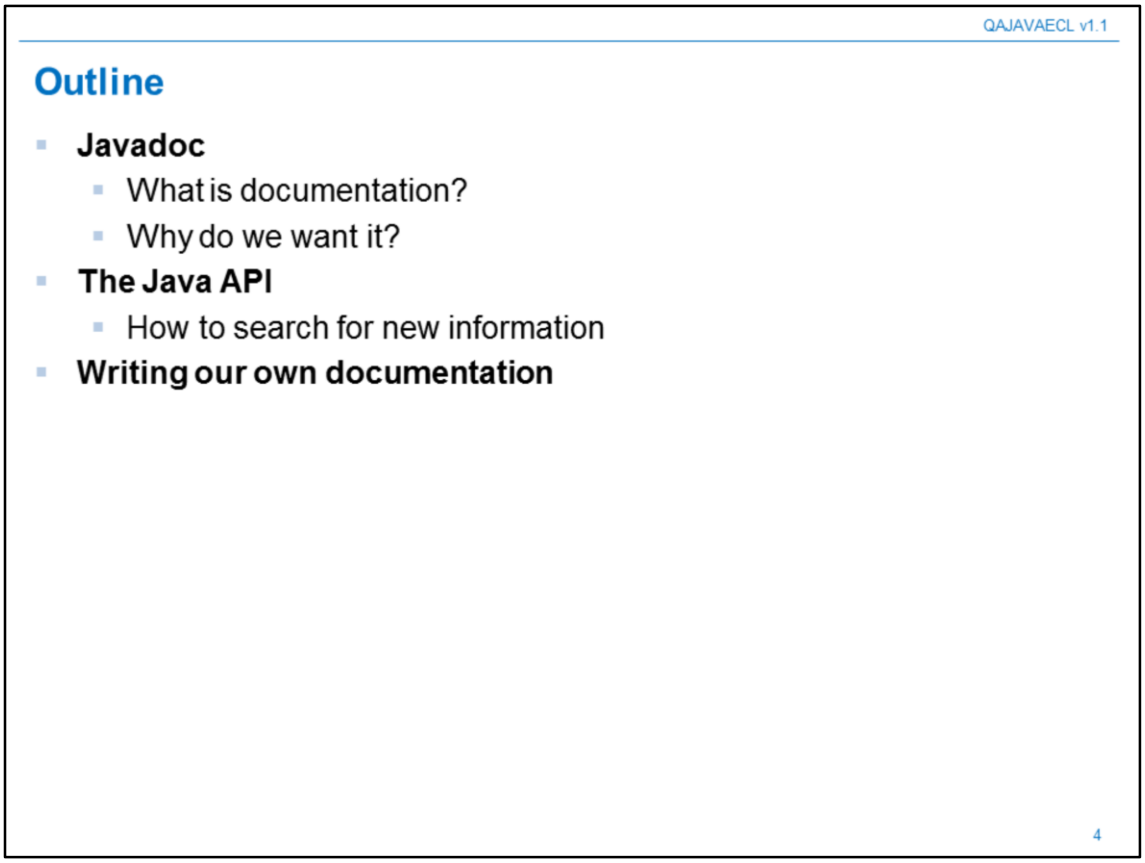


QAJAVAECL v1.1

Objectives

- **Know where to go for more information about java classes**
- **Be able to write our own documentation**
  - Generate Javadoc from comments in the code

3



QAJAVAECL v1.1

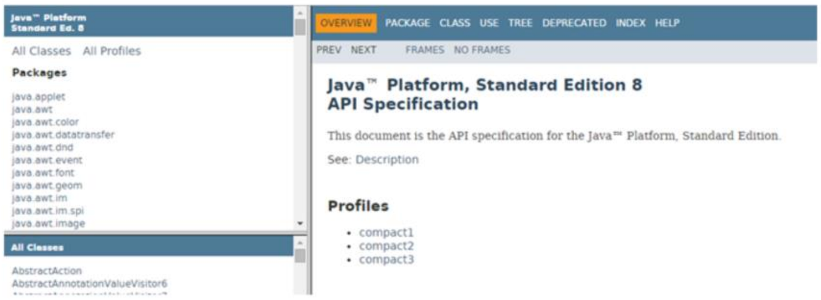
Outline

- **Javadoc**
  - What is documentation?
  - Why do we want it?
- **The Java API**
  - How to search for new information
- **Writing our own documentation**

4

What is Javadoc?

- **Java Documentation**
  - Generated directly from the source code using comments
  - Creates a set of HTML documents
  - Specific format for comments
  - Allows others to understand the classes you have written
  - Also allows us to look up how to use classes in the main API
- **http://docs.oracle.com/javase/8/docs/api/**


Java™ Platform Standard Ed. 8
All Classes All Profiles
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV NEXT FRAMES NO FRAMES

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image

All Classes

AbstractAction
AbstractAnnotationValueVisitor6

Java™ Platform, Standard Edition 8
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

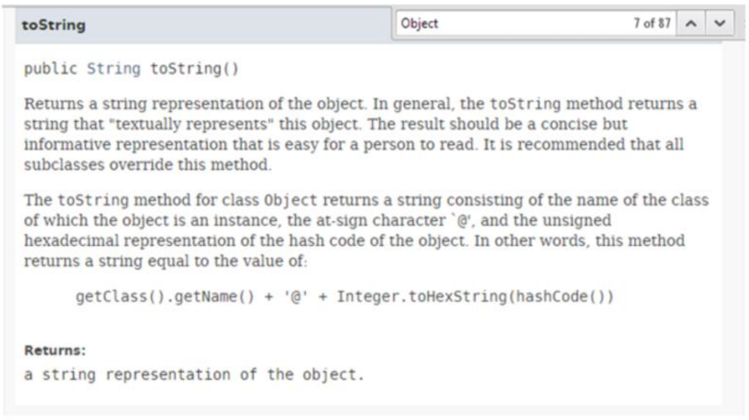
Profiles

- compact1
- compact2
- compact3

There is a very good guide to javadoc on the Oracle website:
http://www.oracle.com/technetwork/articles/java/index-137868.html

Why do we want extensive documentation?

- **Makes your code easier to understand**
  - For you now
  - For you in the future
  - For other people at any point
- **Lets anyone understand what a method does without needing to look at the implementation of that method**



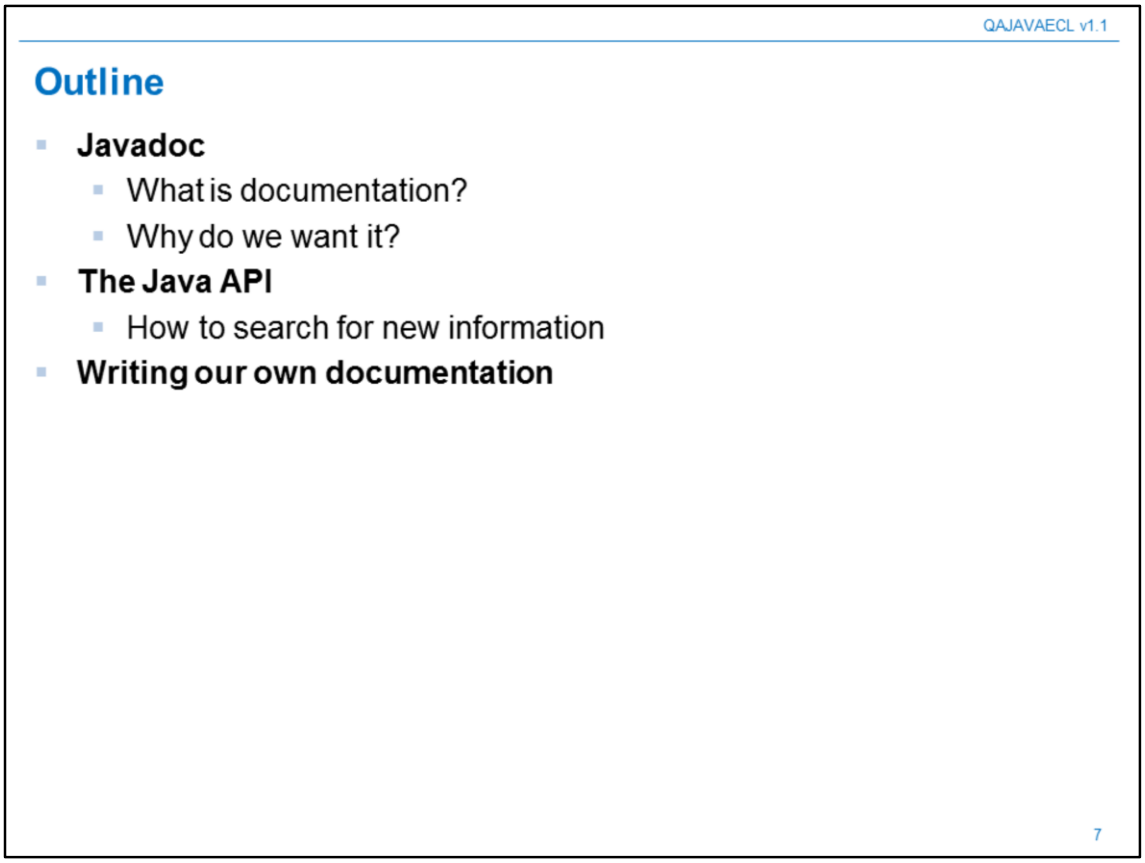
```
public String toString()
```

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of:

```
    getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Returns:
`a string representation of the object.`

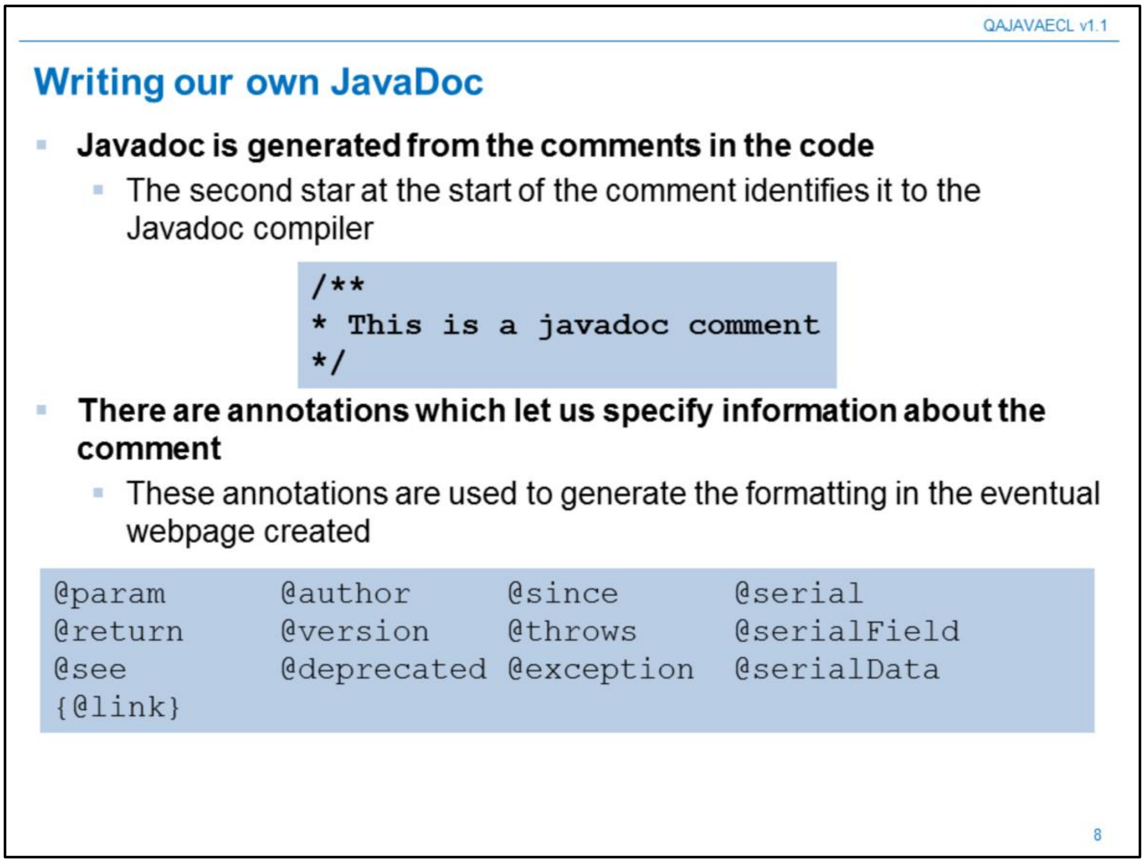


QAJAVAECL v1.1

Outline

- **Javadoc**
  - What is documentation?
  - Why do we want it?
- **The Java API**
  - How to search for new information
- **Writing our own documentation**

7



QAJAVAECL v1.1

Writing our own JavaDoc

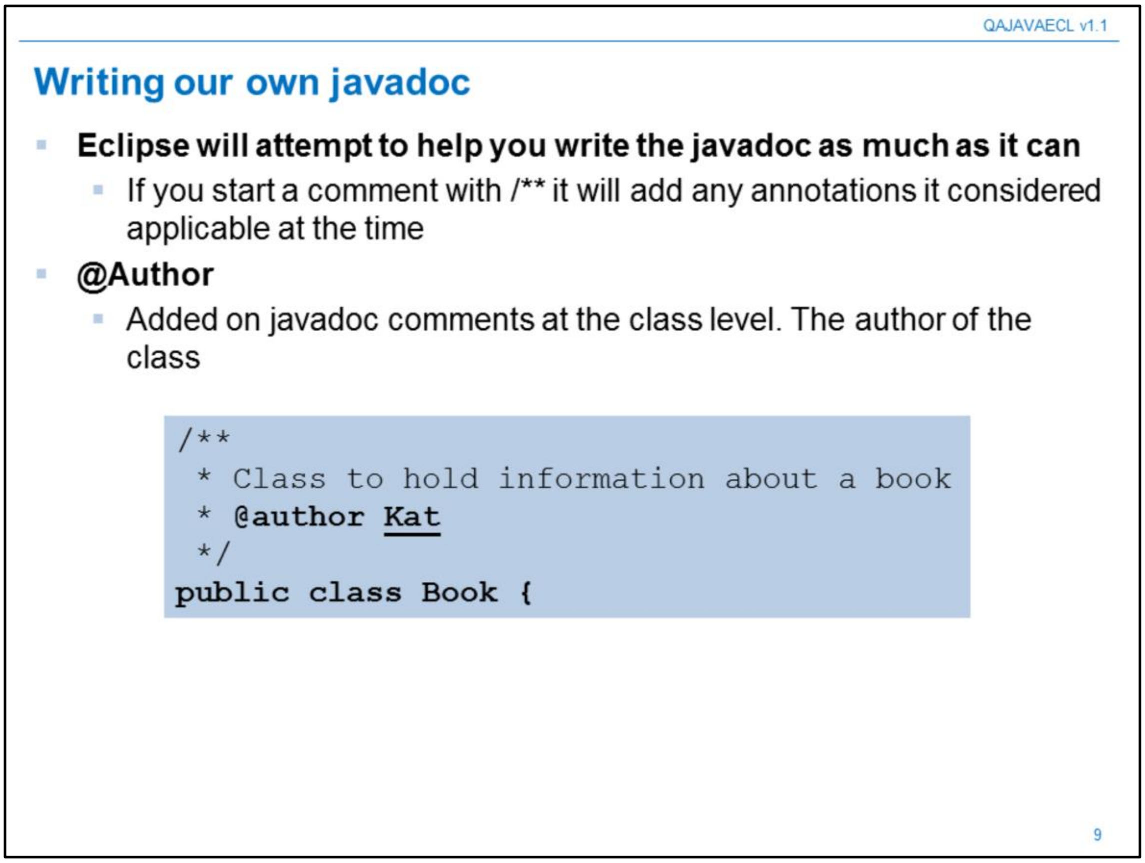
- **Javadoc is generated from the comments in the code**
  - The second star at the start of the comment identifies it to the Javadoc compiler

```
/**
* This is a javadoc comment
*/
```

- **There are annotations which let us specify information about the comment**
  - These annotations are used to generate the formatting in the eventual webpage created

```
@param      @author     @since      @serial
@return     @version    @throws     @serialField
@see        @deprecated @exception  @serialData
{@link}
```

8



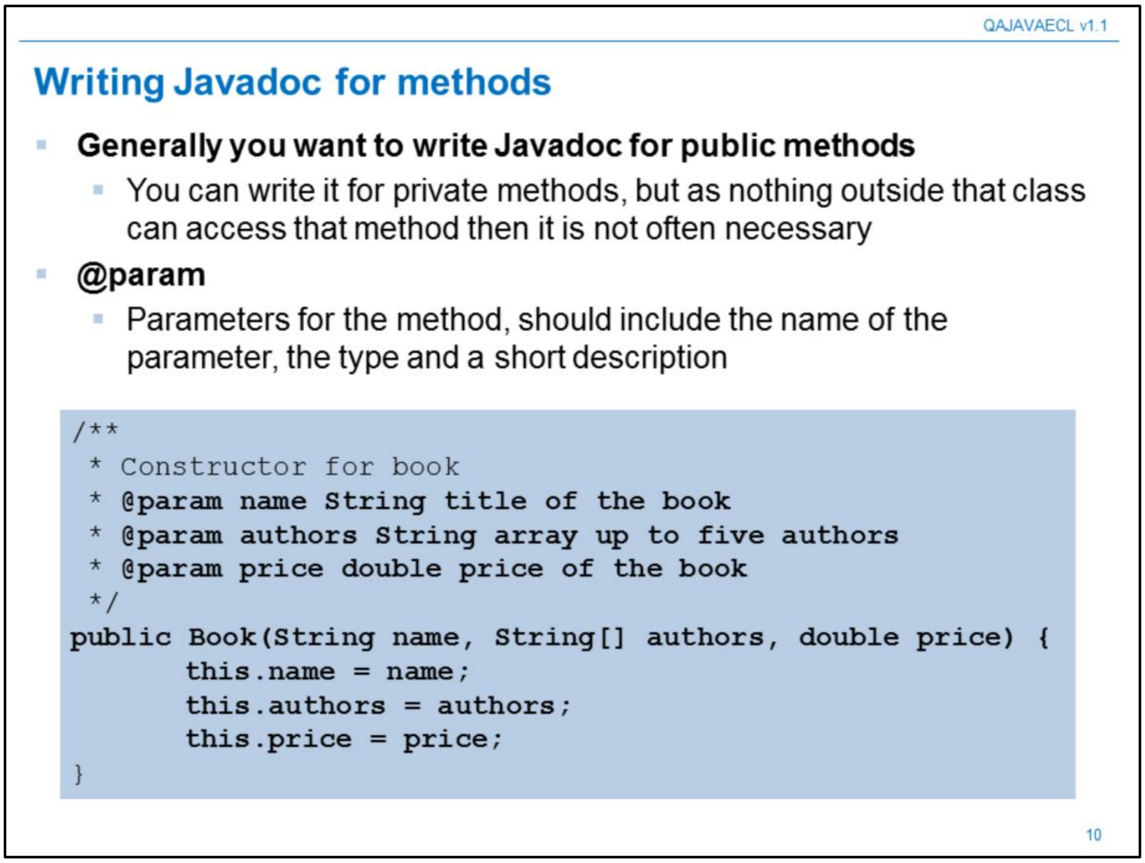
QAJAVAECL v1.1

Writing our own javadoc

- **Eclipse will attempt to help you write the javadoc as much as it can**
  - If you start a comment with /** it will add any annotations it considered applicable at the time
- **@Author**
  - Added on javadoc comments at the class level. The author of the class

```
/**
 * Class to hold information about a book
 * @author Kat
 */
public class Book {
```

9

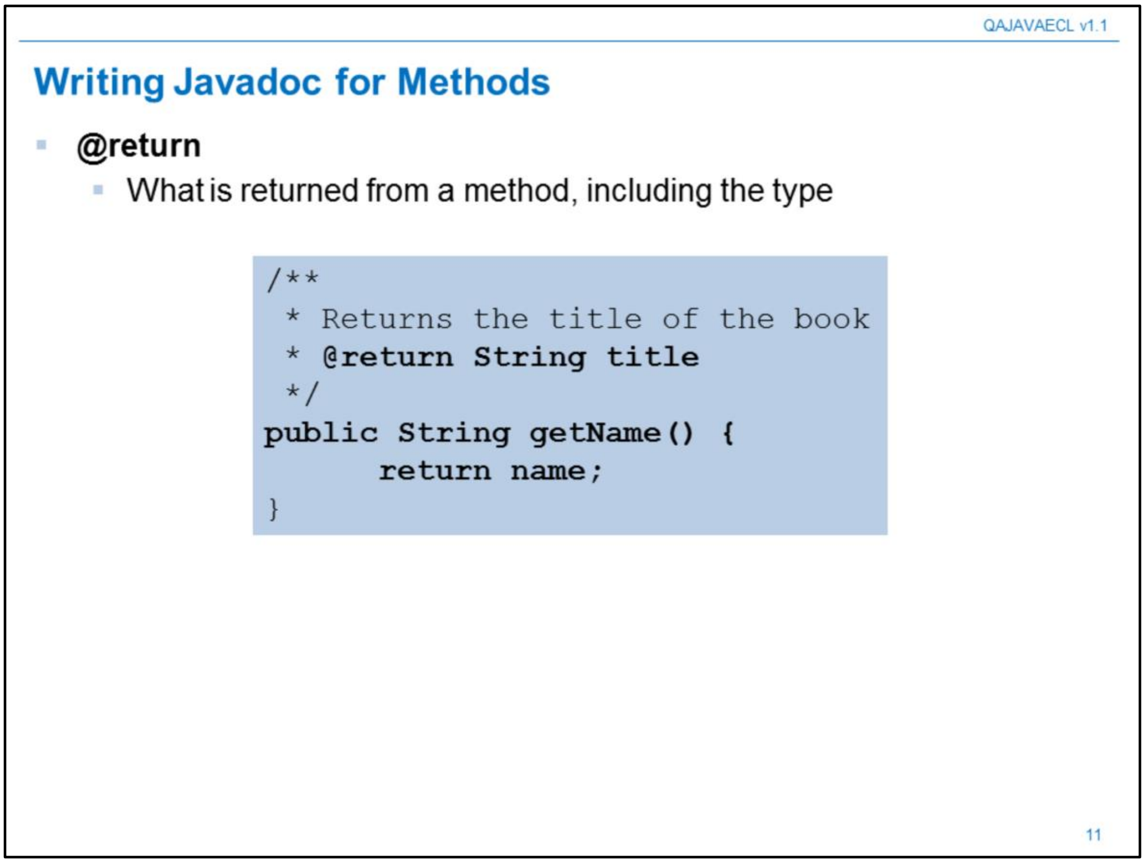


Writing Javadoc for methods

- **Generally you want to write Javadoc for public methods**
  - You can write it for private methods, but as nothing outside that class can access that method then it is not often necessary
- **@param**
  - Parameters for the method, should include the name of the parameter, the type and a short description

```
/**
 * Constructor for book
 * @param name String title of the book
 * @param authors String array up to five authors
 * @param price double price of the book
 */
public Book(String name, String[] authors, double price) {
       this.name = name;
       this.authors = authors;
       this.price = price;
}
```

10



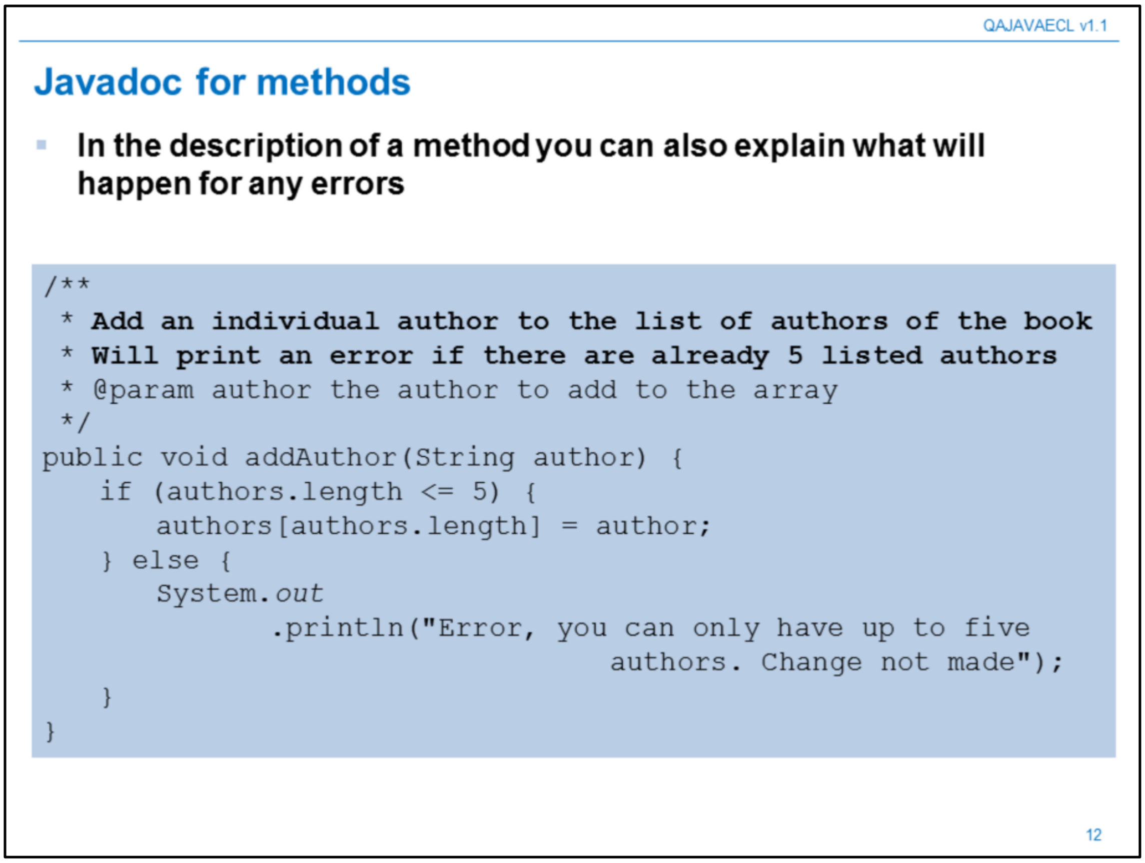
QAJAVAECL v1.1

Writing Javadoc for Methods

- **@return**
  - What is returned from a method, including the type

```
/**
 * Returns the title of the book
 * @return String title
 */
public String getName() {
      return name;
}
```

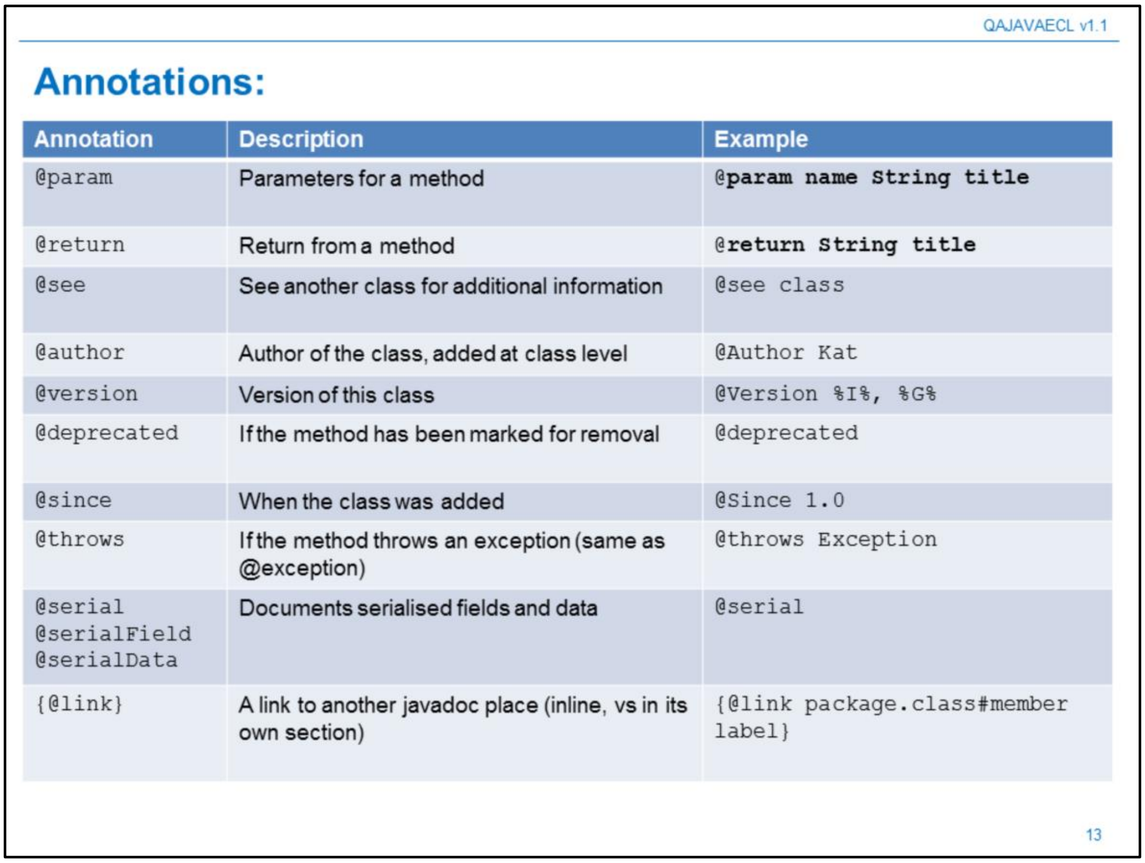
11



Javadoc for methods

- **In the description of a method you can also explain what will happen for any errors**

```
/**
 * Add an individual author to the list of authors of the book
 * Will print an error if there are already 5 listed authors
 * @param author the author to add to the array
 */
public void addAuthor(String author) {
     if (authors.length <= 5) {
          authors[authors.length] = author;
     } else {
          System.out
                    .println("Error, you can only have up to five
                                        authors. Change not made");
     }
}
```



QAJAVAECL v1.1

Annotations:

Annotation	Description	Example
`@param`	Parameters for a method	**`@param name String title`**
`@return`	Return from a method	**`@return String title`**
`@see`	See another class for additional information	`@see class`
`@author`	Author of the class, added at class level	`@Author Kat`
`@version`	Version of this class	`@Version %I%, %G%`
`@deprecated`	If the method has been marked for removal	`@deprecated`
`@since`	When the class was added	`@Since 1.0`
`@throws`	If the method throws an exception (same as @exception)	`@throws Exception`
`@serial` `@serialField` `@serialData`	Documents serialised fields and data	`@serial`
`{@link}`	A link to another javadoc place (inline, vs in its own section)	`{@link package.class#member label}`

13

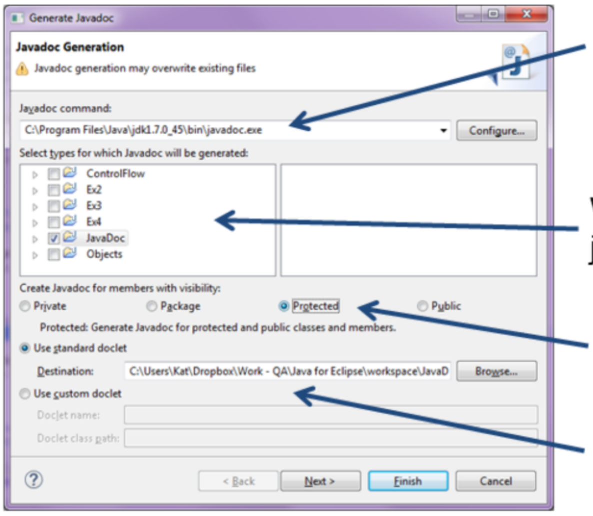
%I% gets incremented each time you edit the file

%G% is the date in the format mm/dd/yy – it can be better to use %G% for dates in the form “December 5th”

We’ll meet serialisation and exceptions later. Serialised fields allow the object to be saved to a file more easily or sent over a communications stream. Exceptions are used for recovering from errors, rather than just exiting the program.

Generating the JavaDoc

- **Eclipse can generate the javadoc for us**
  - Project menu, Generate JavaDoc


location of javadoc.exe
Which projects to generate javadoc for
Scope to use
Where to save the javadoc to

Generate Javadoc

Javadoc Generation

⚠ Javadoc generation may overwrite existing files

Javadoc command: C:\Program Files\Java\jdk1.7.0_45\bin\javadoc.exe

Select types for which Javadoc will be generated:
- ☐ ControlFlow
- ☐ Ex2
- ☐ Ex3
- ☐ Ex4
- ☑ JavaDoc
- ☐ Objects

Create Javadoc for members with visibility: Private Package [✓ Protected] Public

Protected: Generate Javadoc for protected and public classes and members.

[✓ Use standard doclet]

Destination: C:\Users\Kat\Dropbox\Work - QA\Java for Eclipse\workspace\JavaD

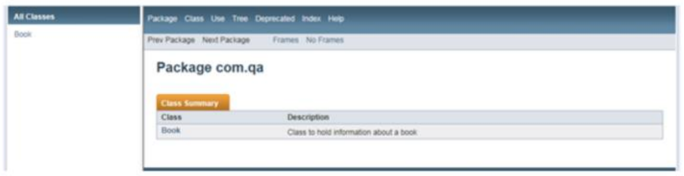
Use custom doclet

Doclet name:

Doclet class path:

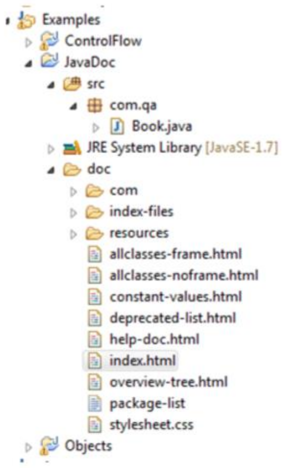
Generating the JavaDoc

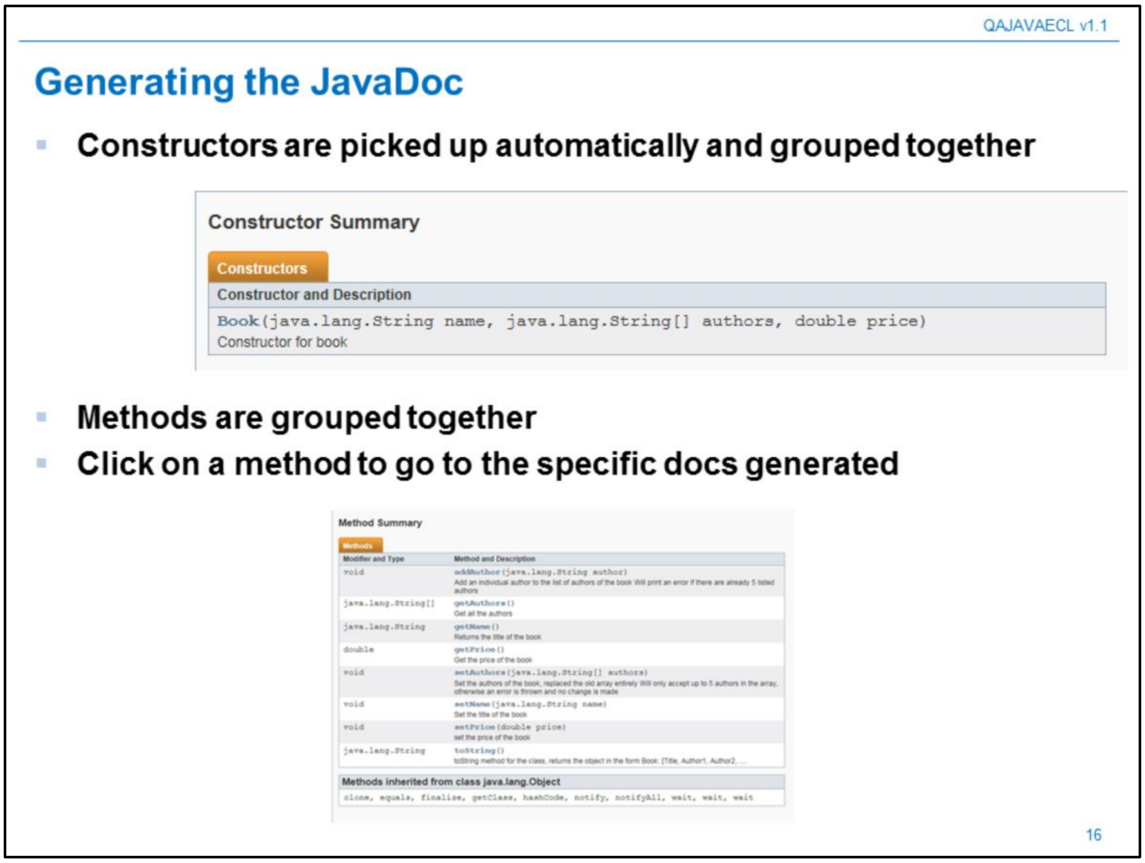
- **The console will tell you of any errors in your javadoc comments**
- **The html files will be generated for you regardless based on the comments you provided**
- **Right click 'index.html'**
  - Open with – Web browser
  - This will open the file in eclipse
  - You could also open it outside eclipse by going to the file directly


All Classes
Book
Package Class Use Tree Deprecated Index Help
Prev Package Next Package Frames No Frames
Class Summary

Package com.qa

Class	Description
Book	Class to hold information about a book





QAJAVAECL v1.1

Generating the JavaDoc

- Constructors are picked up automatically and grouped together

Constructor Summary

Constructors

Constructor and Description
`Book(java.lang.String name, java.lang.String[] authors, double price)` Constructor for book

- Methods are grouped together
- Click on a method to go to the specific docs generated

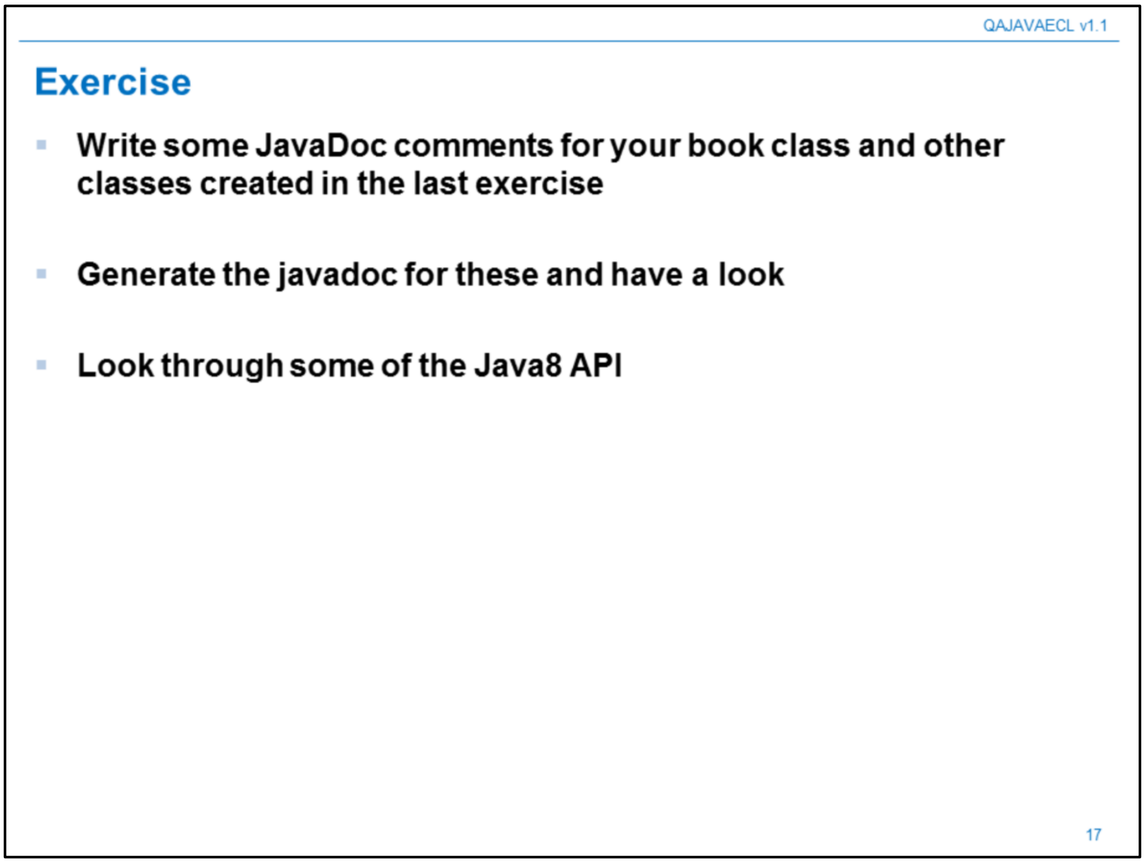
Method Summary

Methods

Modifier and Type	Method and Description
`void`	`addAuthor(java.lang.String author)` Add an individual author to the list of authors of the book Will print an error if there are already 5 listed authors
`java.lang.String[]`	`getAuthors()` Get all the authors
`java.lang.String`	`getName()` Returns the title of the book
`double`	`getPrice()` Get the price of the book
`void`	`setAuthors(java.lang.String[] authors)` Set the authors of the book, replaced the old array entirely Will only accept up to 5 authors in the array, otherwise an error is thrown and no change is made
`void`	`setName(java.lang.String name)` Set the title of the book
`void`	`setPrice(double price)` set the price of the book
`java.lang.String`	`toString()` toString method for the class, returns the object in the form Book: [Title, Author1, Author2, ...

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

16

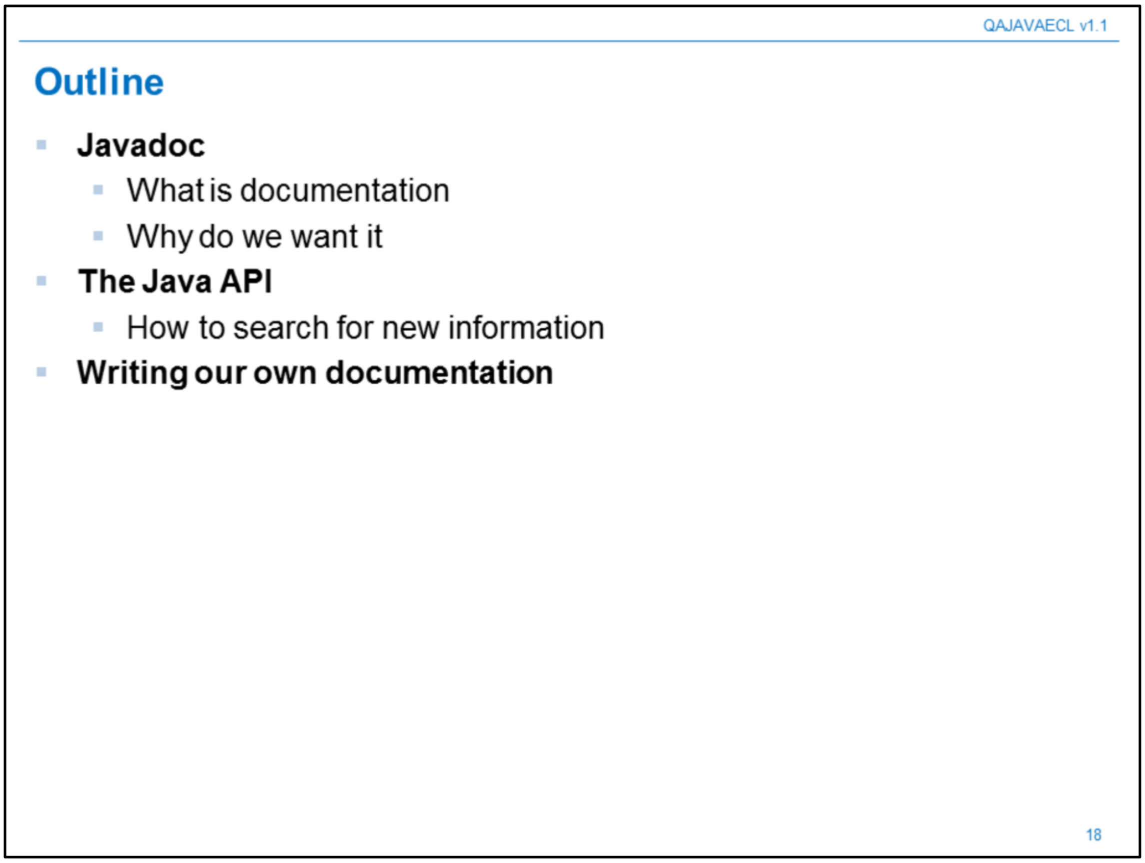


QAJAVAECL v1.1

Exercise

- Write some JavaDoc comments for your book class and other classes created in the last exercise
- Generate the javadoc for these and have a look
- Look through some of the Java8 API

17



QAJAVAECL v1.1

Outline

- **Javadoc**
  - What is documentation
  - Why do we want it
- **The Java API**
  - How to search for new information
- **Writing our own documentation**

18