Hyperiondev

TASK

Software Documentation

Visit our website

Introduction

WELCOME TO THE SOFTWARE DOCUMENTATION TASK!

Reliable documentation is essential for any Software Engineer. Documentation helps keep track of all aspects of an application and it improves the quality of a software product. Successful documentation will make information easily accessible, it helps transfer knowledge to other developers, simplifies the product and helps with maintenance. In this task, we discuss both external and internal documentation.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

DOCUMENTATION IS KEY!

Code as if whoever maintains your program is a violent psychopath who knows where you live.
—Anonymous

We've all, at some point or another, tried to put together a toy, an appliance or even cook a meal without following the instructions. Unless you have a natural flair for mechanics or cooking, you tend to run into problems. In programming, documentation is the manual, or recipe, you write so that the next person looking at your code is able to follow it and understand it. This is done on two different levels: broadly, how to use the software, and more specifically, how the software itself works.

The first level is known as **external documentation**. This can include a separate document that explains to the user how to use the software, known as **user documentation**. It could also use folders, such as Unit Development Folders (UDFs), which document the developer's notes during creation, such as particular design decisions that have been taken and tools that have been used. This is known as **library documentation**. In the UDF, there could be a Detailed-Design Document (DDD): a low-level document that shows design decisions on a class level, as well as what possibilities were considered and why the final decision was taken. Ironically, not all external documentation is outside the code. A DDD could actually be in the code itself.

In comparison, **internal documentation**, the most detailed form of documentation, is always within the code. It is the documentation that you have been marked on in your tasks up until this point: comments and programming style. Because you are already familiar with these, we will first look at internal documentation, and then discuss how we go about generating external documentation.

INTERNAL DOCUMENTATION

Good Programming Style

Have a look at the code below. Can you figure out what's going on?

```
int a=85;char b;
if (a>=80) { b='A';} else if (a>=70&&a<80) {b='B';}
else if (a>=60&&a<70) {b='C';} else if (a>=50&&a<60) {b='D';}
else { b='F';} System.out.println(b);
```

I'm sure you can see it has something to do with determining if something is A, B, C, D or F, so you may correctly deduce it has something to do with grades, but have a look how much easier it is to understand in the code below:

```
      // Determines a student's academic symbol based on their grade (%)

int grade = 85;
      char symbol;

      if (grade >= 80) {
          symbol = 'A';
      }
      else if (grade >= 70 && grade < 80) {
          symbol = 'B';
      }
      else if (grade >=60 && grade <70) {
          symbol = 'C';
      }
      else if (grade >=50 && grade <60) {
          symbol = 'D';
      }
      else {
          symbol = 'F';
      }
      System.out.println(symbol);
```

Now that we have refactored the code by using descriptive variable names, the correct spacing and indentation, and a summary comment at the top, it takes very little effort to understand the code. While this is a simple example, it illustrates how vital good programming style is as a form of internal documentation, especially when the code becomes more complicated. Let us now look at comments in more detail.

Comments

When you have a good programming style, comments that describe each line of code (e.g. `System.out.println(symbol); // this prints the determined symbol`) is not necessary. However, comments have some important functions, including:

1. *Explaining the code:* for a particularly tricky piece of code, this is intended to help someone reading it to follow your logic and understand the process.

2. *Summarising the code:* As with the example above, this is a simple summary of the overall process of the code.

3. *Describing what the code is meant to be doing:* This indicates to the reader that a piece of the code is not doing what it should be, and possibly needs to be reworked.

4. *Code markers:* This is used to show where code still needs to be completed, and is usually indicated by a string of ####################, ********** or !!!!!!!!!!!!!!!!!!!!!!!!!!!!. Code markers can also be used to separate logical sections of code.

5. *Commented out code for debugging:* This is code that is commented out until it is time to debug. For example, where a variable is usually assigned a value from user input, that line could be commented out and the value could be hard-coded for the sake of debugging. Once the debugging process is completed, these comments will usually be deleted. See below:

```
// Scanner input = new Scanner(System.in);
// int number = input.nextInt();
int number = 6;
System.out.println(number);
```

6. *Commented out code that is not meant to be run (at this time):* If a programmer is working on a piece of code, they may be trying different approaches to find the most efficient or understandable way forward. In this time there might be pieces of code that are commented out. Once they have decided on the best way, the unused code will be deleted.

7. *Information that needs to be in the code, but cannot be expressed by the code itself:* These include comments about copyright, confidentiality, etc.

In Java, three types of comments are supported:

1. **`// text`**: ignores all content from `//` to the end of the line
2. **`/* text */`**: ignores all content from `/` to `/`. This is particularly useful when comments take up multiple lines
3. **`/** text */`**: similar to above, but is used specifically in Javadoc, which will be discussed below.

Now let's take a closer look at external documentation in Java.

EXTERNAL DOCUMENTATION

As you can imagine, writing out an entirely separate document for your code could be a gruelling task. Fortunately, though, there are some tools that can help. In Java, we can use Javadoc, a documentation generator that is part of the Java Development Kit (JDK). It creates documentation in HTML format so that you can hyperlink related documents together. What is particularly nice about Javadoc is that you write the documentation in the code, but it creates a separate Javadoc file with all the information, and the comments are removed during compilation so it does not reduce the performance of your code! Guidelines for the format of Javadoc are exemplified in the example code below:

```
/**
* This code combines a person's first name and surname
* <p>
* It is also a simple example to show how Javadoc works
* @author Iron Man
* @version 2.1
*/

public class Names {
    /**
     * String value for first name
     */
    public String firstName;
    /**
     * String value for last name
     */
    public String lastName;
}

/**
 *
 * @param first name value
 * @param last surname value
 * @return returns full name
 */
public String fullName (String first, String last) {
    return first + " " + last;
}
```

- As you can see, each time you want to add a comment for Javadoc, you use the format `/** ... */` with an `*` at the beginning of each line.
- You may have noticed the `<p>` on line 3. This is HTML for a paragraph break.

- On lines 5 and 6, you can see examples of using tags. Some available tags are listed at the end of this task.
- In the class Names, you will see that we can use comments to describe the declared variables. These go **above** the variable declarations.
- For the function fullName, the `@param` tag is used so that the details of each variable can be given and the `@return` tag is used to document what the function returns

You can include specific Javadoc tags. While the most useful tags that document code are `@exception`, `@throw`, `@param` and `@return`, the full list of available Javadoc tags are in the table below:

Tag	Description	Syntax
@author	Describes an author	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML markup or nested Javadoc tags.	{@code text}
{@docRoot}	Represents the relative path to the generated document's root directory from any generated page.	{@docRoot}
@deprecated	Describes an outdated method.	@deprecated deprecatedtext
@exception	Adds a Throws subheading to the generated documentation, with the class name and description text.	@exception class-name description
@inheritDoc	Inherits a comment from the nearest inheritable class or implementable interface.	{@inheritDoc}

@link	Link to another Javadoc entry.	{@link reference}
@linkplain	Identical to {@link}, except the link's label is displayed in plain text rather than code font.	{@linkplain reference}
@param	Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section.	@param parameter-name description
@return	Adds a "Returns" section with the description text.	@return description
@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@serial	Used in the doc comment for a default serializable field.	@serial field-description | include | exclude
@serialData	Documents the data written by the writeObject( ) or writeExternal( ) methods.	@serialData data-description
@serialField	Documents an ObjectStreamField component.	@serialField field-name field-type field-description
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release

@throws	The @throws and @exception tags are synonyms.	@throws class-name description
@value	Return the value of a static field..	{@value #STATIC_FIELD}
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

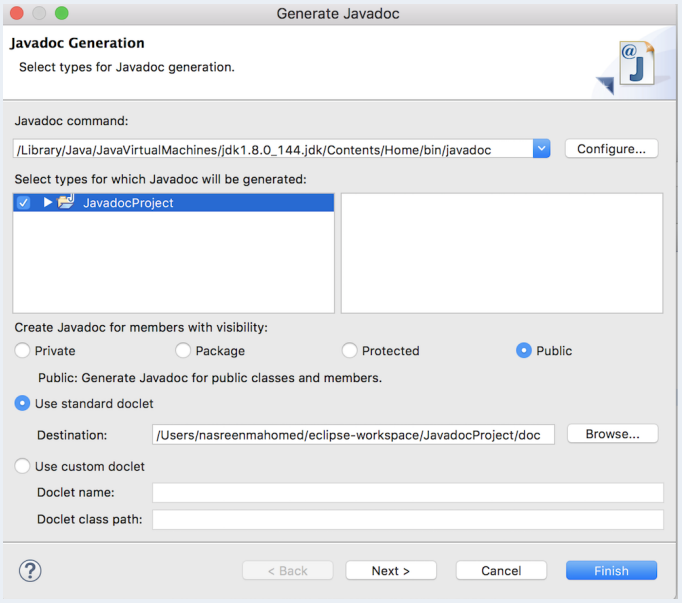
More information on how to write doc comments can be found **here**.

Compulsory Task

Follow these steps:

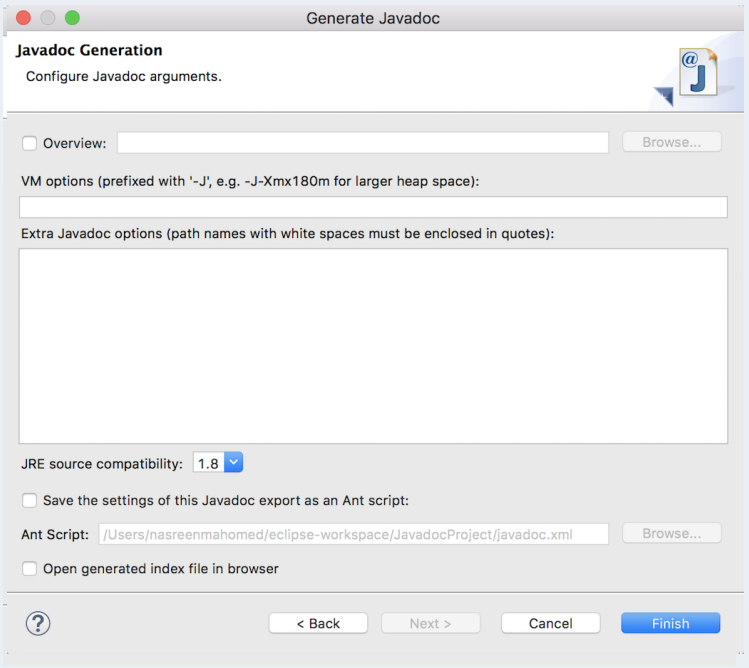
- Eclipse allows you to easily generate Javadoc documentation. In this task we demonstrate how to automatically generate Javadoc documentation.
- Open the Project JavadocProj in Eclipse.
- Select Project → Generate JavaDoc from the menu to open the Javadoc generation wizard.
- At first step of the wizard, you can define settings for:
    - path for the **javadoc.exe** tool from the JDK
    - project resources for which to generate JavaDoc;
    - classes and methods for which to generate JavaDoc based on their visibility;
    - location of the JavaDoc (by default it will be placed in the doc folder in the project location)

  Leave all the settings as they are for now

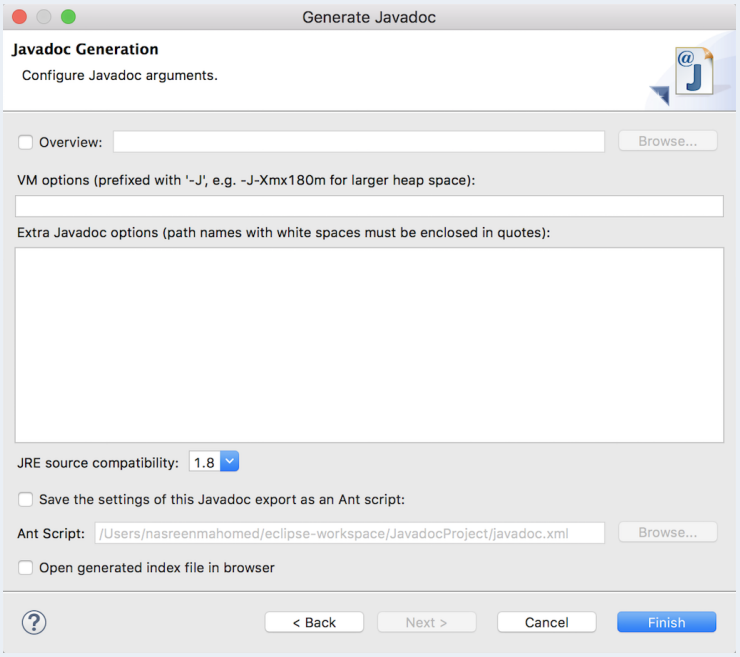


- Click on the Next button to go on to the next step of the wizard. At this step, you can define settings regarding:
    - Documentation structure
    - JavaDoc tags to be processed
    - other resources(archives, projects) used in the project to be included in the documentation
    - another CSS style sheet for the documentation

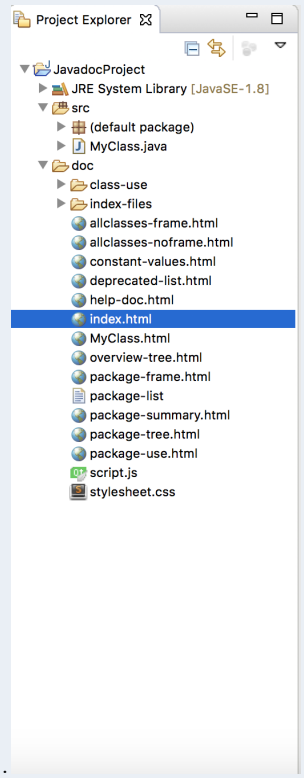
  Leave all the settings as they are for now



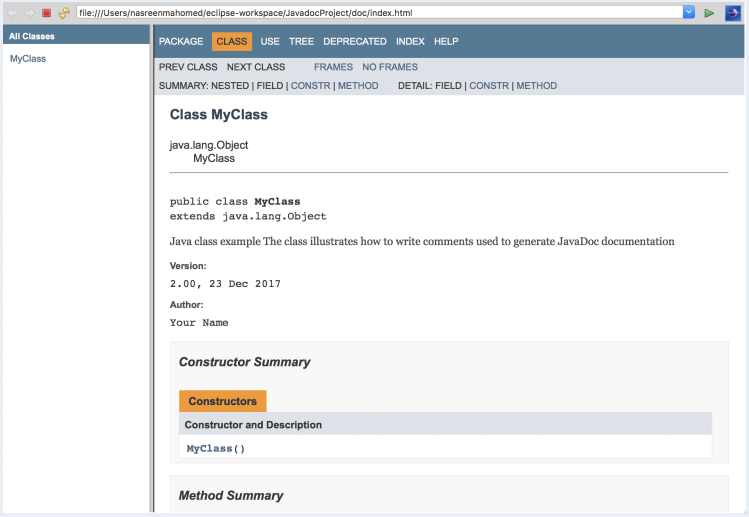
- Click on the Next button again to go to the last step of the wizard. At this step you can save the settings in an Ant script for future use. Leave all the settings as they are for now.



- Click on the Finish button to generate the Javadoc.
- You should now see a doc folder in your Package Explorer. Select index.html form this folder



- You should now see your newly generated Javadoc.



Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.