by Mari Chelo Rubio

Development Environments
Block 3
Unit 6: Software documentation. Tools

3.6.1. Introduction

It doesn't matter how good software is... if documentation is not good enough, people will not use it. And, if it has to be used, unless the documentation is good, it will not be used properly.

In order to have a good documentation, we have to take into account the following types of documentation, so that we can choose the most appropriate one according to its final purpose.

3.6.2. Documentation types

According to how we are going to use the documentation, it can be:

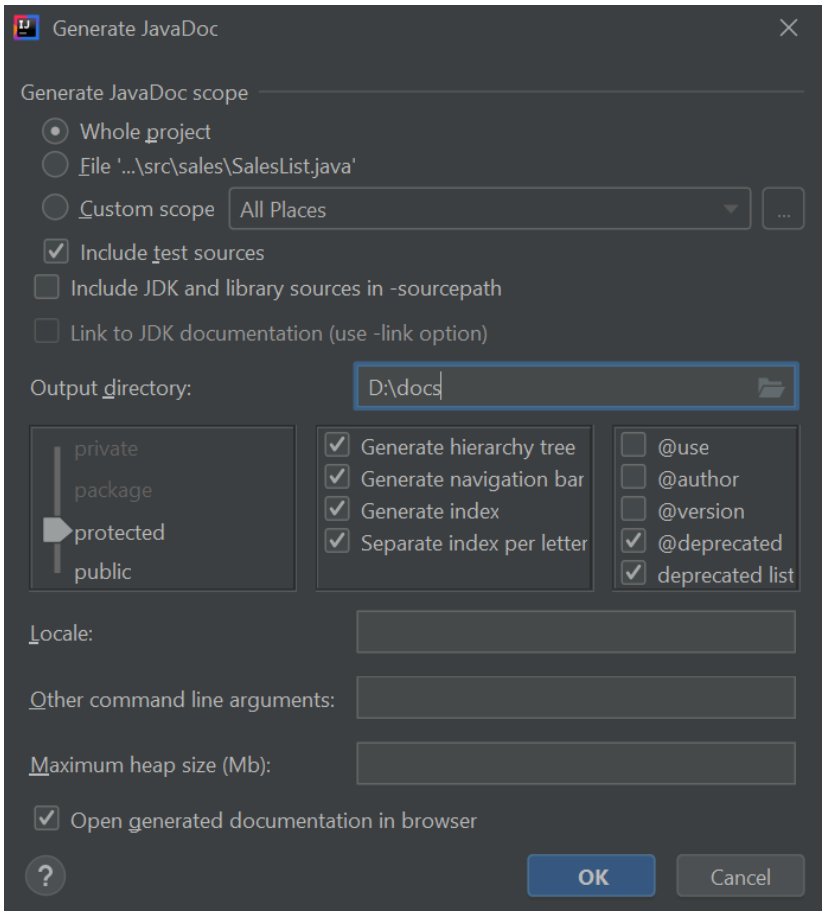
- **Tutorials**
  - Learning-oriented
  - They let people get started
  - It's a lesson
  - Analogy: teach a child to cook
- **How-to guides**
  - Results-oriented
  - They show how to solve a problem or a concrete need
  - It's a list of steps or stages
  - Analogy: recipe of a cooking book
- **Explanations**
  - Understanding-oriented
  - They explain something
  - They offer some context
  - Analogy: an article about culinary social history
- **Reference**
  - Information-oriented
  - It describes how something has been built
  - It's exact and complete
  - Analogy: an entry in an encyclopedia

3.6.3. How to create a Reference for Java with JavaDoc

Documenting a software project is essential for its future maintenance. When we are coding a class, we must generate detailed documentation about it so that other programmers can use it properly without checking its internal implementation. The same happens whenever we use the Java API in our applications, we don't need to check the internal code of the classes to understand what they do.

Javadoc is an Oracle utility to generate API documentation in HTML format from Java source files. It is the standard for documenting Java classes, and most of the IDEs use it to automatically generate this documentation.

In order to generate *javadoc* documentation, we just need to add some comments with a particular syntax in the code. This way, Javadoc generates a reference similar to the Java official one. Once we are done, we must go to *Tools* > *Generate JavaDoc* menu in IntelliJ to generate this documentation. Then, we choose the output folder and the elements for which we want to generate the documentation (typically, the whole project).



After this, you can find the documentation in the output folder that you have specified in this dialog.

> **NOTE**: if you see an error message indicating that IntelliJ doesn't have the PATH properly set to run *javadoc*, you may need to update your version of IntelliJ and/or right click on the project, go to *Open Module Settings* and then, go to the *SDK* option and update the path to your preferred JDK (for instance, *C:\openjdk11*).

Now, let's have a look at what we need to comment in order to generate an appropriate *javadoc* documentation, and how these comments must be.

3.6.3.1. Required data

The required information to document a class are:

- The class name, general description, version number and author(s).
- Documentation of every constructor or method (specially the public ones), including the constructor or method name, general description, return type and description of this return value, and finally, name, type and description of every parameter.

3.6.3.2. Javadoc comments

Javadoc documentation must be included in comments starting with `/**` and finishing with `*/`. Depending on where we are placing the comment, Javadoc considers it a class, method or constructor comment. Many popular IDEs auto-complete this kind of comments with some default structure, so that we just need to fill the fields.

```
import java.time.LocalDate;
import java.time.Period;

/**
 * Class to define people and their ages
 * @author mariaconsuelorubiosanchez
 * @version 2.1
 */
public class Person implements Comparable<Person> {
    ...
}
```

In the code above we have defined a class comment. It includes the general definition of the class, along with some special annotations that Javadoc interprets to generate the documentation. We can se an `@author` and `@version` annotations, that define the author and version of the class file, respectively. We can also add some other annotations, such as `@see` to link with other related classes or web sites, or `@since` to establish the Java version required to use this class, or the date where the class was firstly created.

```
/**
 * Class to define people and their ages
 * @author mariaconsuelorubiosanchez
 * @version 2.1
 * @see https://iessanvicente.com
 * @since 01/04/20019
 *
 */
```

After this general comment, we can also comment the methods of the class. In these other comments we will use the following annotations:

- `@return` to specify the description of the return type
- `@param` to specify the name and description of every parameter

Let's see some examples. This is how we can document a constructor:

```
/**
* Constructor with parameters
* @param name A String with the person name
* @param idCard A String with the person's id card
* @param address A String with the person's address
* @param phoneNumber A String with the person's phone number
* @param birthDate A String with the person's birth date
with format dd/mm/yyyy
* @return It will return a Person instance with the specified
attributes
*/
public Person(String name,String idCard,String address,
              String phoneNumber,String birthDate)
{
    ...
}
```

This is the documentation for getters and setters:

```
/**
* Returns the person's name
* @return Person's name
*/
public String getName() {
    return name;
}

/**
* Establishes the person's name
* @param name Person's name
*/
public void setName(String name) {
    this.name = name;
}
```

And this is how we would document other method types:

```
/**
* Method that calculates the age of a person from his/her
* birth date and current date
* @return An integer with the age in years of this person
* currently
*/
public int calculateAge()
{
    ...
}
```

Once we have generated all the comments, we can generate the Javadoc documentation. Then, we will get an HTML page for each documented class. We can also add HTML tags in our comments. For instance

```
/**
 * <h1>Person class<h1>
 * <p>Class to define people and their ages</p>
 * @author mariaconsuelorubiosanchez
 * @version 2.1
 * @see https://iessanvicente.com
 * @since 01/04/20019
 */
```

> **Proposed exercises:**
>
> **3.6.3.1.** Document the classes of the *Animals* project from Unit 7 of Block 2.

3.6.3.2. Document the classes of the *CulturalOrganization* project from Unit 7 of Block 2.

3.6.4. How to create a Reference for C#

Regarding C#, we also need to document the classes and their public elements in the project. Documentation comments in C# start with a `///`. If we place these symbols right before the element we need to comment, then Visual Studio auto-completes part of the information to be added.

So, if we start a documentation comment before a class name, then Visual Studio lets a structure to enter a summary for this class:

```
/// <summary>
/// Abstract class to represent the general features of every animal
/// </summary>
class Animal
{
    ...
}
```

If we put this comment before any constructor or method, then a `<summary>` tag is automatically added as well, along with a `<param>` name for every parameter of this method/constructor, and a `<returns>` section if the method returns anything, to specify the value that is returned.

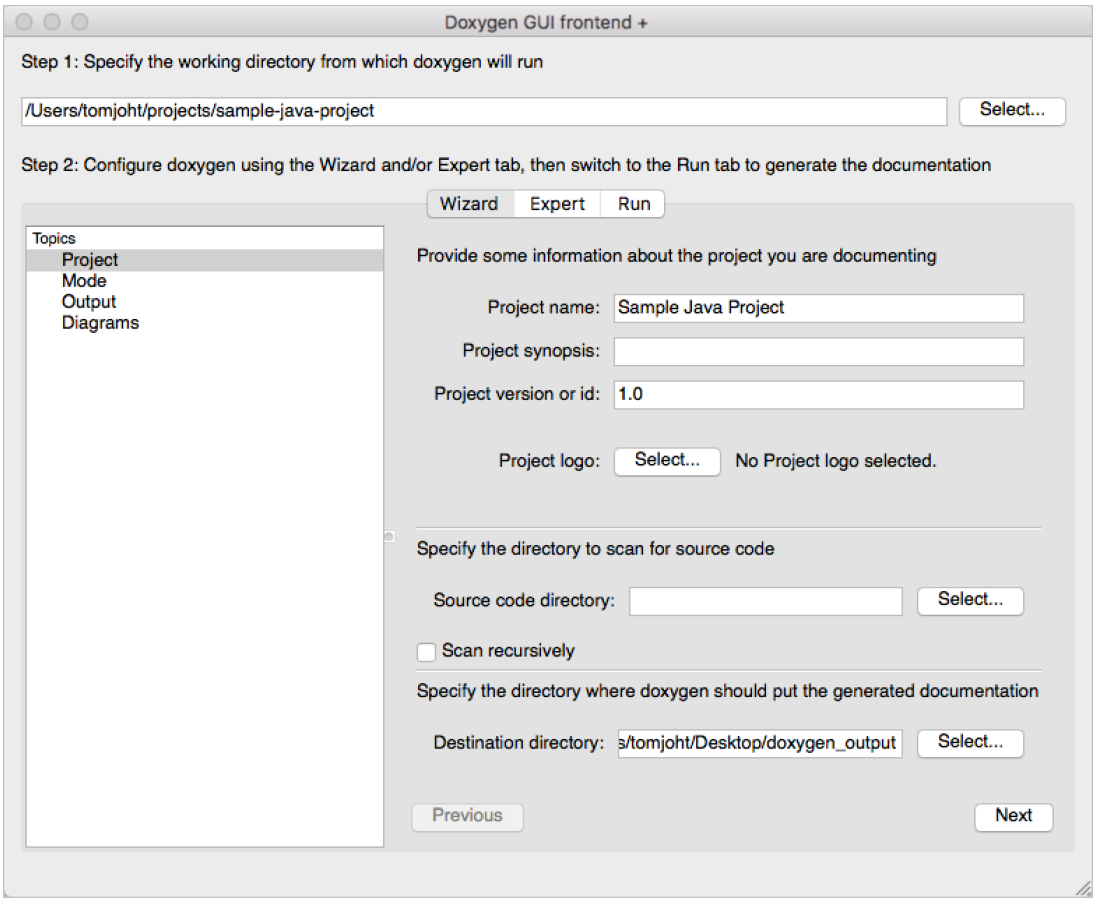
```
    /// <summary>
    /// Constructor to create any type of animal
    /// </summary>
    /// <param name="name">Animal name</param>
    /// <param name="color">Animal color</param>
    public Animal(String name, String color)
    {
        this.name = name;
        this.color = color;
    }
```

3.6.4.1. Generating the documentation with Doxygen

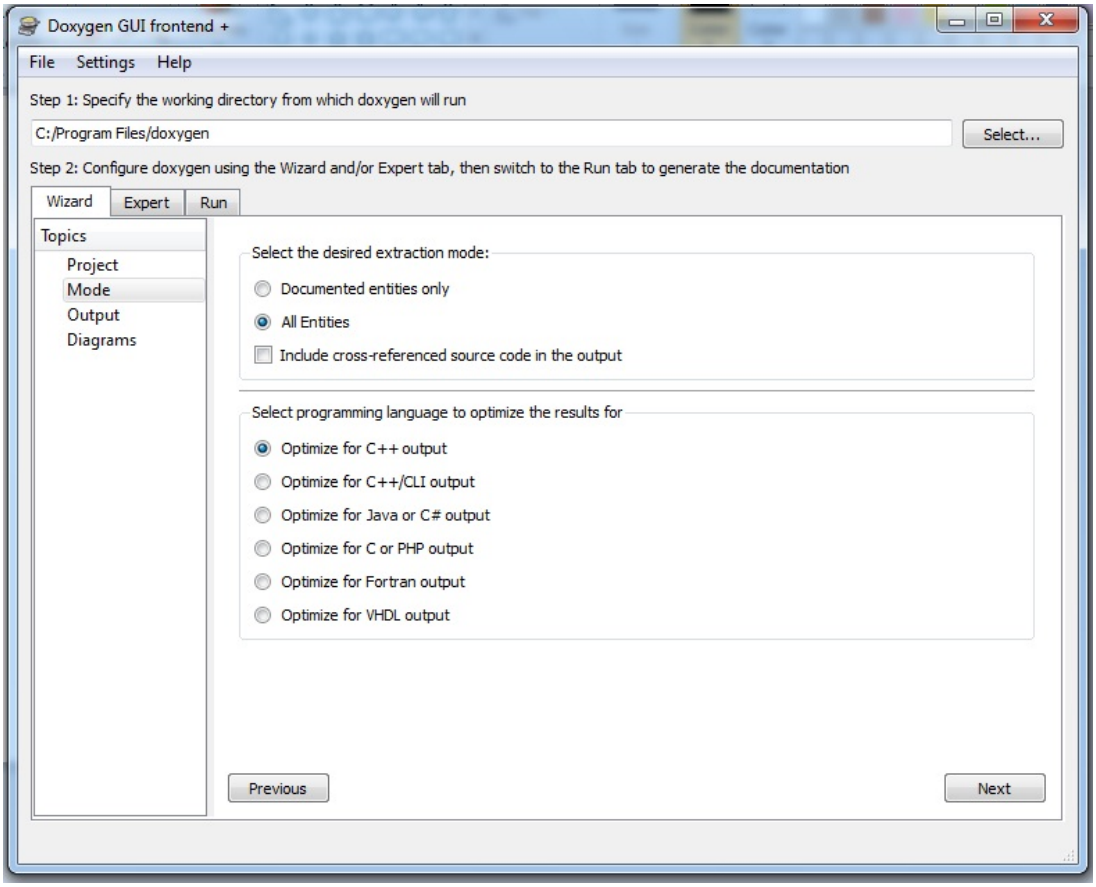
In order to generate the documentation of a C# project, Visual Studio does not include a built-in tool for this, and we need to rely on an external tool. The most popular one is **Doxygen**, an application to generate documentation from source code written in many languages, such as C, C++, C#... and even PHP or Java.

You can download Doxygen from its official web site, in the *Downloads* section. Regarding Windows, it consists in running an installer.

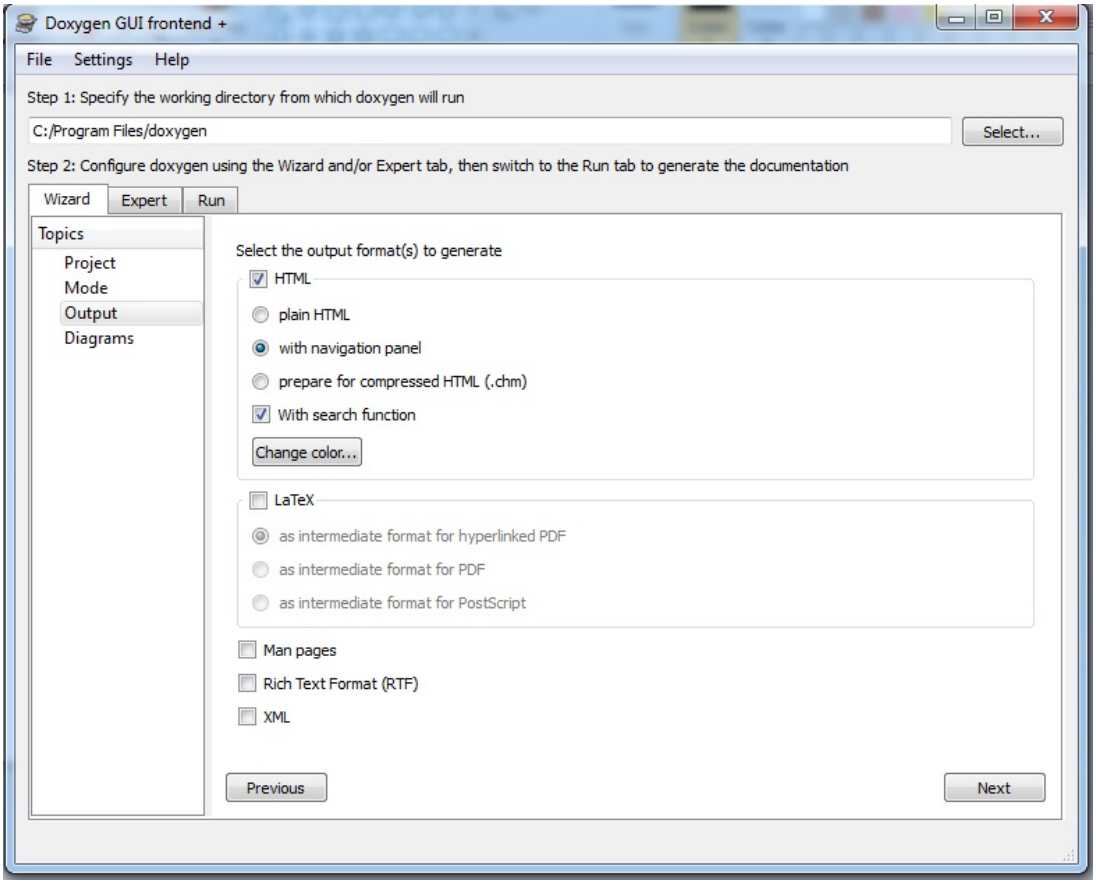
Then, we must launch the Doxygen wizard (also known as *DoxyWizard*), which will show an initial screen like this one:



In the upper text field we must choose the folder in which Doxygen has been installed (typically `C:\Program Files\doxygen`). In the lower text fields, we can just indicate the source folder to check the files (we can check the option to scan it recursively), and the destination folder to generate the documentation in.



We usually choose to include *All Entities*, and then the programming language that we are using (in our case, *Optimize for Java or C# output*). Then we move to next step.

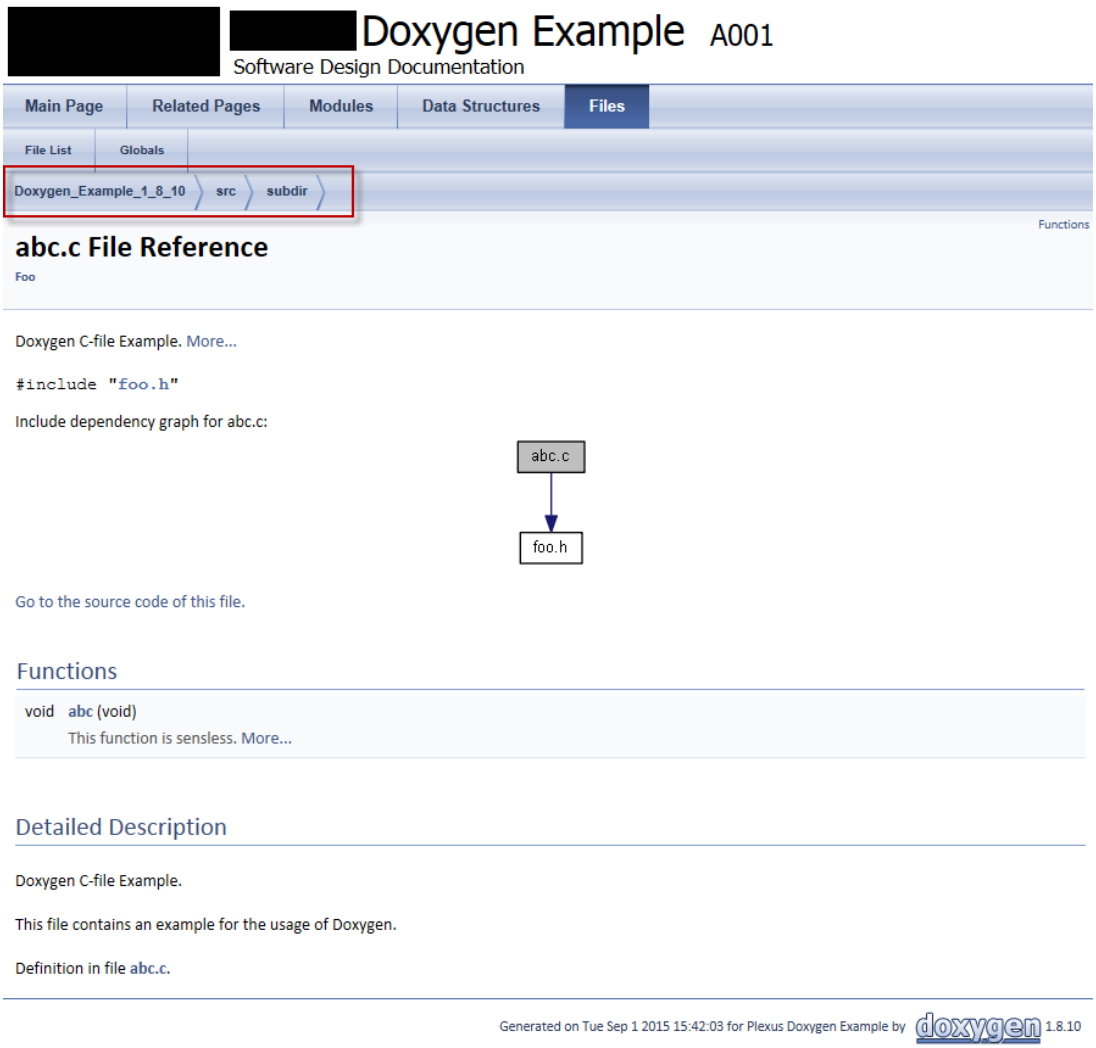


Here we need to choose the output format. We usually check the HTML checkbox, and we can decide if we want to show a navigation panel or a search option. Then, we move to next step.

Last two steps consist in:

- Choosing if you want to generate class diagrams with the internal tool (or with a external tool called GraphViz)
- Run Doxygen

After running *Doxygen*, you will see the progress in the log text area, until a *Doxygen has finished* final message. Then, you can show the HTML output from Doxygen, or navigate to the chosen output folder.



3.6.5. Other tools

There are some other tools that help us generate documentation.

- **Atlassian Confluence**: Atlassian's Jira is one of the most used software products for agile development. The company also has a really interesting module called *Confluence*, which lets us document the *how-to* and technical references, so that we can associate a Jira ticket with code documentation and a Git branch

or Subversion. This way, we can have a more complete overview of the product that help us in its management.

- **Read the Docs**: Free platform in the cloud to manage the documentation of a project.
- **SWAGGER**: Another tool to document the whole development process.

3.6.6. More information

Oracle Javadoc

Documenting with Javadoc

Doxygen