Software documentation

Documentation in Java projects

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.

1. 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

Regarding software products, the most usual types of documentations are:

- User manuals, in order to teach the final user how to use the application
- Software process, a document containing every diagram and report from the analysis and design stages for the application, so that we can see how this application has been conceived
- Reference APIs (*Application Programming Interface*), a complete reference with all the components included in the application, specially classes and their public methods and elements. This last type of documentation is particularly useful for the development team, in order to ease future maintenances or product updates.

In order to create a user manual, we can just use any text editor available, and write a complete document including how to use each component of the application. Alternatively, we can also create an online user manual, so that users can connect to the web site and look for the contents they are interested in.

We are not going to cover how to write these user manuals or analysis/design documentation in this unit. We are going to focus on how to define the reference API of a Java project.

2. 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 a tool 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. 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.

2.1. Required data

The required information to document a class is:

- The class name, including a general description of the class and, optionally, version number and author(s). We can also specify version number and from which version of the program is this class available.
- 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.

2.2. 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;

/**
 * Class to define people information
 * @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 see 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 methods, or `@since` to establish the version from which this class is available.

```
/**
 * Class to define people information
 * @author mariaconsuelorubiosanchez
 * @version 2.1
 * @see MyOtherClass
 * @since 1.5
 */
```

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
- Also some of previous annotations, such as `@see`, `@version` or `@since`, to specify these values for the concrete method we are documenting.

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
*/
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
* @param currentDate Current date to calculate the age
* @return An integer with the age in years of this person currently
*/
public int calculateAge(LocalDate currentDate)
{
    ...
}
```

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
 */
```

> **Exercise 1:**
>
> Document the classes of the *Animals* project (Exercise 1 of this document)

> **Exercise 2:**
>
> Document the classes of the *CulturalOrganization* (Exercise 2 of this document).