What

Creating your own Java documentation

What is JavaDoc?

It is a tool which automatically creates standard documentation in HTML format from the Java source code.

Steps to create JavaDoc:

→ Open your Java program in IntelliJ Idea

→ Click on Tools & then select Generate JavaDoc

→ A pop window will open. Select the project & the packages for which you want to create the JavaDoc.

→ Select the classes for which you want to generate documentation. By default Documentation would be created for all ~~class~~ classes.

→ Select location where you want to save your Javadoc by clicking on the o/p directory.

→ Click OK & your documentation would be saved at specified location

List of JavaDoc Tags

Tag	Syntax	Description
@author	@author name-text	Describes the author of a class
@version	@version version-number	Adds a "Version" heading which specifies current version of release file
@since	@since release-date	Adds a "Since" heading that tells about the release date
@return	description	Adds a "Return" descrip that tells about return value
@param	@param param	
@deprecated	@deprecated deprecatedtext	Adds a "Deprecated" heading indicating that this API should no longer be used

Syntax of Comments in Java Doc

/* Comments */

Annotations in Java

→ Annotations provide metadata to class/methods.
→ Annotations start with '@'.
→ Annotations are helpful for detecting errors. E.g. @override annotations

will make sure that there are no ~~typ~~ types while overriding a method.

Important Annotations in Java:

⇒ @Override
- This annotation makes sure that the sub-class method is successfully overriding the parent class method.
- It can check typing errors with this
- It extracts a warning from the compiler while encountering such situations.

⇒ @Deprecated
- Used for deprecated method.
- Compiler generates warning on use of this method
- High chance of removal in future

⇒ @SuppressWarnings
- Used to suppress warnings generated by compiler

or

⇒ @FunctionalInterface
- An interface which contains only one abstract method is known as functional interface
- It helps us to make sure that F.I.s not having more than one ~~annotation~~ abstract method

E.g code of @Suppress Warning.

```
class KeyPhone {
    @Deprecated
    void sendMessage() {
        sout("Text message sent");
    }
}

class AndroidPhone extends KeyPhone {
    @Override
    void sendMessage() {
        sout("Message sent via Whatsapp");
    }
}

public class CWHS
    p s v m ( ) {
    @Suppress Warnings("deprecation")

    AndroidPhone Samsung = new AndroidPhone();

    Samsung.sendMessage();
    }
}
```

B. Build O/P

This time no warning is generated because we've suppressed the deprecation warning.

E.g. @FunctionInterface
@FunctionalInterface

```
Interface myFunctionalInterface {

    void method1();
    void method2();

}

public class CWH {
        p s v m ( ) {
        out("FunctionalInterface Annotation");

    }
}
```

O/P:
java: Unexpected @FunctionalInterface annotation
myFunctionalInterface is not a functional Interface
multiple non-overriding abstract methods found in interface myFunctionalInterface

→ Note: The above code generates an error ∵ myFunctionalInterface is containing more than one abstract method.