

# javadoc comments in Java

eazy  
bytes

**Javadoc** comments are a special type of comment used in Java programming language to document code. Javadoc comments start with **/\*\* (two asterisks) and end with \*/ (an asterisk and a slash)**. The JDK provides a **javadoc** command-line tool to extract the documentation comments from the source code and generate documentation in HTML format.

Although a documentation comment can appear anywhere in Java source code, the javadoc tool uses only those documentation comments to generate the HTML pages that appear just before the declaration of classes, nested classes, interfaces, enums, annotations, constructors, methods, and fields.

```
/**
 * Takes two {@code int} numbers as input
 * and add them.
 * <p>
 * <b> For example, 2 + 3 =5 </b>
 * @param num1 Represent first number
 * @param num2 Represent second number
 * @return sum value of first and second number
 */
public int sum (int num1, int num2) {
    return num1 + num2;
}
```

In this example, the Javadoc comment starts with a brief description of what the method does. The @param tag is used to document each parameter, and the @return tag is used to document the return value. HTML elements like <p>, <b> </b>, <i> </i> etc. can also be used in Javadoc comments.

Javadoc comments are important for documenting code and making it easier to understand and maintain. They also allow other developers to use your code more easily by providing documentation that explains how to use it.

## Java Doc comment tags

eazy  
bytes

Below are the most commonly used tags in Javadoc comments.

Tag	Description	Applicable to
@since	Conveys under which version the item was added	Variable
@version	Indicates Version number	Class
@author	Author name	Class
@see	Related class name	Class, method, variable
@link	Related URL	Class, method, variable
@code	Source code content	Class, method, variable
@param	Method param name & description	Method
@deprecated	Marks an item obsolete	Class, method, variable
@return	Details of the return value	Method
@exception	Details of the exception	Method

If we want to share the Java documentation in HTML format to others we can use Javadoc tool.  
Project > Generate Javadoc -- Once we click Java Doc corresponding to that will be generated and same can be shared across.  
We can go to the path where the JavaDocumentation is generated as HTML and when we get the path and paste in browser we can see how the Java Documentation for our code looks.

```
@since --- Tells the readers from which version class is available

@author --- we can provide Author Name

package com.javadoc;
/**
 * This is Sample class to show Demo about
 * Java Documentation
 *
 * <p>
 * This is Example Paragraph inside the <b>Java Documentation </b>
 * </p>
 * @see Integer
 * @author Chinmay
 * @since 1.0
 *
 */
public class JavaDocExample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}

/**
 * This method takes two {@code int} numbers as input
 * and returns the sum
 *
 * <p>
 * <b>2+3=5</b>
 * </p>
 * @param num1 represents First number
 * @param num2 represents Second Number
 * @return Sum of First and Second Number
 *
 * @exception InputMismatchException If the input does not match
 */
public static int add(int num1, int num2)
{
    return num1+num2;
}
```

## ANNOTATIONS AND JAVA DOC

- There is a tool called java doc that will help to prepare the documentation for the classes.
- Java provides some tags for java documentation those are known as java doc tool.
- The tags for class or a package are:
  - @author: Adds the author name of the class.
  - @version: Adds a version subheading with specified version text to generated docs

- **@since:** To mention when was the version written or how long it may be valid.
  - **@see:** Adds a see also heading with a link for example to see the references to the given link.
- The tags for methods are:
- **@param:** To mention the parameters taken by particular product.
  - **@return:** To mention the value returned by the method.
  - **@throws/@exception:** To know the exception thrown by the method.
  - **@deprecated:** To mention whether the method is deprecated or not that is the method may not be used longer.
  - **@code:** Displays text in code font without interpreting the text as HTML mark-up or nested java doc tags.
- The other tags available are:
- **@link:** To provide the link for particular resource.
  - **@value:** To provide value for any static variable or a member.
  - **@serial:** To mention the serial id for serialization.

**Annotation:** Annotations are used for giving attributes or defining the attributes for a class or a interface or methods.

➤ Annotations are useful for giving meta data to class or interface or a method.

➤ Built-in annotations can be categorised into two:

→ Applied to code:

These are set of annotations applied upon the code so, this type of annotation gives the hint to the compiler so that it avoids showing errors and warnings.

The in-built annotations applied to the code are:

- **@Override:**  
It informs the compiler that the element is meant to override an element declared in a superclass.
- **@deprecated:**  
It indicates that the marked element is deprecated and should no longer be used.
- **@FunctionalInterface:**  
Indicates that the type declaration is intended to be a functional interface
- **@SuppressWarnings:**  
It tells the compiler to suppress specific warnings that it would otherwise generate.
- **@SafeVarArgs:**  
When it is applied to a method or a constructor, it asserts that the code does not perform potentially unsafe operations on its varargs parameter.

→ Applied to other annotations:

These are set of annotations applied upon user-defined annotation.

The different user-defined annotations are:

- **@Retention:**  
It specifies how the marked annotations are stored.
- **@Documented:**  
It indicates that whenever the specified annotation is used those elements should be documented using java doc tool
- **@Target:**  
It marks another annotation to restrict what kind of java element the java elements can be applied to.
- **@Inherited:**  
It indicates that the annotation type can be inherited from super class.
- **@Repeatable:**  
It indicates that the marked annotation can be applied more than once to the same declaration or type used.

➤ **Meta data**

**Metadata** is "data that provides information about other data" in

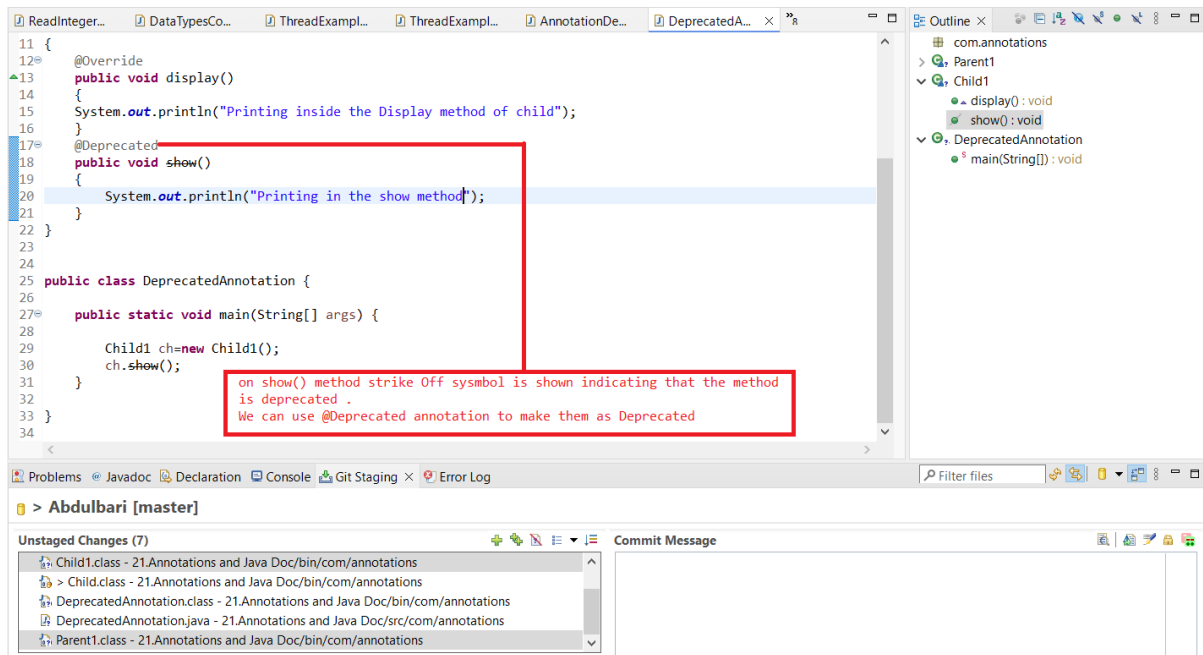
```
1 package com.annotations;
2
3 class Parent
4 {
5     public void display()
6     {
7     }
8 }
9
10 class Child extends Parent
11 {
12     public void displa()
13     {
14     }
15 }
16
17
18 }
```

Here we are trying to override display method but due to some typo issue we have named it as displa(). Compiler will think it as new method. But our designed scenario is to Override the Parent display() method in Child.

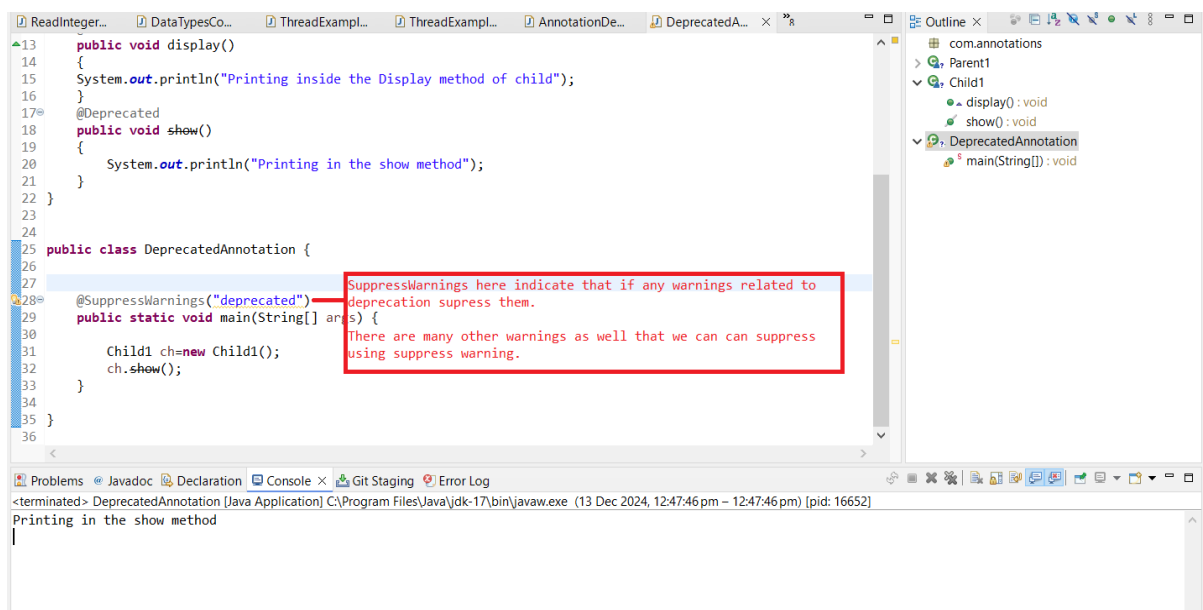
To Overcome this keep @Override on top of that method and then it will throw an error if it didn't override correctly.

```
3 class Parent
4 {
5     public void display()
6     {
7     }
8 }
9
10 class Child extends Parent
11 {
12     @Override
13     public void display()
14     {
15     }
16 }
17 }
```

When ever we are overriding it is good practice to use @Override Annotation.



## @Deprecated Annotation



## @Supress Warnings

```

1 package com.annotations;
2
3 class My<T>
4 {
5     @SafeVarargs    Will be used for variable arguments and that method should be private or final
6     private void show(T...arg)
7     {
8         for(T x:arg)    We can use @Supresswarnings as well to supress those
9             warnings
10            {
11                System.out.println(x);
12            }
13    }
14 }
15 public class SafeVar {
16
17     public static void main(String[] args) {
18
19     }
20 }
21
22 }
23

```

Problems Javadoc Declaration Console × Git Staging Error Log

@Safevargs

## 208.User-Defined Annotation

```

1 package com.annotations; whenever you define annotation that will be automatically inheriting from annotation class that is inside
2
3 @interface MyAnno annotation package which is inside java.lang package.
4 {
5     we can give annotation at class or method level or parameter level or a local variable level Or an instance
6 } variable.
7
8 @MyAnno Annotation definition will be similar to interface but it will be starting with @.
9 public class UserDefinedAnnotation {
10
11     @MyAnno
12     int val;
13     @MyAnno
14     public static void main(@MyAnno String[] args) {
15
16         @MyAnno
17         int a=10; //User Defined Annotation
18     } //@interfae MyAnno
19     {
20 }
21 }

```



User-Defined  
Annotations.txt

209: Built in Annotations:

