**Brief introduction**

Java Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in Java are used to provide additional information, so it is an alternative option for XML and Java marker interfaces.

[**General purpose annotations**](https://www.baeldung.com/java-default-annotations)

**@Override**

It is a marker annotation that can be used only on methods. A method annotated with **@Override** must override a method from a superclass. It is used to ensure that a superclass method is actually overridden, and not simply overloaded.

**What is @Target ?**

Here you can notice one more annotation, we will discuss them a bit later. Shortly, @Target indicates member of class which we can use @Override with. In our case it is method.

**What is @Retention ?**

@Retention(RetentionPolicy.SOURCE) indicates, that @Override annotation will be used at compile-time.

**@Deprecated**

The @Deprecated annotation can be used to mark an API as not intended for use anymore.

[Additionally, Java 9 has added some optional attributes](https://www.baeldung.com/java-deprecated)

[**@SuppressWarnings**](https://www.baeldung.com/java-suppresswarnings#suppresswarningsannotation)

@SuppressWarnings indicates we want to ignore certain warnings from a part of the code.

Although compiler warnings are useful because they can tell us that we have code with bad smell, sometimes we may want to violate these rules by purpose in case of production needs.

And in case if we have CI/CD pipeline with such tool as SonarQube (which controls code quality), we will not pass code quality gates. In this case we may either configure code analysis tool appropriately or use **@SuppressWarnings.**

[**@SafeVarArgs**](https://www.baeldung.com/java-safevarargs)

**@SafeVarargs** is like **@SupressWarnings** in that it allows us to declare that a particular compiler warning is a false positive. Once we ensure our actions are safe, we can add this annotation.

<https://docs.oracle.com/javase/specs/jls/se7/html/jls-9.html#jls-9.6.3.7>

**@FunctionalInterface**

<https://www.baeldung.com/java-default-annotations>

Since JDK 8, Java which was originally pure OOP language has got some features of functional programming.

Functional interfaces are interfaces that ensure that they include precisely only one abstract method.

Also, functional interface can have one or more default methods implementation.

In most cases we are not using this annotation explicitly. For example, I’ve never met with production situation where I have to create **@FunctionalInterface**, because there are a lot of ready implementations in **java.util.function** package.

Runnable is one of the most obvious examples of @FunctionalInterface

[FunctionalInterface can be implemented by a lambda expression.](https://jenkov.com/tutorials/java-functional-programming/functional-interfaces.html)