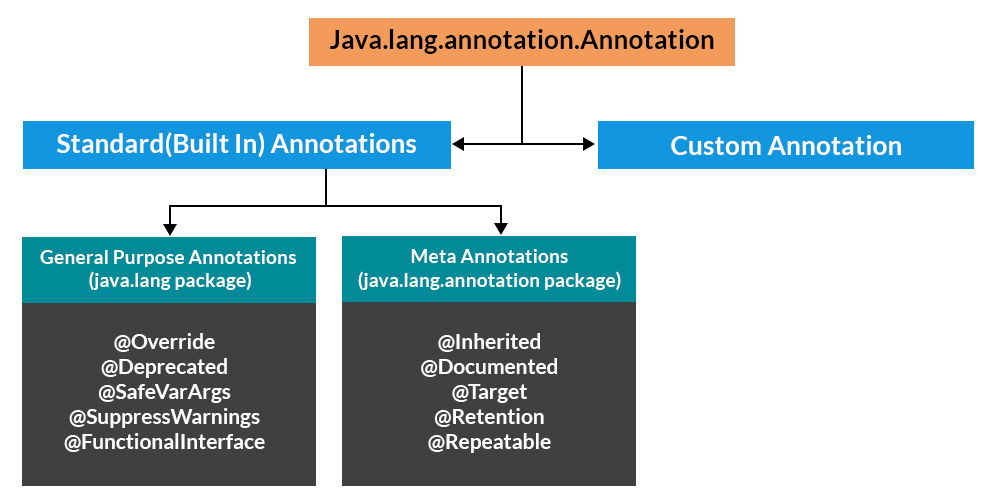
Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

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

**Benefits**

* Java annotations are a mechanism for adding metadata information to our source code (Program).
* They are a powerful part of Java that was added to JDK5.
* Annotations provide an alternative to the use of XML descriptors.
* Also, we are able to attach them to packages, classes, interfaces, methods, and fields, annotations by themselves have no effect on the execution of a source code (Program).

## Hierarchy ****of Annotations in Java****



Built-In Java Annotations used in Java code

@Override

@SuppressWarnings

@Deprecated

Built-In Java Annotations used in other annotations

@Target

@Retention

@Inherited

@Documented

**@Override**

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

**Refer to the Program TestAnnotation1.java**

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

Use of @SuppressWarnings is to suppress or ignore warnings coming from the compiler, i.e., the compiler will ignore warnings if any for that piece of code.

1. @SuppressWarnings("unchecked")

public class Calculator {

}

- Here, it will ignore all unchecked warnings coming from that class. (All methods, variables, constructors).

2. public class Calculator {

@SuppressWarnings("unchecked")

public int sum(x,y) {

.

}

}

- It will stop warning from that function only, and not from other functions of Calculator class.

Possible Values Inside @SuppressWarnings Annotation Element  are as follows:

| Values | **Description** |
| --- | --- |
| All | It will suppress all warnings. |
| Cast | Suppress the warning while casting from a generic type to a nonqualified type or the other way around. |
| Deprecation | Ignores when we’re using a deprecated(no longer important) method or type. |
| divzero | Suppresses division by zero warning. |
| empty | Ignores warning of a statement with an empty body. |
| unchecked | It doesn’t check if the data type is Object or primitive. |
| fallthrough | Ignores fall-through on switch statements usually (if “break” is missing). |
| hiding | It suppresses warnings relative to locals that hide variable |
| serial | It makes the compiler shut up about a missing serialVersionUID. |
| finally | Avoids warnings relative to finally block that doesn’t return. |
| unused | To suppress warnings relative to unused code. |

**Note:** The primary and most important benefit of using @SuppressWarnings Annotation is that if we stuck because of some known warning, then this will ignore the warning and move ahead. E.g. – d*eprecated* and *unchecked* warnings.

**Refer to the Program TestAnnotation2.java**

@Deprecated

@Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

**Refer to the TestA.java**

**Refer to the URL for few Deprecated methods in java**

[**https://www.iitk.ac.in/esc101/05Aug/tutorial/post1.0/converting/deprecated.html**](https://www.iitk.ac.in/esc101/05Aug/tutorial/post1.0/converting/deprecated.html)

**Java Custom annotations** or Java User-defined annotations are easy to create and use. The *@interface* element is used to declare an annotation. For example:

1. @interface MyAnnotation{}

Here, MyAnnotation is the custom annotation name.

**Points to remember for java custom annotation signature**

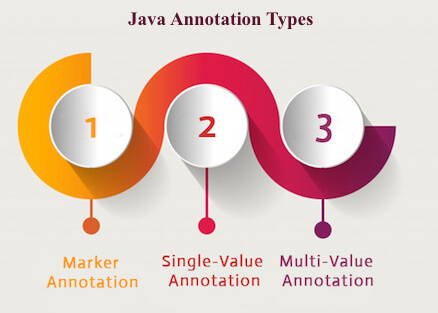
There are few points that should be remembered by the programmer.

1. Method should not have any throws clauses
2. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
3. Method should not have any parameter.
4. We should attach @ just before interface keyword to define annotation.
5. It may assign a default value to the method.

**Types of Annotation**

There are three types of annotations.

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation



**1) Marker Annotation**

An annotation that has no method, is called marker annotation. For example:

1. @interface MyAnnotation{}

The @Override and @Deprecated are marker annotations.

**2) Single-Value Annotation**

An annotation that has one method, is called single-value annotation. For example:

1. @interface MyAnnotation{
2. int value();
3. }

We can provide the default value also. For example:

1. @interface MyAnnotation{
2. int value() default 0;
3. }

**How to apply Single-Value Annotation**

Let's see the code to apply the single value annotation.

1. @MyAnnotation(value=10)

The value can be anything.

**3) Multi-Value Annotation**

An annotation that has more than one method, is called Multi-Value annotation. For example:

1. @interface MyAnnotation{
2. int value1();
3. String value2();
4. String value3();
5. }
6. }

We can provide the default value also. For example:

1. @interface MyAnnotation{
2. int value1() default 1;
3. String value2() default "";
4. String value3() default "ram";
5. }

**How to apply Multi-Value Annotation**

Let's see the code to apply the multi-value annotation.

1. @MyAnnotation(value1=10,value2="Arun Kumar",value3="Hyd")

**Built-in Annotations used in custom annotations in java**

* @Target
* @Retention
* @Inherited
* @Documented

**@Target**

**@Target** tag is used to specify at which type, the annotation is used.

The java.lang.annotation.**ElementType** enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc.

Let's see the constants of ElementType enum:

|  |  |
| --- | --- |
| **Element Types** | **Where the annotation can be applied** |
| TYPE | class, interface or enumeration |
| FIELD | fields |
| METHOD | methods |
| CONSTRUCTOR | constructors |
| LOCAL\_VARIABLE | local variables |
| ANNOTATION\_TYPE | annotation type |
| PARAMETER | parameter |

**Example to specify annoation for a class**

1. @Target(ElementType.TYPE)
2. @interface MyAnnotation{
3. int value1();
4. String value2();
5. }

**Example to specify annotation for a class, methods or fields**

1. @Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
2. @interface MyAnnotation{
3. int value1();
4. String value2();
5. }

**@Retention**

**@Retention** annotation is used to specify to what level annotation will be available.

|  |  |
| --- | --- |
| **RetentionPolicy** | **Availability** |
| RetentionPolicy.SOURCE | refers to the source code, discarded during compilation. It will not be available in the compiled class. |
| RetentionPolicy.CLASS | refers to the .class file, available to java compiler but not to JVM . It is included in the class file. |
| RetentionPolicy.RUNTIME | refers to the runtime, available to java compiler and JVM . |

**Example to specify the RetentionPolicy**

1. @Retention(RetentionPolicy.RUNTIME)
2. @Target(ElementType.TYPE)
3. @interface MyAnnotation{
4. int value1();
5. String value2();
6. }

**Example of custom annotation: creating, applying and accessing annotation**

**Refer to File: Test.java**

**How built-in annotations are used in real scenario?**

In real scenario, java programmer only need to apply annotation. He/She doesn't need to create and access annotation. Creating and Accessing annotation is performed by the implementation provider. On behalf of the annotation, java compiler or JVM performs some additional operations.

**@Inherited**

By default, annotations are not inherited to subclasses. The @Inherited annotation marks the annotation to be inherited to subclasses.

1. @Inherited
2. @interface ForEveryone { }//Now it will be available to subclass also
4. @interface ForEveryone { }
5. class Superclass{}
7. class Subclass extends Superclass{}

**@Documented**

The @Documented Marks the annotation for inclusion in the documentation.

Annotations may include elements that are just name-value pairs separated by commas. Allowed types are primitives, strings, enums and arrays of them:

    @Author(name = "Albert",

           created = "17/09/2010",

           revision = 3,

           reviewers = {"George", "Fred"})

   public class SimpleAnnotationsTest {…}

**Refer to Test1.java**