**Java Annotations**

Java Annotations are new features added from JAVA 1.5 Onwards.  
  
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**. By using Annotations we can provide more information about the class or a method or a constructor etc.   
  
Annotations in java are used to provide additional information, so it is an alternative option for XML and java marker interfaces.  
Annotation file also ends with an extension called as “**.java**”.

## Custom Annotation – User defined annotations :

**Java Custom annotations** or Java User-defined annotations are easy to create and use.

The *@interface* element is used to declare an annotation.

**Example:**

@interface MyAnnotation{}

Here, MyAnnotation is the custom annotation name.

**Example:**

## MyAnnotations.java:

## *public @interface MyAnnotation{*

## *}*

**Compile:**

*No issues.*

**Javap MyAnnotation:**

The following is the internal code of above:



It is an interface which extends java.lang.Annotation.

**Example :**

## MyAnnotations.java:

## *public @interface MyAnnotation{*

## *}*

**COne.java:**

***@MyAnnotation***

*public class COne{*

*}*

**Compile:**

*No issues.*

**Example:**

## MyAnnotations.java:

## *public @interface MyAnnotation{*

## *}*

**Cone.java:**

***@MyAnnotation()***

*public class COne{*

*}*

**Compile:**

*No issues.*

**Note: *@MyAnnotation() and @MyAnnotation are same, if there are no mandatory elements.***

**Example :**

## MyAnnotations.java:

*@MyAnnotation()*

*public class COne{*

*@MyAnnotation()*

*public void methodOne(){*

*}*

*}*

**Compile:**

*Invalid type for Annotation Member*

**NOTE :**

**We should not return “void” in the method return types.**

**Example :**

## MyAnnotations.java:

*@MyAnnotation()*

*public class COne{*

*@MyAnnotation()*

*public String methodOne(){*

*}*

*}*

**Compile:**

*No Issues.*

**Example :**

## MyAnnotations.java:

*@MyAnnotation()*

*public class COne{*

*@MyAnnotation()*

*public int[] methodOne(){*

*}*

*}*

**Compile:**

*No Issues.*

**Example:**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*}*

**Cone.java:**

*@MyAnnotation()*

*public class COne{*

*@MyAnnotation*

*private int roll;*

*@MyAnnotation()*

*public String m1(){*

*return"xx";*

*}*

*public void m2(@MyAnnotation int x){*

*}*

*}*

**Compile:**

Annotation MyAnnotation is missing methodOne

**Note:**

**If we declare a method and if we don’t use that method in the class which is preceded with @Annotation\_Name, then we will get error.**

**Because by default, all methods in the Annotation are mandated.**

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String name();*

*}*

**Cone.java:**

*@MyAnnotation(name="test")*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String name();*

*}*

**Cone.java:**

*@MyAnnotation(name="test")*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String[] name();*

*}*

**Cone.java:**

*@MyAnnotation(name={"test1","test2","test3"})*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String[] name();*

*}*

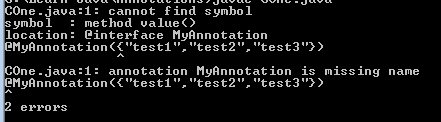
**Cone.java:**

*@MyAnnotation({"test1","test2","test3"})*

*public class COne{*

*}*

**Compile:**



**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String[] name();*

*}*

**Cone.java:**

*@MyAnnotation(name={"test1"})*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String[] name();*

*}*

**Cone.java:**

*@MyAnnotation(name="test1")*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String fname();*

*public String lname();*

*public String collegename();*

*}*

**Cone.java:**

*@MyAnnotation(fname="test1",lname="test2",collegename="test3")*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String fname();*

*public String lname();*

*public String collegename();*

*}*

**Cone.java:**

*@MyAnnotation(lname="test2",fname="test1",collegename="test3")*

*public class COne{*

*}*

**Compile:**

No Issues.

**Example :**

## MyAnnotations.java:

*public @interface MyAnnotation{*

*public String fname();*

*public String[] lname();*

*public int rollno();*

*}*

**Cone.java:**

*@MyAnnotation(lname={"test2","test3","test4"},fname="test1",rollno=20)*

*public class COne{*

*}*

**Compile:**

No Issues.

## 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.

@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** "xyz";
5. }

## How to apply Multi-Value Annotation

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

@MyAnnotation(value1=10,value2="Arun Kumar",value3="Ghaziabad")

## Built-In Java Annotations

There are several built-in annotations in java. Some annotations are applied to java code and some to other annotations.

## Built-In Java Annotations used in java code

* @Override
* @SuppressWarnings
* @Deprecated

## Built-In Java Annotations used in other annotations

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

## Understanding Built-In Annotations in java

Let's understand the built-in annotations first.

## @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.

class Animal{

void eatSomething(){

System.out.println("eating something");}

}

class Dog extends Animal{

@Override

void eatsomething(){

System.out.println("eating foods");

}//should be eatSomething

}

class TestAnnotation1{

public static void main(String args[]){

Animal a=new Dog();

a.eatSomething();

}

}

Output: Comple Time Error

## @SuppressWarnings

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

1. import java.util.\*;
2. class TestAnnotation2{
3. @SuppressWarnings("unchecked")
4. public static void main(String args[]){
5. ArrayList list=new ArrayList();
6. list.add("sonoo");
7. list.add("vimal");
8. list.add("ratan");
10. for(Object obj:list)
11. System.out.println(obj);
13. }}

Now no warning at compile time.

If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

## @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.

1. class A{
2. void m(){System.out.println("hello m");}
4. @Deprecated
5. void n(){System.out.println("hello n");}
6. }
8. class TestAnnotation3{
9. public static void main(String args[]){
11. A a=new A();
12. a.n();
13. }}

## At Compile Time:

Note: Test.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

## At Runtime:

hello n

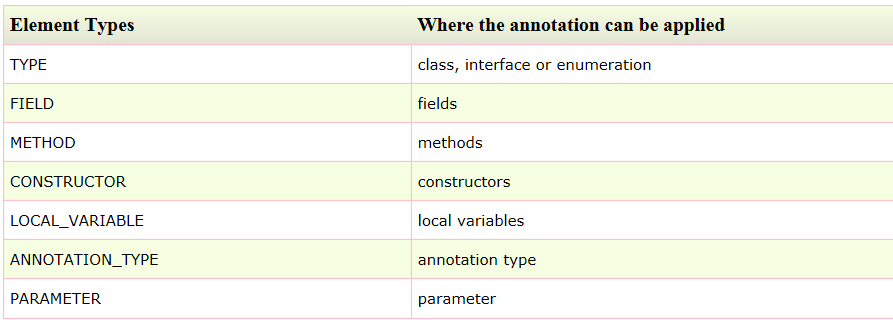
## Built-in Annotations used in custom annotations in java

* @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:



**Example to specify annotation for a class**

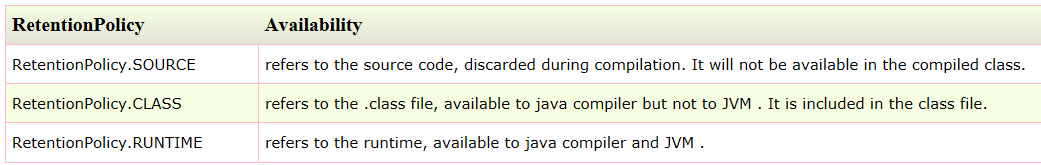
1. @Target(ElementType.TYPE)
2. **@interface** MyAnnotation{
3. **int** value1();
4. String value2();
5. }

**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.



**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

Let's see the simple example of creating, applying and accessing annotation.

File: Test.java

1. //Creating annotation
2. **import** java.lang.annotation.\*;
3. **import** java.lang.reflect.\*;
5. @Retention(RetentionPolicy.RUNTIME)
6. @Target(ElementType.METHOD)
7. **@interface** MyAnnotation{
8. **int** value();
9. }
11. //Applying annotation
12. **class** Hello{
13. @MyAnnotation(value=10)
14. **public** **void** sayHello(){System.out.println("hello annotation");}
15. }
17. //Accessing annotation
18. **class** TestCustomAnnotation1{
19. **public** **static** **void** main(String args[])**throws** Exception{
21. Hello h=**new** Hello();
22. Method m=h.getClass().getMethod("sayHello");
24. MyAnnotation manno=m.getAnnotation(MyAnnotation.**class**);
25. System.out.println("value is: "+manno.value());
26. }}

Output:value is: 10

## How built-in annotaions 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.

Why don’t you support annotation subtyping (where one annotation type extends another)?

It complicates the annotation type system, and makes it much more difficult to write “Specific Tools”.

…

“Specific Tools” — Programs that query known annotation types of arbitrary external programs. Stub generators, for example, fall into this category. These programs will read annotated classes without loading them into the virtual machine, but will load annotation interfaces.