# Annotations

## 1. Introduction

➔ Annotations also called as metadata. But term Annotations is more descriptive and most commonly used.

➔ Annotations enables one to embed supplement information into a source file.

➔ Annotations doesn't changed the action of a program but can be used in various way both for development and deployment.

## 2. Annotations Basics

➔ An annotation is created on the mechanism based on the interface. See the example given below

```
// A simple annotation type.
@interface MyAnno {
String str();
int val();
}
```

➔ Above we can see that @ symbol precedes keyword interface, which tells compiler that an annotation type has been declared.

➔ Also we can see that two members str() and val() are present. All annotations consists solely of methods declarations however, one don't provide bodies for these methods. Instead Java implements these methods.

➔ An annotation doesn't include extends clause however all annotations extends 'Annotation' interface i.e. Annotation is super interface of all annotations. It is declared within **java.lang.annotation** package. It overrides hashCode(), equals() and toString() method of the Object class.

➔ Once annotation has been declared then it can be used to annotate any declaration like classes, methods, fields, parameters, enums etc. Even an annotation can be annotated.

➔ When one annotates something, he will gives values to its members. See the example below

```
// Annotate a method.
@MyAnno(str = "Annotation Example", val = 100)
public static void myMeth() { // ...
```

While assigning values to a member only its name is used(without parenthesis). Above we can see String member assigned with String value and int member with Integer value. Here we can see that members which are declared as methods are actually used as fields.

## 3. Annotation's Retention policy

➔ A retention policy determines at what point an annotation is discarded.

➔ Java defines three such policies which are encapsulated in **java.lang.annotation.RetentionPolicy** enumeration. They are **SOURCE**, **CLASS** and **RUNTIME**.

➔ An annotation with retention policy of SOURCE will retained only in source file and discarded during compilation. Annotation with retention policy of CLASS will be stored in .class file during compilation but not available through the JVM during run-time. An annotation with retention policy of RUNTIME will be stored in .class file during compilation and available through the JVM during run-time.

➔ Example for how retention policy is defined for an annotation

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
String str();
int val();
}
```

**4. Obtaining annotation at run-time using Reflection**

➔ If annotation retention policy is defined as RUNTIME then they can be queried at run-time by any java program using reflection.

➔ Reflection is the feature that enables information about the class during run-time. Reflection API are present in the **java.lang.reflect** package.

➔ To use reflection API one should first get 'Class' object that represents class whose annotation one wants to obtain. 'Class' is one of java's built in classes present in **java.lang** package. There are various ways to get Class object but easiest one is to call **getClass()** method present in class Object.

➔ Once one will get the 'Class' object then one can get information associated with it including annotations. To get annotation information associated with method, field, parameter, constructor etc first one should get those objects using methods like **getMethod()**, **getField()**, **getParameter()** and **getConstructor()** etc. These methods returns object of type **METHOD**, **FIELD**, **PARAMTER** and **CONSTRUCTOR** respectively.

➔ One can then get specific annotations associated with such object using **getAnnotation()** method on these objects. General form of it is shown below

```
Annotation getAnnotation(Class annoType)
```

Here annoType is the Class object that represents annotation which one wanted to obtain. This method returns reference to the annotation and by using reference one can obtain values associated with annotation's members. The method returns null if annotation is not found or if annotation dont have RUNTIME retention.

➔ Check the example below. We can see method1() don't have any argument. But method2() have two arguments string and int. Hence second parameter for getMethod() is a varargs which is 'Class' array with members are nothing but method arguments which one is looking for. The members of 'Class' array will be 'Class' objects i.e. Classes assigned to method arguments. Also note for int the Class will be wrapper class 'Integer.TYPE'. Where TYPE is the Class instance representing the primitive type int. OR one can just used int.class as shown.

➔ One can get all annotations that has RUNTIME retention policy, annotated to a particular item(like Class or Method) by calling getAnnotations() method as shown in the example. Its general form is

```
Annotation[] getAnnotations().
```

**5. AnnotatedElement Interface**

➔ Methods like getAnnotation() and getAnnotations() etc are defined in 'AnnotatedElement' interface.

➔ AnnotatedElement interface is defined in java.lang.reflect package.

➔ This interface supports reflection for annotations and implemented(indirectly) by class Constructor, Method, Field, Class etc.

➔ It defines other methods too like isAnnotationPresent(Class annoType) which returns boolean true or false for annotation

is associated with particular object or not.

### 6. Default Values for Annotations

➔ One can give default values to annotation members which will be used when no values are supplied.

➔ It has a general form

*type member()* default *value*;

where value should be of type 'type'

➔ Example

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno
{
    String str() default "Testing";
    int value() default 100;
}
```

➔ Following are four ways that annotation @MyAnno wiill work

```
@MyAnno // both str and value will be default
@MyAnno(str="String1") //values will be default
@MyAnno(value=100) //str will be default
@MyAnno(str="String2" value=200) // No defaults
```

➔ Please refer the 'AnnotationCls' below.

### 7. Marker Interface

➔ Marker interface is a interface which dont have any members. It is simply used as example @MyMarker without any parentheses. Example

```
@interface MyMarker
{
}
```

and it will be used as

```
@MyMarker
class Marker
{ //...}
```

### 8. Single Member Annotations

➔ A single member annotation as name suggests will have only one member.

➔ One option comes with this is it allows shorthand form of specifying value.

➔ Ex

```
@interface SingleMem
{
int value();
}
```

It could be used as

```
@SingleMem(100)
public void method()
{
}
```

➔ One can use this shorthand for other annotations too who have more than one member but only if other members have default values.

➔ Ex

```
@interface Meta
{
int value();
String str() default "default";
}
```

Then one can use

```
@Meta(10) // str value will be "default"
public void method1()
{
//method body
}
```

**9. Built-in Annotations**

➔ Java defines many built in annotations. Most are specifialized, but seven are general purpose. They are

**1) @Retention**

It is designed to be used as annotation to another annotations. It is used to specify retention policy of as seen before.

**2) @Documented**

Its a market annotation which is designed to use as annotation to other annotation declarations. It tells that an annotated annotation needs to be documented.

**3) @Target**

It is also designed to used as annotation to another annotations. It has only one argument defined as constant from enumeration **EnumType**. Argument describes the type of declaration to which the annotation can be applied.

| Target Constant | Annotation Can Be Applied To |
|---|---|
| ANNOTATION_TYPE | Another annotation |
| CONSTRUCTOR | Constructor |
| FIELD | Field |
| LOCAL_VARIABLE | Local variable |
| METHOD | Method |
| PACKAGE | Package |
| PARAMETER | Parameter |
| TYPE | Class, interface, or enumeration |

One can use more than one values as shown below

```
@Target( { ElementType.FIELD, ElementType.LOCAL_VARIABLE } )
```

**4) @Inherited**

➔ Its a marker annotation which can only be applied to other annotations.

➔ It causes annotations from super class to be inherited by sub class. That is, when request for a specific annotation is made in a sub class and if annotation is not present then its superclass annotations are checked. If annotation is present and if it was annotated with @Inherited then annotation will be returned.

**5) @Override**

➔ Its a marker interface used only for methods. When a method is annotated with @Override then it must override a super class method. If it don't then then there will be compile time error. It ensures that method is actually overridden and not simply overloaded.

**6) @Deprecated**

➔ Its a marker annotation. It indicates that a declaration is obsolete and has been replaced by a newer one.

**7) @SuppressWarnings**

➔ It specifies one or more warning issued by compiler to be suppressed. The warnings to suppress are specified by name, in string form.

**10. <u>Restrictions for Annotations</u>**

1. One annotation cannot inherit another.
2. All methods declared by annotations must be without parameters.
3. They must return one of the following

• Aprimitive type, such as **int** or **double**

• An object of type **String** or **Class**

• An **enum** type

• Another annotation type

• An array of one of the preceding types

4. Annotations cannot be generic.
5. Annotations cannot specify throws clause.

**11. Annotation Example**

```
package annotations;

import java.lang.annotation.Annotation;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.reflect.Method;
import java.util.logging.Level;
import java.util.logging.Logger;


@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno
{
    String str();
    int value();
}

@Retention(RetentionPolicy.RUNTIME)
@interface Meta
{
    String description();
}

@Retention(RetentionPolicy.RUNTIME)
@interface Default
{
    String str1() default "Default";
    int int1() default 1000;
}

public class AnnotationCls {

    @MyAnno(str="method1", value = 101)
    public void method1()
    {
        System.out.println("Inside method2()");
    }

    @Meta(description="This is method2()")
    @MyAnno(str="method2", value = 201)
    public void method2(String str, int i)
    {
```

```
    System.out.println("Inside method2()");
}

@Default
public void method3()
{
    System.out.println("Inside method3()");
}

    public void method4()
    {
        AnnotationCls avc = new AnnotationCls();
        Class c = avc.getClass();

    try {
        //Get method1() annotations
        Method m1 = c.getMethod("method1");
        MyAnno anno = m1.getAnnotation(MyAnno.class);
        System.out.println("Annotation's values for method1() are : "+anno.str()+"
        "+anno.value());
        System.out.println(anno);
        System.out.println();

      //Get method2() annotations
      //  Class [] classArr = new Class[2];
      //  classArr[0] = String.class;
      //  classArr[1] = int.class;//Integer.TYPE;

        Method m2 = c.getMethod("method2", String.class, int.class);// classArr);
        anno = m2.getAnnotation(MyAnno.class);
        System.out.println("Annotation's values for method2() are : "+anno.str()+"
        "+anno.value());
        System.out.println(anno);
        System.out.println();

        //Get all annotationns
        Annotation annArr [] = m2.getAnnotations();
        System.out.println("all annotations of method2()");
        for(Annotation a : annArr)
            System.out.println(a);
        System.out.println();

        //Annotation with Default values
        System.out.println("Annotation with default values");
        Method m3 = c.getMethod("method3");
```

```
            Default def = m3.getAnnotation(Default.class);
            System.out.println(def);
        } catch (NoSuchMethodException ex) {
            System.out.println("method not found");
        }
    }

    public static void main(String args [])
    {
        AnnotationCls avc = new AnnotationCls();
        avc.method4();
    }
}
Output:
Annotation's values for method1() are : method1 101
@annotations.MyAnno(str=method1, value=101)

Annotation's values for method2() are : method2 201
@annotations.MyAnno(str=method2, value=201)

all annotations of method2()
@annotations.Meta(description=This is method2())
@annotations.MyAnno(str=method2, value=201)

Annotation with default values
@annotations.Default(str1=Default, int1=1000)
```