

# Tutorial: Java Inherited Annotations

Vladimir Ovchinnikov<sup>1</sup>  
© Fusionsoft

The open-source library of inherited annotations is for Java-developers solving the annotation inheritance task. The annotations being inherited are of classes, interfaces, or their methods. It uses the consistent inheritance model: inheritance proceeds only if the same annotation is not present on the same element (class, interface or method) within superclasses or superinterfaces. Annotations can be overridden within descendants. The library is open-source and free, with no restriction for commercial application.

---

<sup>1</sup> Please, report all discovered errors and inaccuracies, suggestions and requests to [info@fusionsoft-online.com](mailto:info@fusionsoft-online.com)

# CONTENTS

Problem to Solve .....	2
Inherited Annotations .....	3
Declaration of Annotations.....	3
Inheritance of Annotations .....	3
Conflict of Annotations .....	4
Overriding Annotations .....	4

## Problem to Solve

The Java standard annotation mechanism does not provide inheritance for annotations. Let's consider the following example:

```
@Target( ElementType.TYPE)
@Retention( RetentionPolicy.RUNTIME)
public @interface A {}

@Target( ElementType.METHOD)
@Retention( RetentionPolicy.RUNTIME)
public @interface B {}

@A
public interface BaseInterface {
    @B
    public void method1();
}

public class BaseClass {
    @B
    public void method2(){}
}

public class Derived extends BaseClass implements BaseInterface{
    public void method1(){}
    public void method2(){}
}
```

This example contains two kinds of annotations: the annotation @A for types, and the annotation @B for methods. The annotations were used for describing one superclass and one superinterface. There was a derived class created which is inherited from the superclass and superinterface.

According to Java annotation practice, neither derived class nor its methods inherit annotations defined in the superclass and superinterface. Therefore the following code returns null in both cases:

```
Derived.class.getMethod(
    "method1", new Class[0]).getAnnotation(B.class)

Derived.class.getMethod(
    "method2", new Class[0]).getAnnotation(B.class)
```

Often applications analyzing annotations requires annotations of classes as well as methods inherit. This library solves the task. To get the inherited annotations in our example, one can do the following:

```
AnnotationManager.getAnnotatedClass(Derived.class)
```

```

        .getAnnotation(A.class);

        AnnotationManager.getAnnotatedClass(Derived.class)
        .getAnnotatedMethod("method1", new Class[0])
        .getAnnotation(B.class);

        AnnotationManager.getAnnotatedClass(Derived.class)
        .getAnnotatedMethod("method2", new Class[0])
        .getAnnotation(B.class);

```

## Inherited Annotations

### Declaration of Annotations

In order to get annotations accessible in run-time, one should give them the system annotation `@Retention(RetentionPolicy.RUNTIME)`. Also the annotation `@Target` can be used to restrict annotation application scope. For instance, let's declare the annotation `@A` being applicable to interfaces or classes (types) only:

```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface A {}

```

### Inheritance of Annotations

`AnnotationManager` class is responsible for annotation inheritance. No annotations are copied actually, the class holds relations between code elements and annotations. In order to get inheritance information for some class or interface, one should call the method `AnnotationManager.getAnnotatedClass`:

```

AnnotationManager.getAnnotatedClass(Derived.class)

```

The method returns an object being an annotated class which contains annotation inheritance information for the specified class.

In order to get class annotations (proper and inherited), one should use one of the following methods: `getAllAnnotations` or `getAnnotation`, for example

```

AnnotationManager.getAnnotatedClass(Derived.class)
        .getAnnotation(A.class);

```

An annotated class contains annotated methods for all methods of the source class. The annotated methods know about inherited annotations for the source methods. To get an annotated method one can use the methods `getAnnotatedMethods` and `getAnnotatedMethod`. For instance, an annotated method can be obtained using its signature as follows:

```

AnnotationManager.getAnnotatedClass(Derived.class)
        .getAnnotatedMethod("method1", new Class[0])

```

The annotated method can be obtained also using the source method itself.

Any annotation of an annotated method (proper and inherited) can be obtained through `getAllAnnotations` or `getAnnotation`, for instance:

```

AnnotationManager.getAnnotatedClass(Derived.class)
        .getAnnotatedMethod("method1", new Class[0])

```

```
.getAnnotation(B.class);
```

Also an annotated method allows retrieving the source class (`getAnnotatedClass`) and the source method (`getMethod`).

## Conflict of Annotations

If a conflict of annotations has arisen, no inheritance will be made. A conflict takes place when several superclasses or superinterfaces have an annotation of the same kind, and the annotation is not present in the derived class or interface. For instance, the following case has a conflict since both supertypes has the annotation `@A`.

```
@A
public interface BaseInterface {...

@A
public class BaseClass {...

public class Derived extends BaseClass implements BaseInterface{...
```

In this case, the derived class has no the annotation `@A` at all.

Annotation inheritance in the case of collision is impossible because some annotations can have parameters. So, ambiguity arises laying in what parameters to inherit.

The problem can be solved by declaring the annotation `@A` explicitly in the derived class:

```
@A
public class Derived extends BaseClass implements BaseInterface{...
```

The same situation takes place when superclasses and superinterfaces contain several methods having the same signature and annotated with the same kind of annotation, for instance:

```
public interface BaseInterface {
    @B
    public void method1();
}

public class BaseClass {
    @B
    public void method1(){ }
}

public class Derived extends BaseClass implements BaseInterface{...
```

The problem has the same solution as for class annotation inheritance: one should define in the derived class a method having the same signature and the correct annotation:

```
public class Derived extends BaseClass implements BaseInterface{
    @B
    public void method1(){super.method1();}
    ...
}
```

## Overriding Annotations

If a derived class or interface has an annotation on some element which has another annotation of the same kind in superclass or superinterface, then in the derived class or interface

the element will have the annotation declared on it explicitly. Thus, the initial annotation will be overridden, for instance:

```
@A
public interface BaseInterface {...

public class BaseClass {...

@A
public class Derived extends BaseClass implements BaseInterface{...
```

This is useful for annotations having parameters: annotation parameters can be changed and the annotation itself remains in force.