



Articles » Languages » Java » Beginners

Understanding Annotations in Java

By **Azim Zahir**, 24 Oct 2011

★★★★☆ 4.67 (2 votes)

[Download source - 1.89 KB](#)

```
Command Prompt
D:\java>javac Test.java
Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\java>java Test
Welcome to Java
This is an example of Annotations
Author of the class: Azim
Date of Writing the class: 22/10/2011,23/10/2011
Author of the method: Azim
Date of Writing the method: 22/10/2011

D:\java>_
```

Introduction

This article discusses the way of creating annotations in Java. It also shows how to apply annotations to other declarations. Finally it discusses how to obtain annotation information at runtime using Reflection.

Background

Annotation is a new feature introduced by J2SE 5 that allows programmers to embed additional information called metadata into a Java source file. Annotations do not alter the execution of a program but the information embedded using annotations can be used by various tools during development and

deployment.

Using the code

Creating an annotation is similar to creating an interface. But the annotation declaration is preceded by an **@** symbol. The annotation declaration itself is annotated with the **@Retention** annotation. The **@Retention** annotation is used to specify the retention policy, which can be **SOURCE**, **CLASS**, or **RUNTIME**.

- **RetentionPolicy.SOURCE** retains an annotation only in the source file and discards it during compilation.
- **RetentionPolicy.CLASS** stores the annotation in the **.class** file but does not make it available during runtime.
- **RetentionPolicy.RUNTIME** stores the annotation in the **.class** file and also makes it available during runtime.

```
// Specifying runtime retention policy
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotation
{
    String author();    // Annotation member
    String date();     // Annotation member
}
```

An annotation cannot have the **extends** clause. However, annotations implicitly extend the **Annotation** interface. The body of an annotation consists of method declarations without body. These methods work like fields.

In the above example, I have created two members, **author** and **date** to represent information about the creator, and date of writing of the class and method.

Once an annotation is created, it can be applied to classes, methods, fields, parameters, enums, and so on. While applying an annotation to a declaration, you must provide values for its members as follows:

```
// Applying annotation to the class
@MyAnnotation(author="Azim",date="22/10/2011,23/10/2011")
public class Test
{
    // Applying annotation to the method
    @MyAnnotation(author="Azim",date="22/10/2011")
    public static void testMethod()
    {
        System.out.println("Welcome to Java");
        System.out.println("This is an example of Annotations");
    }
    public static void main(String args[])
    {
        testMethod();
        showAnnotations();
    }
}
```

Annotations can be queried at runtime using Reflection as follows:

```
public static void showAnnotations()
// Function to show annotation information
{
```

```
Test test=new Test(); // Instantiating Test class
try
{
    Class c=test.getClass(); // Getting Class reference
    Method m=c.getMethod("testMethod"); // Getting Method reference
    // Getting Class annotation
    MyAnnotation annotation1=
        (MyAnnotation)c.getAnnotation(MyAnnotation.class);
    // Getting Method annotation
    MyAnnotation annotation2=m.getAnnotation(MyAnnotation.class);
    // Displaying annotation information
    System.out.println("Author of the class: "+annotation1.author());
    // Displaying annotation information
    System.out.println("Date of Writing the class: "+annotation1.date());
    // Displaying annotation information
    System.out.println("Author of the method: "+annotation2.author());
    // Displaying annotation information
    System.out.println("Date of Writing the method: "+annotation2.date());
}
catch(NoSuchMethodException ex)
{
    System.out.println("Invalid Method..." +ex.getMessage());
}
}
```

In the above code, I have queried annotations applied to the class **Test** as well as the method **testMethod()**.

To obtain annotation information for the class, the following statement is used:

```
MyAnnotation annotation1=(MyAnnotation)c.getAnnotation(MyAnnotation.class);
```

To obtain annotation information for the method, the following statement is used:

```
MyAnnotation annotation2=m.getAnnotation(MyAnnotation.class);
```

The annotation information is printed by using annotation objects.

Points of interest

The Java program is compiled on the command line as follows:

```
javac Test.java
```

and executed as follows:

```
java Test
```

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

About the Author



Azim Zahir

Instructor / Trainer NIIT, India


India 
Member

I am a trainer by profession. Currently I am working with [NIIT \(Mumbai, India\)](#) as a Senior Faculty. I enjoy programming as a hobby. My favorite technologies are Flash, Flex and Silverlight.

Of late I have developed keen interest in WPF and Windows Mobile programming.

Apart from computers, my favorite pastime is bicycling.

Comments and Discussions

 **0 messages** have been posted for this article Visit

<http://www.codeproject.com/Articles/272736/Understanding-Annotations-in-Java> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Mobile](#)
Web02 | 2.6.121206.1 | Last Updated 23 Oct 2011

Article Copyright 2011 by Azim Zahir
Everything else Copyright © [CodeProject](#), 1999-2012
[Terms of Use](#)