Annotations in Java

Annotations are used to provide supplement information about a program.

- Annotations start with '**@**'.
- Annotations do not change action of a compiled program.
- Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.
- Annotations are not pure comments as they can change the way a program is treated by compiler. See below code for example.

```java
/* Java program to demonstrate that annotations are
   not barely comments (This program throws compiler
   error because we have mentioned override, but not
   overridden, we haver overloaded display) */
class Base
{
     public void display()
     {
         System.out.println("Base display()");
     }
}
class Derived extends Base
{
     @Override
     public void display(int x)
     {
         System.out.println("Derived display(int )");
     }

     public static void main(String args[])
     {
         Derived obj = new Derived();
         obj.display();
     }
}
```

Run on IDE

Output :

```
10: error: method does not override or implement
    a method from a supertype
```

If we remove parameter (int x) or we remove @override, the program compiles fine.

Categories of Annotations

There are 3 categories of Annotations:-

1. Marker Annotations:

The only purpose is to mark a declaration. These annotations contain no members and do not consist any data. Thus, its presence as an annotation is sufficient. Since, marker interface contains no members, simply determining whether it is present or absent is sufficient. **@Override** is an example of Marker Annotation.

```
Example: - @TestAnnotation()
```

2. Single value Annotations:

These annotations contain only one member and allow a shorthand form of specifying the value of the member. We only need to specify the value for that member when the annotation is applied and don't need to specify the name of the member. However in order to use this shorthand, the name of the member must be **value.**

WOO COMMERCE
Build Your Online Store, Your Way.
Get the most customizable eCommerce platform for building your online business.
Get Started

```
Example: - @TestAnnotation("testing");
```

3. Full Annotations:

These annotations consist of multiple data members/ name, value, pairs.

```
Example:- @TestAnnotation(owner="Rahul", value="Class Geeks")
```

Predefined/ Standard Annotations

Java defines seven built-in annotations.

- Four are imported from java.lang.annotation: **@Retention**, **@Documented**, **@Target**, and **@Inherited**.
- Three are included in java.lang: **@Deprecated, @Override** and **@SuppressWarnings**

@Deprecated Annotation

- It is a marker annotation. It indicates that a declaration is obsolete and has been replaced by a newer form.
- The Javadoc @deprecated tag should be used when an element has been deprecated.
- @deprecated tag is for documentation and @Deprecated annotation is for runtime reflection.
- @deprecated tag has higher priority than @Deprecated annotation when both are together used.

```java
public class DeprecatedTest
{
    @Deprecated
    public void Display()
    {
        System.out.println("Deprecatedtest display()");
    }

    public static void main(String args[])
    {
        DeprecatedTest d1 = new DeprecatedTest();
        d1.Display();
    }
}
```

Run on IDE

Output:

```
Deprecatedtest display()
```

@Override Annotation

It is a marker annotation that can be used only on methods. A method annotated with **@Override** must override a method from a superclass. If it doesn't, a compile-time error will result (see this for example). It is used to ensure that a superclass method is actually overridden, and not simply overloaded.

Example:-

```java
class Base
{
     public void Display()
     {
         System.out.println("Base display()");
     }

     public static void main(String args[])
     {
         Base t1 = new Derived();
         t1.Display();
     }
}
class Derived extends Base
{
     @Override
     public void Display()
     {
         System.out.println("Derived display()");
     }
}
```

Run on IDE

Output:

```
Derived display()
```

@SuppressWarnings

It is used to inform the compiler to suppress specified compiler warnings. The warnings to suppress are specified by name, in string form. This type of annotation can be applied to any type of declaration.

Java groups warnings under two categories. They are : **deprecation** and **unchecked.**. Any unchecked warning is generated when a legacy code interfaces with a code that use generics.

```java
class DeprecatedTest
{
    @Deprecated
    public void Display()
    {
        System.out.println("Deprecatedtest display()");
    }
}

public class SuppressWarningTest
{
    // If we comment below annotation, program generates
    // warning
    @SuppressWarnings({"checked", "deprecation"})
    public static void main(String args[])
    {
        DeprecatedTest d1 = new DeprecatedTest();
        d1.Display();
    }
}
```

Run on IDE

Output:

```
Deprecatedtest display()
```

@Documented Annotations

It is a marker interface that tells a tool that an annotation is to be documented. Annotations are not included by Javadoc comments. Use of @Documented annotation in the code enables tools like Javadoc to process it and include the annotation type information in the generated document.

@Target

It is designed to be used only as an annotation to another annotation. **@Target** takes one argument, which must be constant from the **ElementType** enumeration. This argument specifies the type of declarations to which the annotation can be applied. The constants are shown below along with the type of declaration to which they correspond.

Target Constant	Annotations 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

We can specify one or more of these values in a **@Target**annotation. To specify multiple values, we must specify them within a braces-delimited list. For example, to specify that an annotation applies only to fields and local variables, you can use this @Target annotation: @Target({ElementType.FIELD, ElementType.LOCAL_VARIABLE}) **@Retention Annotation** It determines where and how long the annotation is retent. The 3 values that the @Retention annotation can have:

- **SOURCE:** Annotations will be retained at the source level and ignored by the compiler.
- **CLASS:** Annotations will be retained at compile time and ignored by the JVM.
- **RUNTIME:** These will be retained at runtime.

@Inherited

@Inherited is a marker annotation that can be used only on annotation declaration. It affects only annotations that will be used on class declarations. **@Inherited** causes the annotation for a superclass to be inherited by a subclass. Therefore, when a request for a specific annotation is made to the subclass, if that annotation is not present in the subclass, then its superclass is checked. If that annotation is present in the superclass, and if it is annotated with **@Inherited,** then that annotation will be returned.

User-defined/ Custom Annotations

User-defined annotations can be used to annotate program elements, i.e. variables, constructors, methods, etc. These annotations can be applied just before declaration of an element (constructor, method, classes, etc).

Syntax of Declaration:-

```
[Access Specifier] @interface<AnnotationName>
{
   DataType <Method Name>() [default value];
}
```

- **AnnotationName** is an identifier.
- Parameter should not be associated with method declarations and **throws** clause should not be used with method declaration.
- Parameters will not have a null value but can have a default value.
- ***default value*** is optional.
- Return type of method should be either primitive, enum, string, class name or array of primitive, enum, string or class name type.

```java
package source;
// A Java program to demonstrate user defined annotations
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

// user-defined annotation
@Documented
@Retention(RetentionPolicy.RUNTIME)
@ interface TestAnnotation
{
    String Developer() default "Rahul";
    String Expirydate();
} // will be retained at runtime

// Driver class that uses @TestAnnotation
public class Test
{
    @TestAnnotation(Developer="Rahul", Expirydate="01-10-2020")
    void fun1()
    {
        System.out.println("Test method 1");
    }

    @TestAnnotation(Developer="Anil", Expirydate="01-10-2021")
    void fun2()
    {
        System.out.println("Test method 2");
    }

    public static void main(String args[])
    {
        System.out.println("Hello");
    }
}
```

Run on IDE

Output :

```
Hello
```