



# Annotation



# Agenda

---

- 1 What is an Annotation**
- 2 Annotation used by the compiler**
- 3 Categories of an Annotation**

# Objectives (Contd.).

---

- Define Annotation
- Understand the Annotation used by compiler
- Differentiate between categories of annotation
- Explore the advantages using annotation

# What is an Annotation?



# What is an Annotation?

---

- Annotation is a new feature added in J2SE 5.0 (Tiger)
- Annotations are used to add meta-data to the Java Elements
- Annotations leads to a declarative programming style where the programmer says what should be done and the tools emit the code for it
- It is a mechanism for associating a meta-tag with program elements and allowing the compiler or the VM to extract program behaviors
- It is a special form of metadata that can be added to any program element
  - Classes, methods, variables, parameters and Packages can be annotated

# What is an Annotation?

---

Annotations are used to add meta-data information to an element ( The element can be a class, a method, a field, or anything). The meta-data added to these elements are processed by tools like compilers, javadoc, etc.

Let us look at an example to understand What is a Metadata?

```
final class Emp
{
String name;
int id;
Employee(String n,int id)
{
this.name=name;
this.id=id;
}
}
```

# What is an Annotation?

- In the previous example we have a final class called Emp. The final keyword before the class indicates that the class cannot be subclassed. So, the inclusion of the final keyword, to the class adds some additional information to the class definition, that no other class can extend the Emp class. Therefore, the final keyword forms a part in providing **meta-data** information in the class definition.
- An annotation definition looks like the following,
- TestAnnotation.java
- public @interface TestAnnotation
- {
- String str();
- int val();
- }
- An annotation is created through a mechanism based on the interface.

# How an annotation looks?

---

```
public @interface TestAnnotation
{
    String str();
    int val();
}
```



# Simple Example

- The annotation definition

```
@interface author
{
    String value() default "Patrick Norton";
}
```

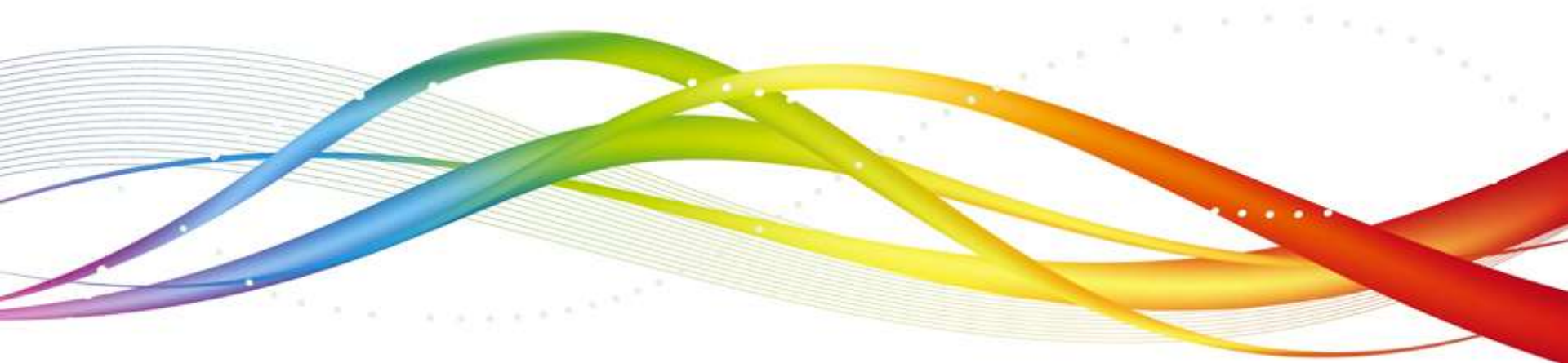
- Defines an "author" annotation
- Has one string attribute (value)

- The usage

```
@author (value="Sriram")
public void calculateEMI()
{
}
```

- Adds author annotation as modifier to the method calculateEMI

# Annotations used by the Compiler



# Annotations used by the Compiler

---

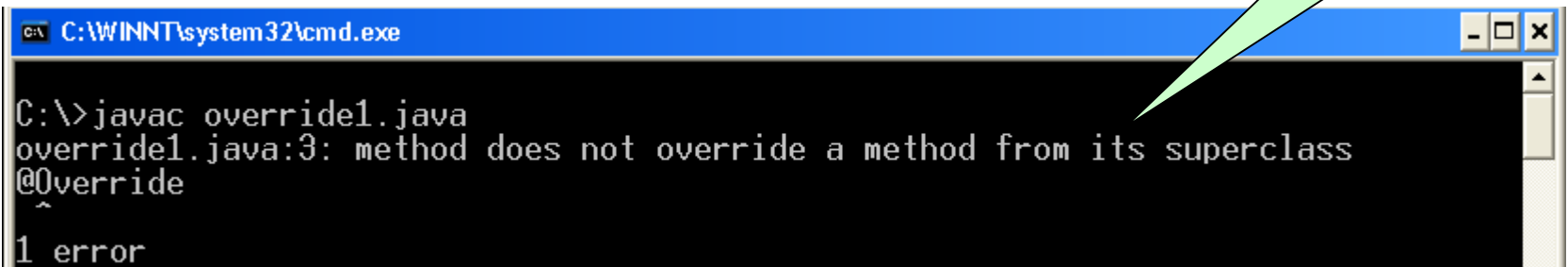
- There are three annotation types that are predefined by the language specification:
  - @Override
  - @Deprecated
  - @SuppressWarnings
  - These are examples of **Simple annotations**
  - Simple annotations are annotations that can be used only in code
  - They cannot be used to create custom annotation types

# @Override

- Used to check if the function is an override
- It produces a compilation error if the method does not exist in the parent class

```
class override1
{
@Override
public String toString()
{
return "Example of Override annotation";
}
}
```

Output



The screenshot shows a Windows command prompt window with the title bar 'C:\WINNT\system32\cmd.exe'. The command prompt displays the following text:

```
C:\>javac override1.java
override1.java:3: method does not override a method from its superclass
@Override
^
1 error
```

A green speech bubble with the word 'Output' points to the error message in the command prompt.

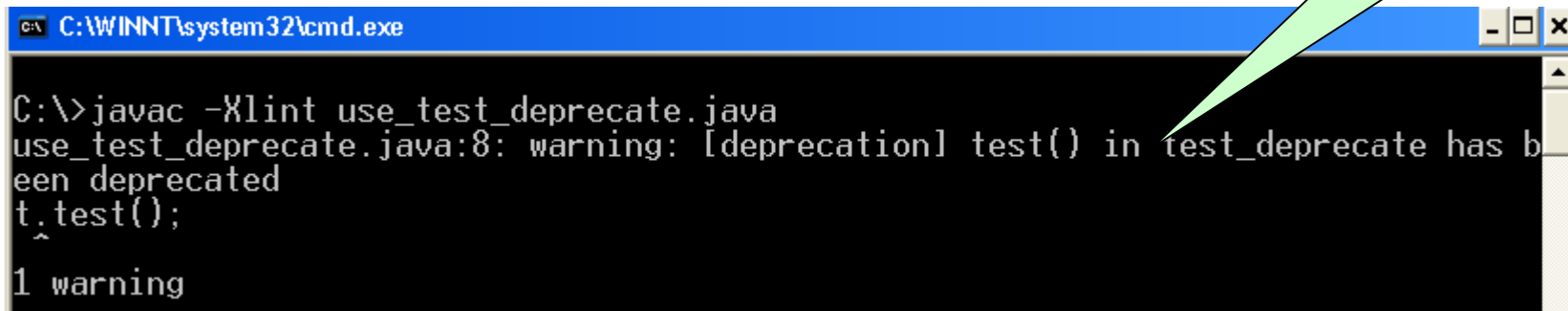
# @Deprecated

- Used to mark a method obsolete
- It produces a warning if the function is used

```
class test_deprecate
{
@Deprecated
void test()
{
System.out.println("Testing
deprecation");
}
```

```
class use_test_deprecate
{
public static void main(String ae[])
{
test_deprecate t=new test_deprecate();
t.test();
}
```

Output



```
C:\WINNT\system32\cmd.exe
C:\>javac -Xlint use_test_deprecate.java
use_test_deprecate.java:8: warning: [deprecation] test() in test_deprecate has been deprecated
t.test();
1 warning
```

# @Deprecated

---

- The interfaces and classes in an application gets updated every now and then. The Interfaces used by the clients may undergo so many revisions, a lot of new methods can be added and some of the existing methods may be removed or updated.
- Imagine the case where a method defined in a class or an interface has now become obsolete. We should warn the client applications not to use them anymore. Another way is to remove the method itself from the interface. But this approach will have a huge impact on the code that is dependent on this interface.

# @Deprecated

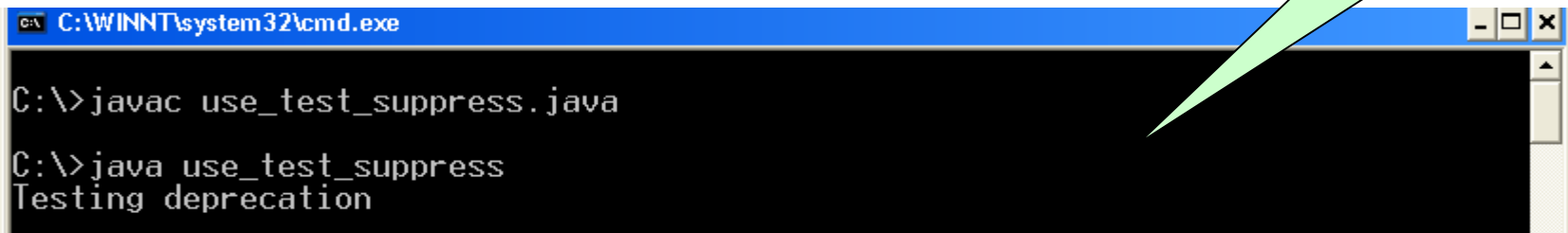
- Another elegant way is to mark the old method as deprecated which tells the client not to use this method anymore because in the future versions this old method may not be supported. Clients may prepare themselves not to depend on the old method anymore.
- In the example above test\_deprecate has a method test() which is tagged as deprecated. Now if any of the client code tries to access this method, then it throws the following error
- C:\>javac -Xlint use\_test\_deprecate.java
- use\_test\_deprecate.java:8: warning: [deprecation] test() in test\_deprecate has been deprecated
- t.test();
- ^
- 1 warning

# @SuppressWarnings

- Used to instruct the compiler to suppress the warnings specified in the annotation parameters

```
class test_deprecate
{
    @Deprecated
    void test()
    {
        System.out.println
        ("Testing deprecation");
    }
}
```

```
class use_test_suppress
{
    @SuppressWarnings({"deprecation"})
    public static void main(String
        ae[])
    {
        test_deprecate t=new
            test_deprecate();
        t.test();
    } }
```



```
C:\WINNT\system32\cmd.exe
C:\>javac use_test_suppress.java
C:\>java use_test_suppress
Testing deprecation
```

Output



# @SuppressWarnings

---

- **@SuppressWarnings** specifies that one or more warnings that might be issued by the compiler are to be suppressed. The warnings to suppress are specified by name, in String form. This annotation can be applied to any type of declaration. In the above example you can see that the compiler is not creating a warning about using a deprecated method, as we have used the the SuppressWarnings annotation to suppress the deprecation warnings.

# Review Questions

---

1. Which of the following keyword is used to create an user defined annotation?
  - a. class
  - b. enum
  - c. interface
  - d. None of the above
  
2. Which of the following annotation will be used only in case of Inheritance?
  - a. @Deprecated
  - b. @SuppressWarnings
  - c. @Override

# Categories of an Annotation



# Categories of Annotation – Marker Annotation

- Marker Annotation
  - Contains only the name
  - Does not contain any other element
  - Creation

```
public @interface marker
{
}
```

- Usage

```
@marker
public void sampleMethod
{
}
```

A *marker* annotation is a special kind of annotation that contains no members. Its sole purpose is to mark a declaration. Thus, its presence as an annotation is sufficient.

The best way to determine if a marker annotation is present is to use the method **isAnnotationPresent( )**, which is defined by the **AnnotatedElement** interface.

# Categories of Annotation – Single value Annotation

- Single value Annotation

- They provide a single piece of data
- Can be provided as a data value pair or can use shortcut and provide the value within quotation marks
- Creation

```
@interface author
{
    String value() default "Patrick Norton";
}
```

- Usage

@author("Sriram")	(or)
public void calculateEMI()	@author(value="Sriram")
{	public void calculateEMI()
	{
}	}

# Categories of Annotation – Single value Annotation

---

- A *single-member* annotation contains only one member. It works like a normal annotation except that it allows a shorthand form of specifying the value of the member. When only one member is present, you can simply specify the value for that member when the annotation is applied—you 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**.

# Categories of Annotation – Multi value Annotation

- Multi value/Full value Annotation
  - They can have multiple data members
  - We have to pass value to all the data members
- Creation

```
@interface calldetails
{
String severity() default "Medium";
String personAssigned();
int no_of_escalations();
String date();
}
```

# Categories of Annotation – Multi value Annotation

- Usage

```
class itimhelpdesk
{
    @calldetails(severity="High",personAssigned="GovindSamy",no_of_escalations=0,date="1-2-2012")
    public void logCall()
    {
    }
}
```



# Review Questions

---

1. Which of the following annotation is an example of Marker Annotation?
  - a. @Deprecated
  - b. @SuppressWarnings
  - c. @Override
  
2. Which of the following is an example of Multi value Annotation?
  - a. @Deprecated
  - b. @SuppressWarnings
  - c. @Override

# Meta Annotations

---

- They are used to annotate the annotation type declaration
- There are 4 types of Meta annotations
  - Target
  - Retention
  - Documented
  - Inherited

Metadata is a way to add some supplement information to the source code. This metadata can be used by other tools such as source code generator for instance to generate additional code at the runtime.

# Meta Annotations - Target

- **Target**

- It is used to specify which element of the class to be annotated
- **@Target(ElementType.TYPE)** - can be applied to any element of a class
- **@Target(ElementType.FIELD)** - can be applied to field or property
- **@Target(ElementType.PARAMETER)** - can be applied to the parameters of a method
- **@Target(ElementType.LOCAL\_VARIABLE)** - can be applied to local variables
- **@Target(ElementType.METHOD)** - can be applied to method level annotation
- **@Target(ElementType.CONSTRUCTOR)** - can be applied to constructors
- **@Target(ElementType.ANNOTATION\_TYPE)** - used to specify that the declared type itself is an annotation type

# Meta Annotations – Target (Contd.).

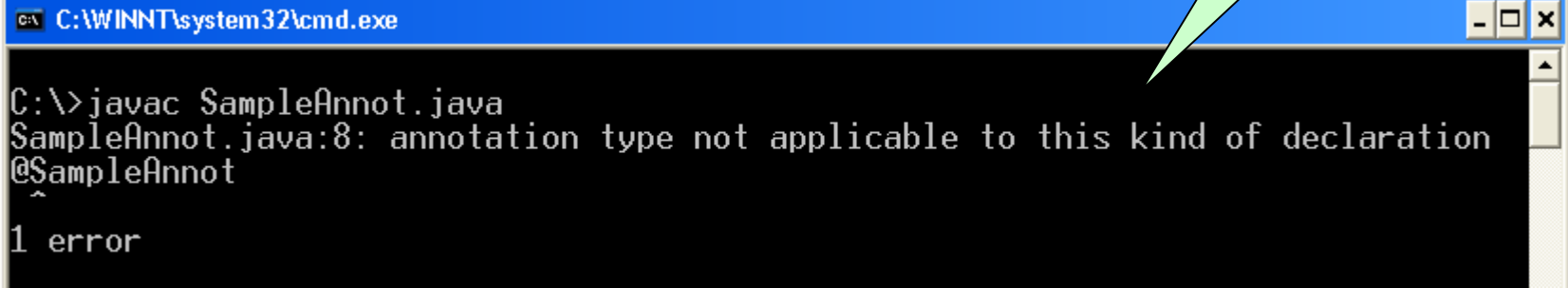
- Example

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Target;
@Target(ElementType.METHOD)
public @interface SampleAnnot{

}
```

```
@SampleAnnot
class Example
{
}
```

Output



A screenshot of a Windows command prompt window titled "C:\WINNT\system32\cmd.exe". The window has a blue title bar and a black background. The text inside shows the command "C:\>javac SampleAnnot.java" followed by the error message "SampleAnnot.java:8: annotation type not applicable to this kind of declaration" and "@SampleAnnot" with a tilde under the 'S'. At the bottom, it says "1 error". A green speech bubble with the word "Output" points to the error message.

```
C:\>javac SampleAnnot.java
SampleAnnot.java:8: annotation type not applicable to this kind of declaration
@SampleAnnot
~
1 error
```

# Meta Annotations – Target (Contd.).

The `@Target(ElementType.METHOD)` indicates that this annotation type can be used to annotate only at the method levels.

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Target;
@Target(ElementType.METHOD)
public @interface SampleAnnot{
```

```
}
class Example
{
    @SampleAnnot
    public void sampleMethod()
    {}
}
```

The above code will compile without any problem because `SampleAnnot` is an annotation that can be used only at the method level. If it will be used in any other level the compiler will throw compilation error.

# Meta Annotations - Retention

---

- Used to indicate how long annotations of this type are to be retained
- There are 3 values
- **RetentionPolicy.SOURCE**
  - These annotations will be retained only at the source level and will be ignored by the compiler
- **RetentionPolicy.CLASS**
  - These annotations will be by retained by the compiler at compile time, but will be ignored by the VM
- **RetentionPolicy.RUNTIME**
  - These annotations will be retained by the VM so they can be read at run-time

# Meta Annotations – Retention (Contd.).

---

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
@interface author
{
    String value() default "Patrick Norton";
}
```

# Meta Annotations – Retention (Contd.).

- An annotation need to have a RetentionPolicy that will be the scope of the annotation, where at this point the annotation will be ignored or discarded. The values are RetentionPolicy.SOURCE, RetentionPolicy.CLASS and RetentionPolicy.RUNTIME. When no retention policy defined it will use the default retention policy which is **RetentionPolicy.CLASS**
- Annotation with SOURCE retention policy will be retained only in the source code, it available to the compiler when it compiles the class and will be discarded after that. The CLASS retention policy will make the annotation stored in the class file during compilation, but will not available during the runtime. And the RUNTIME retention policy will stored the annotation in the class file during compilation and it also available to JVM at runtime.



# Meta Annotations - Documented

- Used to inform that an annotation with this type should be documented by the javadoc tool

```
import java.lang.annotation.Documented;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;

@Documented
@Target({ElementType.METHOD })
@interface author
{
    String value();
}

public class Example
{
    @author("Anitha")
    public void sampleMethod()
    {
    }
}
```

# Meta Annotations – Documented (Contd.).

## Method Summary

void	<a href="#">sampleMethod()</a>
------	--------------------------------

## Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

## Constructor Detail

### Example

```
public Example()
```

## Method Detail

### sampleMethod

```
@author(value="Anitha")  
public void sampleMethod()
```

[Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

# Meta Annotations - Inherited

- A child class inherits the annotation which is marked with `@Inherited` annotation

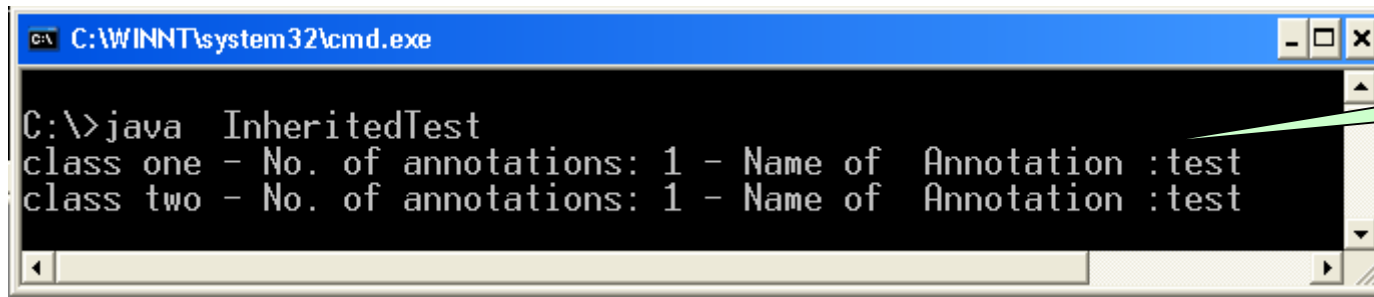
Inherit.java

```
-----  
@Target (ElementType.TYPE)  
@Retention (RetentionPolicy.RUNTIME)  
@Inherited  
@interface TestAnnotation{  
    String value();  
}  
@TestAnnotation("test")  
class one { }  
class two extends one  
{ }
```

# Meta Annotations – Inherited (Contd.).

```
import java.lang.annotation.Annotation;

public class InheritedTest {
    public static void main(String[] args) {
        Class[] classes = {one.class, two.class};
        for (Class classObj : classes) {
            System.out.print(classObj+" - ");
            Annotation[] annotations = classObj.getAnnotations();
            System.out.print("No. of annotations: " +
                annotations.length);
            for (Annotation annotation : annotations) {
                TestAnnotation t = (TestAnnotation)annotation;
                System.out.println(" - Name of Annotation :"+t.value());
            }
        }
    }
}
```

A screenshot of a Windows command prompt window. The title bar is blue and reads "C:\WINNT\system32\cmd.exe". The command prompt shows the command "C:\>java InheritedTest" and its output: "class one - No. of annotations: 1 - Name of Annotation :test" and "class two - No. of annotations: 1 - Name of Annotation :test". A green speech bubble points to the output text.

```
C:\WINNT\system32\cmd.exe

C:\>java InheritedTest
class one - No. of annotations: 1 - Name of Annotation :test
class two - No. of annotations: 1 - Name of Annotation :test
```

Output

# Advantages with Annotation

---

- Annotations helps to shift the responsibility of writing boilerplate code from the programmer to the Compiler or other tools
- The resulting code is less error prone
- Provides information to the compiler
  - It can be used by the compiler to detect errors or suppress warnings
- Compiler-time and deployment-time processing
  - Software tools can process annotation information to generate code, XML files, and so forth
- Runtime processing
  - Some annotations are available to be examined at runtime

# Quiz

---

1. Which of the following symbol is used to represent an annotation?
  - a. #
  - b. \$
  - c. @
  - d. %
2. Which of the following is a valid annotation?
  - a. `public interface NewAnnotation{`
  - b. `public annotation NewAnnotation{`
  - c. `public @interface NewAnnotation{`
  - d. `public @annotation NewAnnotation{`
3. Can you apply more than one annotation in a Target?
  - a. Yes
  - b. No

# Quiz

---

4. Which of the following retention policy and type defined @Override?
  - a. Class,Method
  - b. Source,Method
  - c. Runtime,Class
  - d. Type,Class
5. Which of the following are valid retention policies available in Java?
  - a. CODE
  - b. SOURCE
  - c. TOOLS
  - d. RUNTIME

# Assignment

---





# Summary

---

- In this module, you have understood
  - What is an Annotation?
  - What are the uses of an Annotation?
  - How to create a user defined Annotation?
  - The various types of Annotations and their usage



**Thank you**

