1. **Definition**
2. **Annotation Types.**
3. **General Purpose Annotation.**
4. **Custom Annotation.**
5. **Meta Annotation.**

**Definition**

The Java developer write description(Comment) of elements (class, interface, enumeration, constructors, methods and fields) in source code. The comment of element is also called as metadata of element. The description can’t be embedded in and read from .class file generated by java compiler and the description is not available to JVM.

Upto Java 1.4, The XML technology was used to bring the comment of elements into .class file and into run time environment. By using this technology, there were several problems.

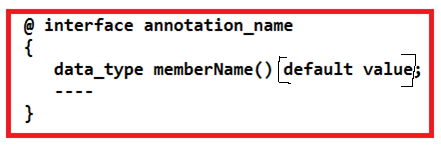
* The developer must learn XML technology.
* The developer must check whether xml document is well form document or not.
* The developer must check whether xml document is placed in right place or not.

To overcome these problems, sun micro system introduced annotation technology in Java 1.5v. java 1.5 onwards, we can use either xml technology or annotation technology.

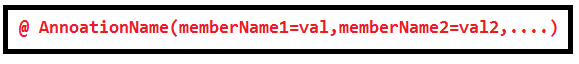
**1.Definition:**-Annotation is tag. The annotation helps to associate metadata(Information) to the program elements(class,interface,constructor,method and field). The annotation is used to provide additional information of programming elements to compiler and JVM.

The annotations start with ‘@’.

Syntax to declare the Annotation:



Syntax to Annotation Utilization:



**ANNOTATION TYPES**

**1.Annotation Types:-** Based on no.of elements in Annotation , we divide the annotation into 3 types.

a) Marker Annotation

b)single valued Annotation.

c) Multi valued Annotation.

**A)Marker Annotation:-** we declare annotation without member. That annotation is marker annotation.

Ex:-

@interface suku{}

**B) single Valued Annotation:-** Single Valued Annotation has only single member.

Ex:-

@interface sv{ int salary()default 20;}

C) MultiValued Annotation:- We declare annotation with multiple members. That annotation is multivalued annotation.

Ex:-

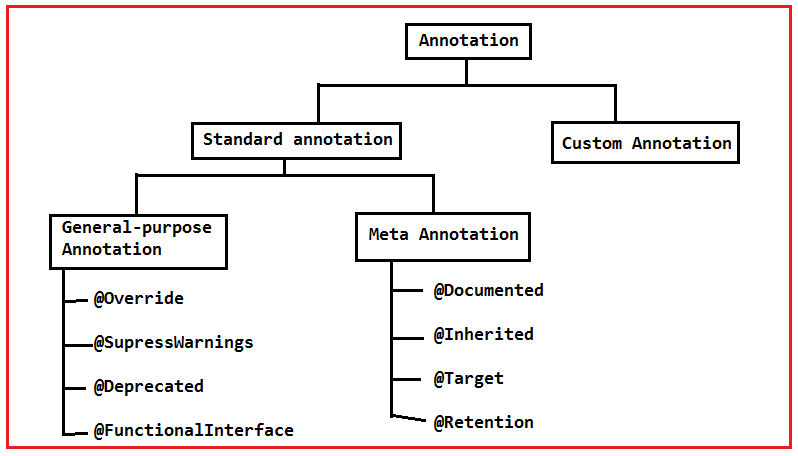
@interface Rock{

Int salary();

Name String() default “suku”;

}

**1.1. General Annotation Types:-**



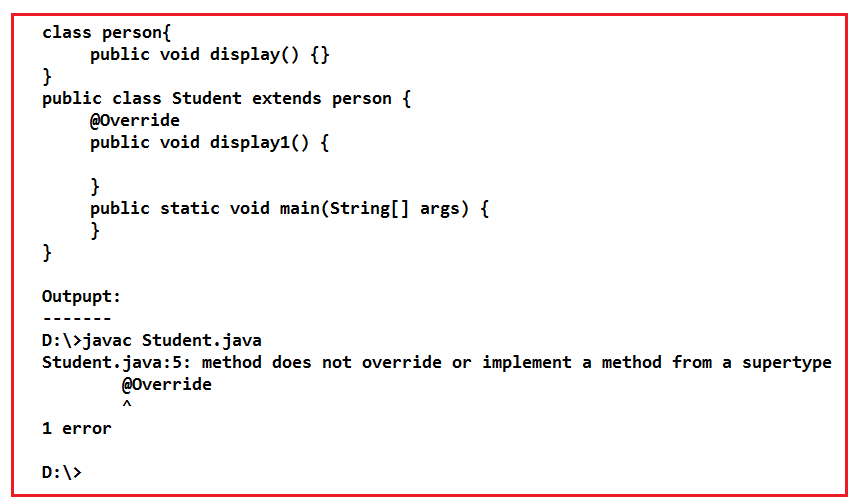
All annotations are in **java.lang.annotations** package.

**GENERAL PURPOSE ANNOTATION**

**3.General-purpose Annotation:**

[**3.1.@override**](mailto:3.1.@override)**:** It is a marker annotation that can be used only on methods. sometimes, the programmer does silly mistake(spelling mistake) in overriding method name. The compiler does not show error message because compiler does not suppose the method as overriding method.

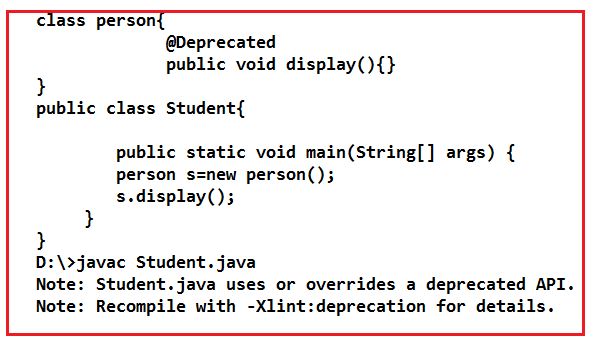
@override annotation assures that the subclass method is overriding the parent class method, if it is not so, compile time occurs.

Example:-

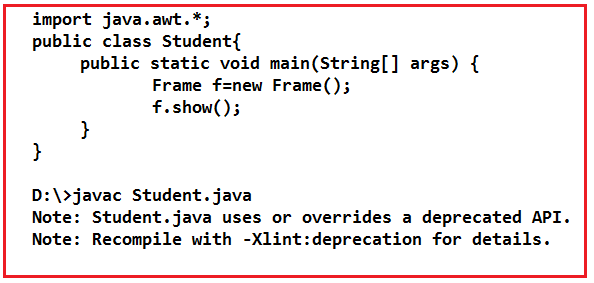
[3.2.@Deprecated:-](mailto:3.2.@Deprecated:-) It is a marker annotation. It indicates that a declaration is obsolete and has been replaced by newer form.

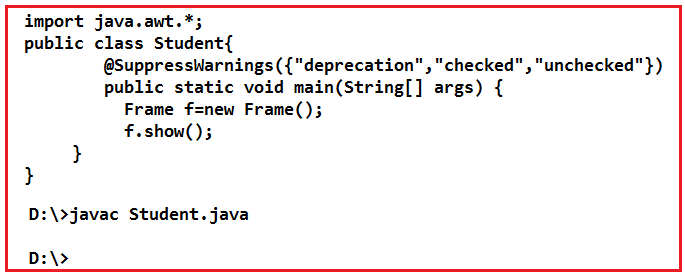
If we want to get warning message regarding method as “Deprecated API”, then we should use “Deprected” annotation.

Example:-



[3.3.@SuppressWarnings:-](mailto:3.3.@SuppressWarnings:-) It is used to suppress the warnings issued by the compiler. It is single valued Annotations.





[3.4.@FunctionalInterface:-](mailto:3.4.@FunctionalInterface:-) This annotation is available java8 onwards.

The Functional Interface has only one abstract method. The functional interface does not allow zero or more than one abstract methods.

When Annotation(@FunctionalInterface) is applied to interface, the interface will allows only one abstract method . If interface has zero or more than one method , we will get compile time error.

Example:1

@FunctionalInerface

Interface abc{

Public void display();

}

No CE.

Example:2

@FunctionalInterface

Interface abc{}

It leads to CE.

Example:3

@FunctionalInterface

Interface abc{

Public void display();

Public void display1():

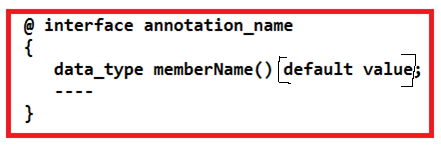
}

It leads to CE.

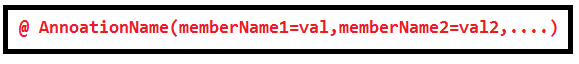
**CUSTOM ANNOTATION**

**4.Custom Annotations:-**

Syntax to declare the Custom Annotation:



Syntax to Custom Annotation Utilization:



4.1.Steps:-

a. Open the file. Save the file with .java extension.

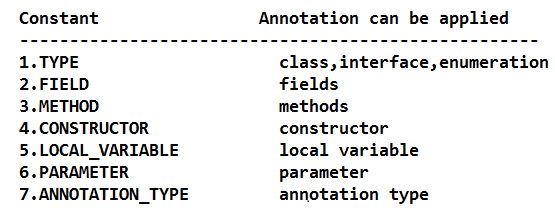
b. Write User-defined annotation in that file.

**META ANNOTAIONS**

**5.MetaAnnotations:-**

**5.1.@Target:-** we apply annotations to variables, class, methods, interface, other annotations …etc. The @Target annotation is used to specify programming element to which annotation can be applied.

The java.lang.annotation.ElementType enum declares many constants to specify the type of element where annotation is to applied.



Example:1

Course.java

**package** com.raos;

**import** java.lang.annotation.\*;

@Target(ElementType.***TYPE***)

@Retention(RetentionPolicy.***RUNTIME***)

**@interface** Course {

**int** cid()**default** 1;

String cname()**default** "mca";

}

Student.java

**package** com.raos;

**import** java.lang.annotation.Annotation;

@Course(cid=7,cname="MCA")

**public** **class** Student {

**int** sid=9;

String sname="sukumar";

**public** **void** display() {

System.***out***.println(sid);

System.***out***.println(sname);

}

**public** **static** **void** main(String[] args) {

Student s1=**new** Student();

Class c=s1.getClass();

Annotation a1=c.getAnnotation(Course.**class**);

Course c2=(Course)a1;

s1.display();

System.***out***.println(c2.cid());

System.***out***.println(c2.cname());

}

}

Output:

9

sukumar

7

MCA

Example:2

Course.java

**import** java.lang.annotation.\*;

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***RUNTIME***)

**@interface** Course {

**int** cid()**default** 1;

String cname()**default** "mca";

}

Student.java

**import** java.lang.annotation.Annotation;

**import** java.lang.reflect.Method;

**public** **class** Student {

**int** sid=9;

String sname="sukumar";

@Course(cid=7,cname="MCA")

**public** **void** display() {

System.***out***.println(sid);

System.***out***.println(sname);

}

**public** **static** **void** main(String[] args) **throws** Exception {

Student s1=**new** Student();

Class c=s1.getClass();

Method m1=c.getMethod("display");

Annotation a1=m1.getAnnotation(Course.**class**);

Course c2=(Course)a1;

s1.display();

System.***out***.println(c2.cid());

System.***out***.println(c2.cname());

}

}

Output:

9

sukumar

7

MCA

[5.2.@Inherited:-](mailto:5.2.@Inherited:-) `By Default, annotations are not inherited to subclass. The @Inherited annotation marks the annotation to be inherited to subclass.

Example:

Course.java

**import** java.lang.annotation.\*;

@Inherited

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***RUNTIME***)

**@interface** Course {

**int** cid()**default** 1;

String cname()**default** "mca";

}

Student.java

**import** java.lang.annotation.Annotation;

**import** java.lang.reflect.Method;

**class** Hello{

@Course(cid=7,cname="MCA")

**public** **void** hai() {

System.***out***.println("Hai");

}

}

**public** **class** Student {

**int** sid=9;

String sname="sukumar";

**public** **void** display() {

System.***out***.println(sid);

System.***out***.println(sname);

}

**public** **static** **void** main(String[] args) **throws** Exception {

Student s1=**new** Student();

Hello h1=**new** Hello();

Class c=h1.getClass();

Method m1=c.getMethod("hai");

Annotation a1=m1.getAnnotation(Course.**class**);

Course c2=(Course)a1;

s1.display();

System.***out***.println(c2.cid());

System.***out***.println(c2.cname());

}

}

Output:

9

sukumar

7

MCA

[5.3.@Documented:-](mailto:5.3.@Documented:-) When u apply annotation to element, By Default annotation will not be listed in HTML Documented(Generated By javadoc tool). This @Documented marks the annotation for inclusion in the HTML documentation.

[5.4.@Retention:-](mailto:5.4.@Retention:-) @Retention annotation is used to specify to what level annotation will be available.

