Table of Contents

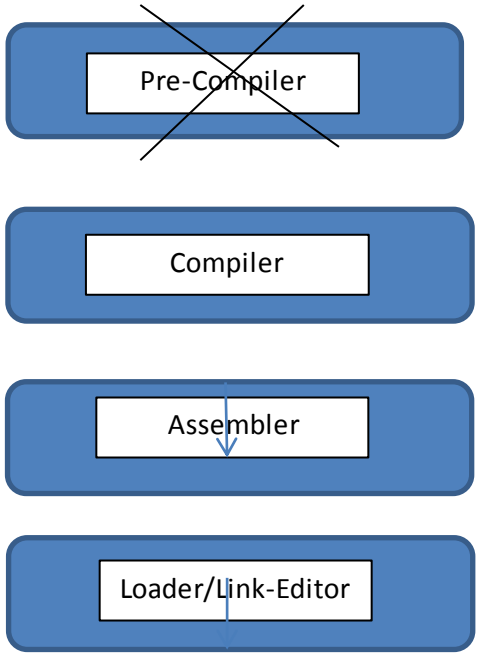
1 Introduction

Annotations are basically to describe metadata in java programs, which is added a new feature in java 1.5 versions. In java, annotations are used to describe **metadata about programs/code.**

- Comments are also provides metadata in some way which is only available visible source code.
- Comments are not available in compiled .class file.

Java follows below steps in compilation process–

Pre-Compiler phase is not available in Java, as all the required libraries are imported from packages instead of header files. Usually Pre-compiler phase used to load header files (which are usually available in C/C++ languages).



Compiler phase once again has following steps/phases –

Lexical analysis

Syntax analysis

Semantix Analysis

Intermediate code generation

Code optimization

Code generation.

In java, lexical analysis phase reads all the java source code and generated tokens for next step, and it also **removes commented code from Source** file before generating the tokens. Because of this comment code will not be available till runtime.

So to Annotations are introduced in 1.5 to provide metadata information which can be available till runtime and we **can extract metadata information programmatically in programs.**

Before Java 1.5 versions, all the metadata information is usually provide in XML documents, and these documents read/parsed from the source code to pull annotated information about a program.

Below are the few issues to use XML documents -

1. Learn XML technology
2. Need to make sure XML documents located/saved properly once each change.
3. XML documents should be formatted properly (well-formed or not).
4. Right parsing mechanism to load and read the document.
5. Length code to describe simple metadata.

Annotations can be used to overcome these problems -

Examples –

Prior to annotations to define a servlet, we need to define all the servlet configuration information in web.xml, information like class name, parameters, mapping etc. To provide on simple servlet we may need to write almost 10 lines of xml document.

This can be avoided simply by one annotations

```
@WebsServlet("/login")

Public class LoginServlet extends HttpServlet{

}
```

2 XML vs Annotation Based Technologies

XML Based	Annotation Based
Upto JDK 1.4 version	JDK 1.5 onwards
JDBC 3.x	JDBC 4.X
Servlets 2.5	Servlets 3.X
Struts 1.X	Struts 2.X
JSF 1.X	JSF 2.X
Hibernate 3.2.4	Hibernate 3.2.5
Spring 2.X	Spring 3.X
EJB's 2.X	EJB 3.X

Most of the technologies support both XML's and annotations for configurations/metadata.

3 Annotation classifications/Types

Annotations execution can be done by apt. – annotation processing tool.

3.1 Declarations and Syntax

Two types of syntax's are available

- Annotation Declaration syntax: -

  @interfae Annotation_Name {
  --Members--
  }
  Members syntax –
  data type member_name() [default value]

- Annotation Utilization syntax -

  In java we can utilize annotation for variables, methods, classes, etc.

  @AnnotationName(member=value, member2=value,etc)

  Programming element

3.2 Annotation classification on the basis of members

On the basis of members, annotations can be classified into below types –

Marker Annotation – Without the members

Ex - @overrid)

Single values annotations – Which contains only one member

Ex - @ suppressWarning("uncheck") or @- suppressWarning(value = "uncheck")

Multiple valued annotations – which contains more than one value

Ex - @WebServlet(name="myservlet", urlPattern="/login")

3.3 Annotation classification

Annotations are classified as below

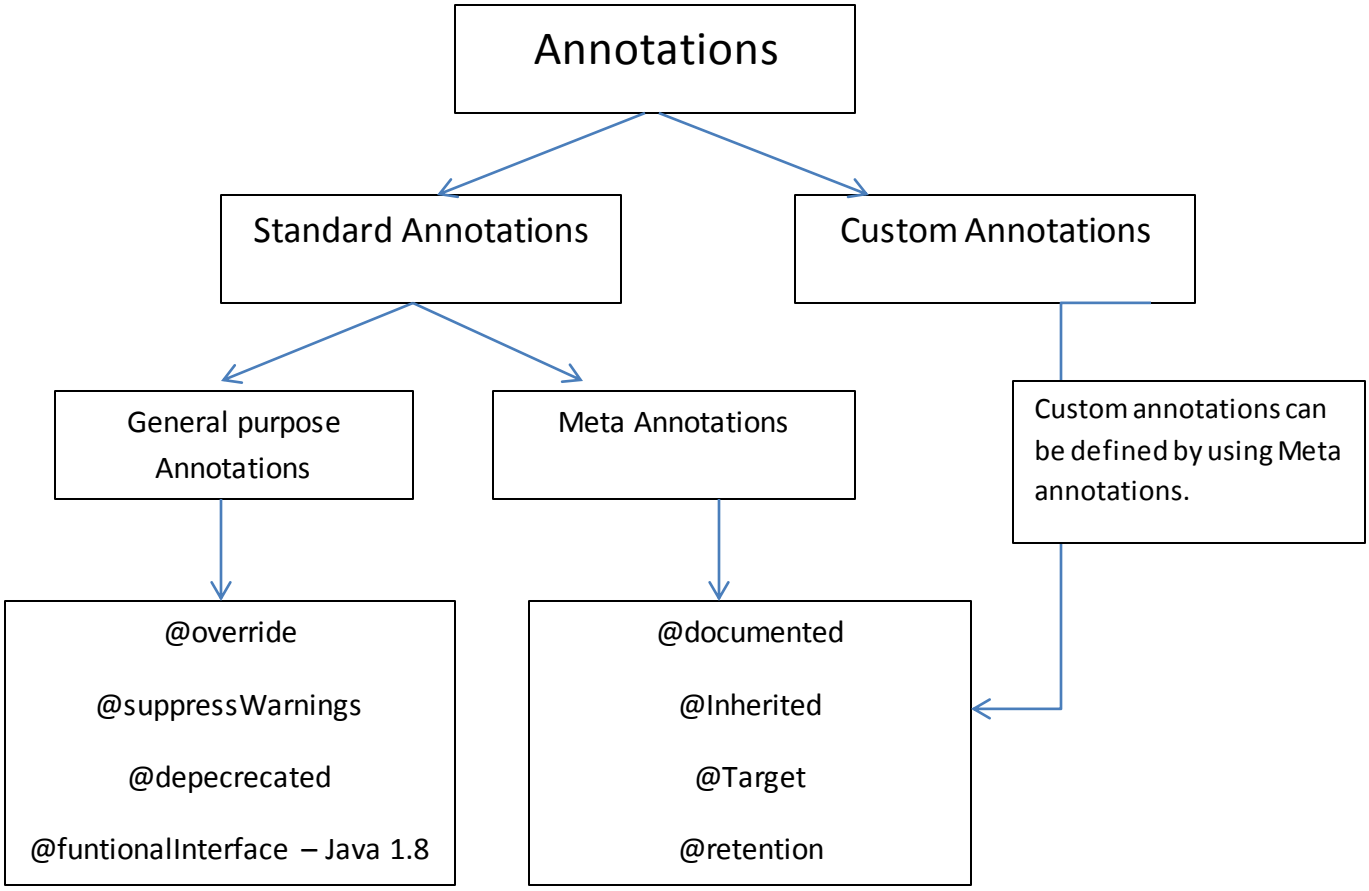
1. Standard Annotations
   a. General purpose Annotations
      i. @override
      ii. @suppressWarnings
      iii. @depecrecated
      iv. @funtionalInterface – Java 1.8

b. Meta Annotations
    i. @documented
    ii. @Inherited
    iii. @Target
    iv. @retention

2. Custom Annotations
    a. Annotations which are defined by using meta annotations.

Below are some of the important points -

- All the standard annotations are available in java.lang package.
- Meta Annotattions are available in java.lang.Annotations package
- **java.lang.annotation.Annotations interface** is parent of all the annotations implicitly.
- All the annotations are interfaces.



4 Standard Annotations

4.1 @Override

Method overriding is a process of replacing super class method functionally within a subclass.

Example –

```
Class DB_Driver {
        Public void getDriver(){
                System.out.println("Type-1 Driver");
        }
}
Class New_DB_Driver extends DB_Driver {
        Public void getDriver(){
                System.out.println("Type-4 Driver");
        }
}
Class Test {
        Public static void main(String[] args){
                DB_Driver  d = new New_DB_Driver ();
                d.getDriver();
        }
}
```

How @override is useful?
If programmer made a typo in subclass methods, then compiler treat that method as a new method, But intention is still to override super class method.
In this context if programmer annotates subclass method with "**@override"** annotation then compiler will catch those kinds of errors and alert programmer.

So above Subclass getDriver() method can be annotate with @override as below –

```
Class New_DB_Driver extends DB_Driver {
        @override
        Public void getDriver(){
                System.out.println("Type-4 Driver");
        }
}
```

4.2 @SuppressWarning

Usually collection can store heterogeneous objects, and in this context collection object is not type safe /Unchecked, so a warning will be generated by compiler to alert programmer.

However programmer knows about the issue of heterogeneous objects in the collection and he doesn't want to generate a warning message by compiler, so to avoid warning message programmer can use @supressWarning annotation.

To hide/ SuppressWarning we use @SuppressWarning annotation.

Ex-

```
Class Bank  {
        @SuppressWarning ("unchecked")
        Public ArrayList getCustomerDetails(){
                ArrayList list = new ArrayList();
                list.add("aaa");
                list.add("bbb");
                Return list;
        }
}
Class  Test {
        Public static void main(String[] args){
                DB_Driver   d = new New_DB_Driver  ();
                d.getDriver();
        }
}
```

4.3 @deprecated

All the outdated/deprecated in java usually show warning message to the programmer to alert. Programmer can use newly implemented method by observing that warning message.

@deprecated can be used to deprecate a method in java. In the below example getSalary() method is deprecated, Hence programmer will get an warning message when this method is called.

```
        @Deprecated
        Public getSalary(){
                System.out.println("Salary cal by using Basic and HRA");
        }
        Public getNewSalary(){
                System.out.println("Salary cal by using Basic, HRA and TA");
        }
```

4.4 @FunctionalInterface

If we declare an interface with only one abstract method exactly then we call that interface as functional Interface, and we must declare suck kind of interface as functional Interface by using @functionalInterface annotation.

Ex –

```
@FunctionalInterface
Interface Account {
        Public void getAccount();
        Public void getBankDetails();
}
Class Test {
        Public static void main (Stirng[] arg){
        }
}
```

We will get a compilation error in this case saying – Multiple abstract methods provided for Functional Interface.

To avoid the error, we need to have only one abstract method in interface like below –

```
@FunctionalInterface
Interface Account {
        Public void getAccount();
}
```

5 Meta Annotations

Annotation about annotation., these meta annotations are used to define other Annotations.

Java has provide all the annotation in java.lang.annotation

Below are the meta annotations -

1. @Inherited
2. @documented
3. @Target
4. @retention

5.1 @Inherited

By default annotations are not inheritable

Ex –

```
@Persistable
Class Employee {
        String eid;
        String ename;
}
Class Manger extends Employee {
        Void getMgrDetails() {
                s.o.p("eid" +eid);
                s.o.p("ename" +ename);
        }
}
```

In child class we can reuse members of super class. Inheritance is providing reusability feature.
All the super class members are inheritable to child class.

But in case of annotations, it's not possible. **Because by default annotations are not inheritable.**
Even though Employee is @Persistable, persitable nature will not available to Manger.

So to achieve inheritance for annotations, in the declaration of @Persistable annotation @inherited annotated should be used.

```
@ inherited
@Interface Persistable {
}
```

5.2 @ Documented

The purpose of @Documented is to document annotations. By default annotations are not documentable.

Ex –

```
Class Employee {
        String eid;
        String ename;
        Public Employee(String eid){
        }
        Public Employee(String eid, String ename){
        }
        Public void add(){
        }
}
```

Java doc Employee.java will generate html files, which will have all the information about class like variables, methods, etc.

If the same Employee class is annotated with @ Persistable like below, java document tool will not generate documentation for @ Persistable annotation.

```
@ Persistable
Class Employee {
        ------
        ------
}
```

To generate documentation for annotations in Java, Persistable annotation must be annotated with @Documented annotation like below –

```
@Documented
@Interface Persistable {
}
```

To make any annotation a documentable.

5.3 @ Target

@Target annotation can be used to define target elements for an annotation. In Java target elements can be variables, methods, classes.

Ex –

@Target(ElementType val)

ElementType is an enum here –

ElementType.TYPE –can be used for class, AC, Interface
ElementType.FIELD - variable level.
ElementType.METHOD – can be used for method
ElementType.CONSTRUCTOR –can be used for constructor

@Target(ElementType.TYPE , ElementType.FIELD)

5.4 @ Retention

If you want to provide life time for an annotation, then we need to use @Retention..

@Retention annotation makes availabl an annotation to defined level.

Ex –

@Retention (@RetentionPolicy val)

```
@ Retention(@RetentionPolicy.RUNTIME)
@Interface Persistable {
}
```

6 Custom Annotations

Below are the different steps to define Custom annotations

1. Define user defined annotation
2. Utilize user defined annotation in java application
3. Access data from user defined Annotation.

6.1 Define user defined annotations

Ex –

```
Import java.lang.annotation
@Inherited
@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Interface Course {
        String courseId() default “c_111”;
        String cname() default “C programming”;
        Int cost() default “1000”;
}
```

6.2 Utilize user defined annotations

```
@Course (courseId=”j-111”, ccname=”Java”, cost=10000)
Class Student {
        String sid;
        String sname;
        String saddr;
        Public Student (String sid, String sname, String saddr){
                This.sid=sid
                This.ssname= ssname
                This. saddr =saddr
        }
        Public void getStudentDetails(){
        {
                System.out.println(“Student details”);
                System.out.println(sid + sname + saddr);
        }
}
```

6.3 Access data from user defined Annotation

Use reflection api to get Annotation data programmatically.

```
Class Test {
        Public static void main(string[] s){
        {
                Student s = new Student("s1", "name", "Hyd");
                s.getStudentDetails();
                Class c = s.getClass();
                Annotation ann = c.getAnnotation(Course.class);
                Course crs = (Course) ann;
                System.out.println(crs.cid()); - can call members of annotation as method
                System.out.println(crs.ccname());
                System.out.println(crs.cost());
        }
}
```

Program will display student details along with Course details.

To define above annotation as method level annotation, below change is required –

1. Definition has to be changed like below -

```
@Target(ElementType.TYPE, ElementType.METHOD)
@Interface Course {
```

2. Need to apply annotation on a method in student calss.
3. In Test program has be to changed to get the Annotation Object by using method name like below –

```
Method m = c.getMethod ("getMethodDetails");
Annotation ann =  m.getAnnotation(Course.class);
```