Java Annotation

Annotation is a Feature provided by JDK 5.0 version , it can be used to represent metadata in Java applications . such as annotation is attached with class , interface , methods or fields to indicate some additional information which can be used by java compiler and JVM. Java annotation is a alternative way for XML.

জাভা ৫.০ ভার্সন এ এনোটেশন এ আবির্ভূত হয়। এনোটেশন হচ্ছে এক ধরণের ট্যাগ। যা একটি জাভা এপ্লিকেশন এ এক্সট্রা কিছু তথ্য যোগ করার জন্য ব্যবহার করা হয়। একটি জাভা এপ্লিকেশন এ কমেন্ট এর মাধ্যমে ও আমরা অতিরিক্ত তথ্য যোগ করতে পারি কিন্তু প্রব্লেম হচ্ছে , জাভা কমেন্ট এর মাধ্যমে যদি তথ্য যোগ করি তা শুধু জাভা সোর্স ফাইল পর্যন্ত সীমাবদ্ধ।যদি জাভা কমেন্ট ব্যবহার করে তথ্য যোগ করি , তাহলে লেক্সিক্যাল এনালাইজার সেই কমেন্ট মেটাডাটা কম্পাইলাশন এর আগেই রিমুভ করে দেয়। সেই কমেন্ট আমরা জাভা রান টাইম পর্যন্ত ব্যবহার করা যাবে না। এই সমস্যা দূর করার জন্য জাভা এনোটেশন ব্যবহার করা হয়। এই এনোটেশন আমরা চাইলে ক্লাস , মেথডস , ইন্টারফেস , ফিল্ডস এর সাথেও ব্যবহার করতে পারি। জাভা এনোটেশন .ফাইল , .ক্লাস , কম্পাইলেশন পর্যন্ত ব্যবহার করা যায়। যা জাভা কমেন্ট এর চেয়ে ভালো।

Difference between java annotation and XML based system:

Web.XML	Java Annotation

web.xml Code :
 	
<web-app>
 	
<servlet>
<servlet-name>ls</servlet-name>
<servlet-class>LoginServlet</servlet-class>
</servlet>
 	
<servlet-mapping>
<servlet-name>ls</servlet-name>
<url-pattern>/login</url-pattern>
</servlet-mapping>
 	
</web-app>
	Java Annotation Code:
	
	@WebServlet("/login")
	Public class LoginServlet extends HttpServlet{
	::::::::::::::::::::::::::::::
	}

Here 10 line code.	Here only two line code.it is better than XML
JDK 1.4 JDBC 3.0 Servlets 2.5 Struts 1.x JSF 1.x Hibernate 3.2.4 EJBs 2.x Spring 2.x	JDK 5.0 JDBC 4.0 Servlets 3.0 Struts 2.x JSF 2.x Hibernate 3.5 EJBs 3.x Spring 3.x
Annotation are not the complete alternative way for XML tech. annotation use only alternative way for XML based document. But if we needed the application distributed then we have to use different technology when annotation are not able to solve this problem ,when we have to use XML technology. Java annotation processing tool managed java up to JAVA7 Version. Java annotation was removed from JAVA8 Version.	

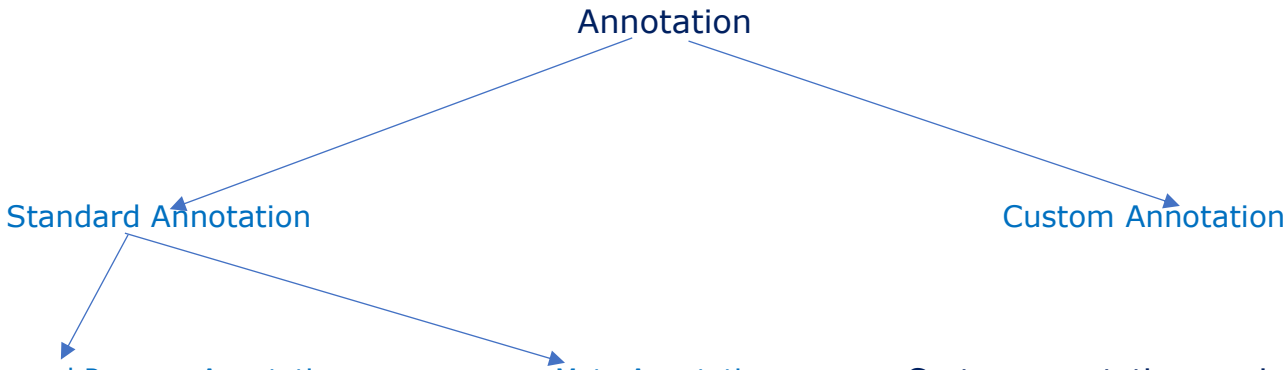
Annotations:

Syntax to declare annotation:

```
@interface Annotation_Name {
        Data_Type Member_Name () [default value];
    }
```

Syntax to use Annotation:

@Annotation_Name (member_name1=value, member_Name2=value2----------)



1. @Override
2. @SuppressWarnings
3. @Depricated
4. @FunctionalInterface[JAVA8]

This Annotation are included in the java.lang package.

1. @Documented
2. @Inherited
3. @Target
4. @Rentention

This Annotation are included in the java.lang.annotation package.

Custom annotation are included in the java.lang.annotation.Annotation package

There are Three Types of Annotations:

1. Marker Annotation
2. Singled-valued Annotation
3. Multi-valued Annotation

Marker Annotation :

```
package java_marker_annotation_example1;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@interface MyMarker {
        // this is a marker annotation
        // marker annotation will be without members
}

public class Marker {

        @MyMarker
        public void myMethod() {
                try {
                    Class<? extends Marker> cls = this.getClass();
                    Method m = cls.getMethod("myMethod");
                    if (m.isAnnotationPresent(MyMarker.class)) {
                        System.out.println("MyMarker is present");
                    } else {
                        System.out.println("MyMarker is not found");
                        }
                } catch (Exception e) {
                        System.out.println(e);
                }
        }

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

Syntax:
@interface MyMarker{}

This annotation name will be user-defined name or java built in annotation name.

Here MyMarker is a marker Annotation. There is no member in marker annotation.

Marker is a java class. And here we will create a method that name is "myMethod".

Before the method ,we can not use @MyMarker then we can not get MyMarker class.

Also if we do not use @Retention(RetentionPoilcy.RUNTIME) Then we can not MyMarker Class. Because to take MyMarker annotation class to runtime used @Retention other wise we will not our suitable output.

```
                Marker mk = new Marker();
                mk.myMethod();

        }
}
```

Output is :
MyMarker is present

Single valued Annotation :

```
package java_single_valued_annotation_example2;

import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@interface SingleValuedAnno {
        int value();
        /* this variable name must be value */
}

public class Single {
        /* annotate a method using a single member annotation
*/
        @SingleValuedAnno(10)
        public void mysingleanno() {
           try {
                Class<? extends Single> cls = this.getClass();
                Method m = cls.getMethod("mysingleanno");
                SingleValuedAnno siganno =
m.getAnnotation(SingleValuedAnno.class);
                System.out.println(siganno.value());
                } catch (Exception e) {
                        System.out.println(e);
                }
        }

        public static void main(String[] args) {
                Single sg = new Single();
                sg.mysingleanno();
        }
}
```

Syntax:

```
@interface MyMarker{
    int value();
}
```

I am use user-defined annotation.

This is single valued annotation . this annotation we have use only one single value.

Here we pass the value with single member variable.

Output :
10

Multivalued valued Annotation :

```
package java_smultivalued_valued_annotation_example3;

import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@interface MultiValuedAnnotation {
        int id();

        String name();

        String address();
}


public class MultiValued {
```

If we use multivalued annotation then we can add more than one member.

Same description of single valued annotation . only difference it is multivalued.

This example is a custom annotation example.

```
@MultiValuedAnnotation(id = 101, name = "Wornoz",
address = "Dhaka")
public void MultiDisplay() {
	try {
	Class<? extends MultiValued> cls = this.getClass();
	Method m = cls.getMethod("MultiDisplay");
	MultiValuedAnnotation mva =
m.getAnnotation(MultiValuedAnnotation.class);
	System.out.println("ID is:" + mva.id() + "\nName
is:" + mva.name() + "\nAddress is :" + mva.address());
		} catch (Exception e) {
			System.out.println(e);
		}
	}

	public static void main(String[] args) {
		MultiValued mv = new MultiValued();
		mv.MultiDisplay();

	}
}
```

Output is :

```
ID is:101
Name is:Wornoz
Address is :Dhaka
```

@Override Annotation:

```
package java_override_annotation_example4;

class A {
	public void getDisplay() {
		System.out.println("Super class A");
	}
}

class B extends A{
	@Override
	public void getDisplay() {
		System.out.println("Sub class B");
	}
}

public class TestClass {
	public static void main(String[] args) {
		A a=new B();
		a.getDisplay();
	}
}

Ouput is :
Sub class B
```

Here A is a super class . and have one method.

B is a sub class .and we extends the super class A. So A class all the features are available in sub class B.

if we want to perform method overriding then we have to provide a method in subclass with the same prototype of super class method.

In case if we any mistake provide the same super class method at sub class method then we can not get any error. but we will get super Class A method output.

if we want to get an error about to describe failure case of method overriding from compiler then we have to use @Override

If we compile the above programme then compiler will rise an error like "method does not override or implement a method from a SuperType".

@Depricated Annotation:

If we want to update our old system in a new way without deleting it, we can do it using deprecated annotations to the old system.

```java
package java_depricated_annotation_example5;

class Employee {
	@Deprecated
	public void gen_Salary(int basic, double hq) {
		double total = basic + hq;
		System.out.println("Salary is claculated , basic amount and hq:" + total);
	}

	public void update_gen_salary(int basic, double hq, int ta, double pf) {
		double total = basic + hq + ta + pf;
		System.out.println("Salary is updated,basic,hq,ta,pf:" + total);
	}
}

public class TestClass {

	public static void main(String[] args) {
		Employee em = new Employee();
		em.gen_Salary(102, 302);
		em.update_gen_salary(303, 45, 304, 209);
	}
}
```

If we compile the above code then compiler will provide the following deprecation message "Note:java uses or overrides a deprecated API"

@SuppressWarnings Annotation:

```java
package
java_suppresswarnings_annotation_example6;

import java.util.*;

class Bank {
	@SuppressWarnings("unchecked")
	public ArrayList listCustomer() {
		ArrayList al = new ArrayList();
		al.add("Wornoz");
		al.add("Qurishe");
		al.add(1993);
		return al;
	}
}

public class TestClass {

	public static void main(String[] args) {
		Bank b = new Bank();
		List l = b.listCustomer();
		System.out.println(l);
	}
}
```

*here we declare ArrayList so we can add
* any type of value which are unsafed.
* when we will compile , we will get a some warnings message from compiler.
* which we can not get any warnings from compiler then we will do use @SuppressWarnings

Warning message are:
java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

When we use @SuppressWarnings("unchecked")before the method warnings message are remove.

Output is :

```
[Wornoz, Qurishe, 1993]
```

@FunctionalInterfaces Annotation:

It is new annotation which is provided java8 version .

```
package java_functional_interface_annotation_example7;

@FunctionalInterface
interface Loan {
	void getLoan();
}

class GoldLoan implements Loan {
	public void getLoan() {
		System.out.println("GoldLoan");
	}
}

public class TestClass {

	public static void main(String[] args) {
		Loan l = new GoldLoan();
		l.getLoan();
	}
}
```

In java if we provided any interface without abstract method that is a "marker interface"

In java if we provide any interface with one abstract method that is a "Functional Interface"

To make any interface as functional interface and to allow exactly one abstract method in any interface then we have to use "@FunctionalInterface" annotation.

Output:
GoldLoan

@inherited Annotation:

```
package java_inhereted_annotation_example8;

import java.lang.annotation.*;

@Target(ElementType.TYPE)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface InheritedAnnotationType {
	String value1() default "this is inherited interface";
}
//////////////////////////////////////////////
package java_inhereted_annotation_example8;

import java.lang.annotation.*;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface UninheritedAnnotationType {
	String value1() default "this is uninherited interface";
}
//////////////////////////////////////////////
package
java_inhereted_annotation_example8;

@UninheritedAnnotationType
public class A {

}
//////////////////////////////////////
package java_inhereted_annotation_example8;

@InheritedAnnotationType()
public class B extends A{

}
//////////////////////////////////////
package java_inhereted_annotation_example8;
```

In general all the annotations are not inheritable by default.

If we want to make/prepare any annotation as Inheritable annotation then we have to declare that annotation as Inheritable annotation then we have to declare that annotation with

@Inherited annotation.

Ex-1:

```
@interface Persistable { }

@Persistable
class Employee { }

class Manager extends Employee{ }
```

in the above example,only Employee class objects are Persistable i.e eligible to store in database.

Ex-2:

```
@Inherited
@interface Persistable { }

@Persistable
class Employee { }

class Manager extends Employee{ }
```

In the above example,both Employee class objects and manager class objects are Persistable i.e eligible to store in database.

```
public class C extends B { }


////////////////////////////////////
/
package java_inhereted_annotation_example8;

public class TestClass {

public static void main(String[] args) {
System.out.println(new
A().getClass().getAnnotation(InheritedAnnotationT
ype.class));
System.out.println(new
B().getClass().getAnnotation(InheritedAnnotationT
ype.class));
System.out.println(new
C().getClass().getAnnotation(InheritedAnnotationT
ype.class));
                System.out.println("----------------
---------------");

System.out.println(new
A().getClass().getAnnotation(UninheritedAnnotatio
nType.class));
System.out.println(new
B().getClass().getAnnotation(UninheritedAnnotatio
nType.class));
System.out.println(new
C().getClass().getAnnotation(UninheritedAnnotatio
nType.class));
        }
}
```

Output is:

```
null
@java_inhereted_annotation_example8.InheritedAnnotationTy
pe(value1="this is inherited interface")
@java_inhereted_annotation_example8.InheritedAnnotationTy
pe(value1="this is inherited interface")
---------------------------------
@java_inhereted_annotation_example8.UninheritedAnnotation
Type(value1="this is uninherited interface")
null
null
```

@Documented Annotation:

If we want to any java program to make html format then we will be use @Documented annotation.

@Target:

The main intention of this annotation is to define a list of target elements to which we are applying the respective annotation.

Example:

@Target(ElementType.TYPE,ElementType.FIELD,ElementType.ME THOD)

Custom Annotations:

	Custom annotations are defined by the developers as per their application requirements. To use custom annotations in java applications, we have to use the following steps.

```java
package java_custom_annotation_example9;

import java.lang.annotation.*;

/*--
 * Declare user defined Annotation
 * --*/
@Inherited
@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@interface Bank {
        String name() default "IBBL";

        String branch() default "Savar-Branch";

        String phone() default "09580347567";
    }
```

```java
package java_custom_annotation_example9;

import java.lang.reflect.*;

/*--
 * Utilize User defined Annotations in
 * java Application
 * --*/

public class Account {
        String accNo;
        String accName;
        String accType;

        public Account(String accNo, String accName, String accType) {
                this.accNo = accNo;
                this.accName = accName;
                this.accType = accType;
        }

        public void getAccountDetails() {
                System.out.println("-------Account Details-------");
                System.out.println("-----------------------------");
                System.out.println("Account Number : " + accNo);
                System.out.println("Account Name   : " + accName);
                System.out.println("Account Type   : " + accType);
                System.out.println("---End of Account Details---");
        }

        /*step-1-@Bank we can not add any value so default value is shown-*/
        /*step-2-@Bank(name="Qurishe",branch="Dhaka",phone="09897890")-*/
        /*-if we use step-2 then we will get new inserted value -*/

        @Bank()
        public void getBankDetails() {
                try {
                        Class<? extends Account> c = this.getClass();
                        Method m = c.getMethod("getBankDetails");
                        Bank bn = m.getAnnotation(Bank.class);
```

```
                System.out.println("--------Bank Details--------");
                System.out.println("----------------------------");
                System.out.println("Bank Name     : " + bn.name());
                System.out.println("Branch Name   : " + bn.branch());
                System.out.println("Phone         : " + bn.phone());

            } catch (Exception e) {
                System.out.println(e);
            }
        }
}
```

```
package java_custom_annotation_example9;

import java.lang.annotation.*;
/*--
 * Access the data from User-Defined
 * Annotation in TestClass Application
 * --*/
public class TestClass {

        public static void main(String[] args) {
                Account acc=new Account("IBBL-0011","Qurishe","Student-Account");
                acc.getAccountDetails();
                System.out.println();

                acc.getBankDetails();

                /*-this is another system and this system is used for class label annotation
                 * as a alternative system---
                 * we use getBankDetails() Method
                 * and we call here getBankDetails() Method
                 * -*/

                /*----
                Class c=acc.getClass();
                Annotation ann=c.getAnnotation(Bank.class);

                Bank b=(Bank)ann;
                System.out.println("----Bank Details----");
                System.out.println("--------------------");
                System.out.println("Bank Name   : "+b.name());
                System.out.println("Branch Name : "+b.branch());
                System.out.println("Phone       : "+b.phone());
                ----*/
        }
}
```

Output is :

```
-------Account Details-------
---------------------------
Account Number : IBBL-0011
Account Name   : Qurishe
Account Type   : Student-Account
---End of Account Details---

--------Bank Details--------
----------------------------
Bank Name     : IBBL
Branch Name   : Savar-Branch
Phone         : 09580347567
```