Java Annotations

provides information about the code

What is Annotation

- Before java annotations, program metadata was available through java comments or by Javadoc but annotation offers more than that.
- Annotations metadata can be available at runtime too and annotation parsers can use it to determine the process flow.
- Annotation methods can’t have parameters.
- Annotation methods return types are limited to primitives, String, Enums, Annotation or array of these.
- Java Annotation methods can have default values.
- Annotations can have meta annotations attached to them. Meta annotations are used to provide information about the annotation.

Sample Annotation

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


// Pre-defined Annotation @Override, @SuppressWarnings, @Depricated, @Target, @Retention, @Inherited,
//@Documented
// Annotations can inherited
@Target(ElementType.MODULE) // ElementType is an attribute of Target Annotation
@Retention(RetentionPolicy.SOURCE)
public @interface MySampleAnnotation {
}
```

Types of Annotation

- Java annotations has no direct affect on code they annotated.
- Built-in Java annotations
- How to write Custom Annotation
- Annotations usage and how to parse annotations using Reflection API.
- Java 1.5 introduced annotations and now it’s heavily used in Java EE frameworks like Hibernate, Jersey, and Spring.
- Java Annotation is metadata about the program embedded in the program itself.
- It can be parsed by the annotation parsing tool or by the compiler. We can also specify annotation availability to either compile time only or till runtime

Marker Annotation

```java
package annotation_demo;

// Marker annotations are always without any method
public @interface MarkerAnnotation {

}
```

Single Value Annotation

```
package annotation_demo;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface SingleValueAnnotation {
String message() default "No Msg";
}
```

Multi-Value Annotation

```
package annotation_demo;

// Wrapper classes are not allowed in method name
public @interface MultiValueAnnotation {
String message() default "No Message";
int length() default 0;
boolean status() default false;
float area() default 0.0f;
double pi() default 3.14;
}
```

Testing Annotation

```
package annotation_demo;

public class MainApp {
@SuppressWarnings("deprecation")
public static void main(String[] args) {
One one=new One();
one.methodInOne();  // this method will still valid
one.methodInOneDepricated(); // This is not recommended to be used in furture
}
// Type of Annotations
// 1. Marker Annotation - without any method
}
```

How to use Annotation

```
// Annotation are used to create meta data from JVM to process faster
public class MainApp {
@SuppressWarnings({ "rawtypes", "unused" }) // Annotations are the instructions to JVM
private static List lst;

@SuppressWarnings({ "rawtypes", "unchecked" })
public static void main(String[] args) {
lst = new ArrayList();
lst.add("ABC");
lst.add("KBC");
System.out.println(lst);}}
```

```
@Documented
@Target(ElementType.METHOD)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface MethodInfo {
String author() default "Tarkeshwar Barua";
String date();
int revision() default 1;
String comments();

}
```

- **@Documented –** Elements using this annotation should be documented by javadoc and similar tools. This type should be used to annotate the declarations of types whose annotations affect the use of annotated elements by their clients. If a type declaration is annotated with Documented, its annotations become part of the public API of the annotated elements.
- **@Target –** indicates the kinds of program element to which an annotation type is applicable. Some possible values are TYPE, METHOD, CONSTRUCTOR, FIELD etc. If Target meta-annotation is not present, then annotation can be used on any program element.
- **@Inherited –** indicates that an annotation type is automatically inherited. If user queries the annotation type on a class declaration, and the class declaration has no annotation for this type, then the class's superclass will automatically be queried for the annotation type. This process will be repeated until an annotation for this type is found, or the top of the class hierarchy (Object) is reached.
- **@Retention –** indicates how long annotations with the annotated type are to be retained. It takes RetentionPolicy argument whose Possible values are SOURCE, CLASS and RUNTIME
- **@Repeatable –** used to indicate that the annotation type whose declaration it annotates is repeatable.

Method Depricated Annotation

```
package annotation_demo;

public class One {
public void methodInOne() {
System.out.println("Hello i am from Method One");
}
@Deprecated
public void methodInOneDepricated() {
System.out.println("Hello i am from Method One which is depricated");
}
}
```

- **@Override –** When we want to override a method of Superclass, we should use this annotation to inform compiler that we are overriding a method. So when superclass method is removed or changed, compiler will show error message. Learn why we should always use java override annotation while overriding a method.
- **@Deprecated –** when we want the compiler to know that a method is deprecated, we should use this annotation. Java recommends that in javadoc, we should provide information for why this method is deprecated and what is the alternative to use.
- **@SuppressWarnings –** This is just to tell compiler to ignore specific warnings they produce, for example using raw types in java generics. It's retention policy is SOURCE and it gets discarded by compiler.
- **@FunctionalInterface –** This annotation was introduced in Java 8 to indicate that the interface is intended to be a functional interface.
- **@SafeVarargs –** A programmer assertion that the body of the annotated method or constructor does not perform potentially unsafe operations on its varargs parameter.

```
@Override
@MethodInfo(author = "Tarkeshwar", comments = "Main method", date = "Nov 17 2021", revision = 1)
public String toString() {
return "Overriden toString method";
}

@Deprecated
@MethodInfo(comments = "deprecated method", date = "Nov 17 2021")
public static void oldMethod() {
System.out.println("old method, don't use it.");
}

@SuppressWarnings({ "unchecked", "deprecation" })
@MethodInfo(author = "Tarkeshwar", comments = "Main method", date = "Nov 17 2021", revision =
10)
public static void genericsTest() throws FileNotFoundException {
List l = new ArrayList();
l.add("abc");
oldMethod();
}
```

```
try {
for (Method method : AnnotationParsing.class.getClassLoader().loadClass(("springCore.MainApp"))
.getMethods()) {
// checks if MethodInfo annotation is present for the method
if (method.isAnnotationPresent(springCore.MethodInfo.class)) {
try {
// iterates all the annotations available in the method
for (Annotation anno : method.getDeclaredAnnotations()) {
System.out.println("Annotation in Method '" + method + "' : " + anno);
}
MethodInfo methodAnno = method.getAnnotation(MethodInfo.class);
if (methodAnno.revision() == 1) {
System.out.println("Method with revision no 1 = " + method);
}

} catch (Throwable ex) {
ex.printStackTrace();
}
}
}
} catch (SecurityException |

ClassNotFoundException e) {
e.printStackTrace();
}
```

Annotation Inheritance

```
package annotation_demo;


import java.lang.annotation.Inherited;
// ParentInterface is an annotation
@Inherited   // after applying inherited annotation the same annotation will be available in child anoota
@interface ParentAnnotation{
}
@interface ChildAnnotation {
}
class MySuperClass{
}
public class AnnotationInhertance extends MySuperClass {
}
```

MainApp.java

```
@MarkerAnnotation
@MultiValueAnnotation(area = 3.555f, length = 3, message = "Messafe from Multi value Annotation", pi = 4.55, status=true)
public class UsingAnnotation {


public static void main(String[] args) {
System.out.println("Welcome to Annotations");
UsingAnnotation usingAnnotation=new UsingAnnotation();
try {
Method method=usingAnnotation.getClass().getMethod("addition");
SingleValueAnnotation annotationValue=method.getAnnotation(SingleValueAnnotation.class);
System.out.println(annotationValue.message());
} catch (NoSuchMethodException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (SecurityException e) {
// TODO Auto-generated catch block
e.printStackTrace();}}
//@SingleValueAnnotation
@SingleValueAnnotation(message = "I am using in the main App")
public int addition() {
return 0;}}
```