Annotations in Java

An **annotation in Java** is a kind of note (or documentation) in the application source code to instruct the Java compiler to do something.

When the Java compiler sees annotation in a particular source code, it knows that it needs to do something special with the code that follows.

We can annotate Java packages, classes, interfaces, constructors, methods, fields, local variables, and parameters.

Standard Built-in Annotations (Predefined annotations) in Java

Java 5 contains three general-purpose commonly standard built-in annotations, defined in **java.lang.annotation** package that is used to give instructions to the Java compiler. They are as follows:

1. **@Deprecated**
2. **@Override**
3. **@SuppressWarnings**

Later on, Java 7 and Java 8 added **SafeVarargs** and **FunctionalInterface** annotations in the **java.lang.annotation** package.

Deprecated Annotation

The @Deprecated annotation is a type of marker annotation that can be used to mark a class, method, field, or other programming features as deprecated, meaning it should be no longer in use.

When it happens, we need to update their code and stop using that particular class, method, field, or other programming features.

```
public class DeprecatedTest
{
```

```
public void x() {
       System.out.println("Hello x");
}
@Deprecated
 public void y()
 {
      System.out.println("Hello y");
 }
public static void main(String[] args) {
DeprecatedTest obj = new DeprecatedTest();
   obj.x();
   obj.y();
 }
}
```

@Override Annotation

The @Override annotation is a type of marker annotation that is used above a method to indicate to the Java compiler that the subclass method is overriding the superclass method.

The @Override annotation is not essential to override a method in the superclass. In case someone changed the name of the overridden method in the superclass, or did a spelling mistake then the subclass method would no longer override it.

```
public class A
{
void m1() {
       System.out.println("A-m1");
 }
}
public class B extends A
{
@Override
 void m1(int a)
 {
      System.out.println("B-m1");
 }
public static void main(String[] args) {
B b = new B();
   b.m1(20);
 }
}
```

@Suppress Warnings Annotation

The @SuppressWarnings annotation is used to suppress warnings generated by the compiler. In

other words, this annotation is used to turn off inappropriate compiler warnings.

This @SuppressWarnings annotation can be applied with types, constructors, methods, fields, parameters, and local variables.

```
import java.util.ArrayList;
public class SupressWarningsTest
{
@SuppressWarnings({ "unchecked", "rawtypes" })
public static void main(String[] args) {
ArrayList arList = new ArrayList();
  arList.add("Orange");
  arList.add("Pink");
  arList.add("Red");
  arList.add("Green");
  arList.add("Blue");

 for(Object obj : arList) {
  System.out.println(obj);
   }
 }
}
```

@FunctionalInterface Annotation

A **functional interface in java** is an interface in which we can declare only one abstract method.

The @FunctionalInterface is a marker annotation type that is used to verify compiler that interface has one and only abstract method.

If an interface is not annotated with this annotation is not functional interface, the compiler will generate a compile-time error. It cannot be used with classes, annotation types, and enums.

```
@FunctionalInterface
public interface Person {
         void play(); // Only one abstract method.
}
```

```
@FunctionalInterface
public interface Student
{
    void play();
```

```
    void read();
}
```

@SafeVarargs Annotation

When we use @SafeVarargs annotation with method or constructor, method or constructor does not perform potentially unsafe or harmful operations on its varargs parameters.

When this annotation is used with method or constructor, unchecked warnings relating to varargs usage are suppressed.

Standard Meta Annotations in Java

Annotations that are used to annotate annotations are called **meta annotations in java**. In other simple words, annotations that are applied to other annotations are called meta-annotations.

There are Six types of meta-annotations that can be used to annotate annotations.

1. Documented
2. Inherited
3. Retention
4. Target
5. Repeatable
6. Native

@Documented Annotation

The @Documented is a type of marker meta annotation that is used to annotate the declaration of an annotation type.

```
package annotationProgram;
public class Test
{
  @Override
  public String toString()
```

```
  {
        return "Test";
   }
}
```

@Inherited Annotation

The @Inherited annotation is a type of meta marker annotation that is used to inherit an instance of annotation type.

```
public @interface Test1 {
     int id();
}
@Inherited
public @interface Test2 {
    int id();
}
```

```
@Test1(id = 100)
@Test2(id = 200)
public class A {
        // Code for class A goes here.
}
// class B inherits Test2(id = 200) annotation from class A.
public class B extends A {
       // Code for class B goes here.
}

/**
In the above snippet of code, class B inherits @Test2(id = 200) annotation from class
 A because Test2 annotation type is annotated with an Inherited meta-annotation.

Class B does not inherit @Test1(id = 100) annotation because Test1 annotation type is
 not annotated with an Inherited meta-annotation.

*/
```

@Retation Annotation

The @Retation annotation is a meta-annotation type that is used to specify how an instance of
annotation type should be retained by Java. It is also known as retention policy of an annotation type.

An annotation can be retained at three levels in Java that are as follows:

- Source code only
- Class file only (default)
- Class file and runtime (simply known as runtime)

```
@Retention(value = SOURCE)
public @interface SuppressWarnings
```

@Target Annotation

The @Target annotation is a type of meta annotation that is used to annotate an annotation type to specify the context in which annotation type can be used.

It has only one element named value. The value of Target can be one of the members of the java.lang.annotation.ElementType enum:

a) **ANNOTATION_TYPE**: The annotation can be used to annotate another annotation type declaration.

b) **CONSTRUCTOR**: The annotation can be used to annotate constructor declaration.

c) **FIELD**: The annotation can be used to annotate fields declaration and enum constants.

d) **LOCAL_VARIABLE**: The annotation can be used to annotate local variable declaration.

e) **METHOD**: The annotation can be used to annotate method declaration.

f) **PACKAGE**: The annotation can be used to annotate package declaration.

g) **PARAMETER**: It can be used to annotate parameter declaration.

h) **TYPE**: The annotation can be used to annotate annotation type, class, interface, and enum declarations.

```
@Target(value = METHOD)
// Declaration of multiple values in the Target annotation.
@Target(value = {TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE})
```

@Repeatable Annotation

Most annotations can be applied only once to a class, method, field, etc. In some cases, we need to apply an annotation more than once.

```
// A repeatable annotation type that contains B as Containing annotation type.
package annotationsProgram;
import java.lang.annotation.Repeatable;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
@Repeatable(B.class)
public @interface A
{
    String date();
}
// A Containing annotation type for A Repeatable annotation type.
package annotationsProgram;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
public @interface B {
  A[ ] value();
}
```

We can use A annotation to log change history for the Test class, as shown below:

```
package annotationsProgram;
@A(date="28/09/2021")
@A(date="08/10/2021")
public class Test {
public static void process()
{
    // code goes here.
 }
}
```

@Native Annotation

The @Native annotation is a type of meta-annotation that is used to annotate fields. It is a marker annotation that indicates that the annotated field may be referenced from the native code.