Annotation

Annotation

- Annotation – Izoh deb tarjima qilinadi

Annotation

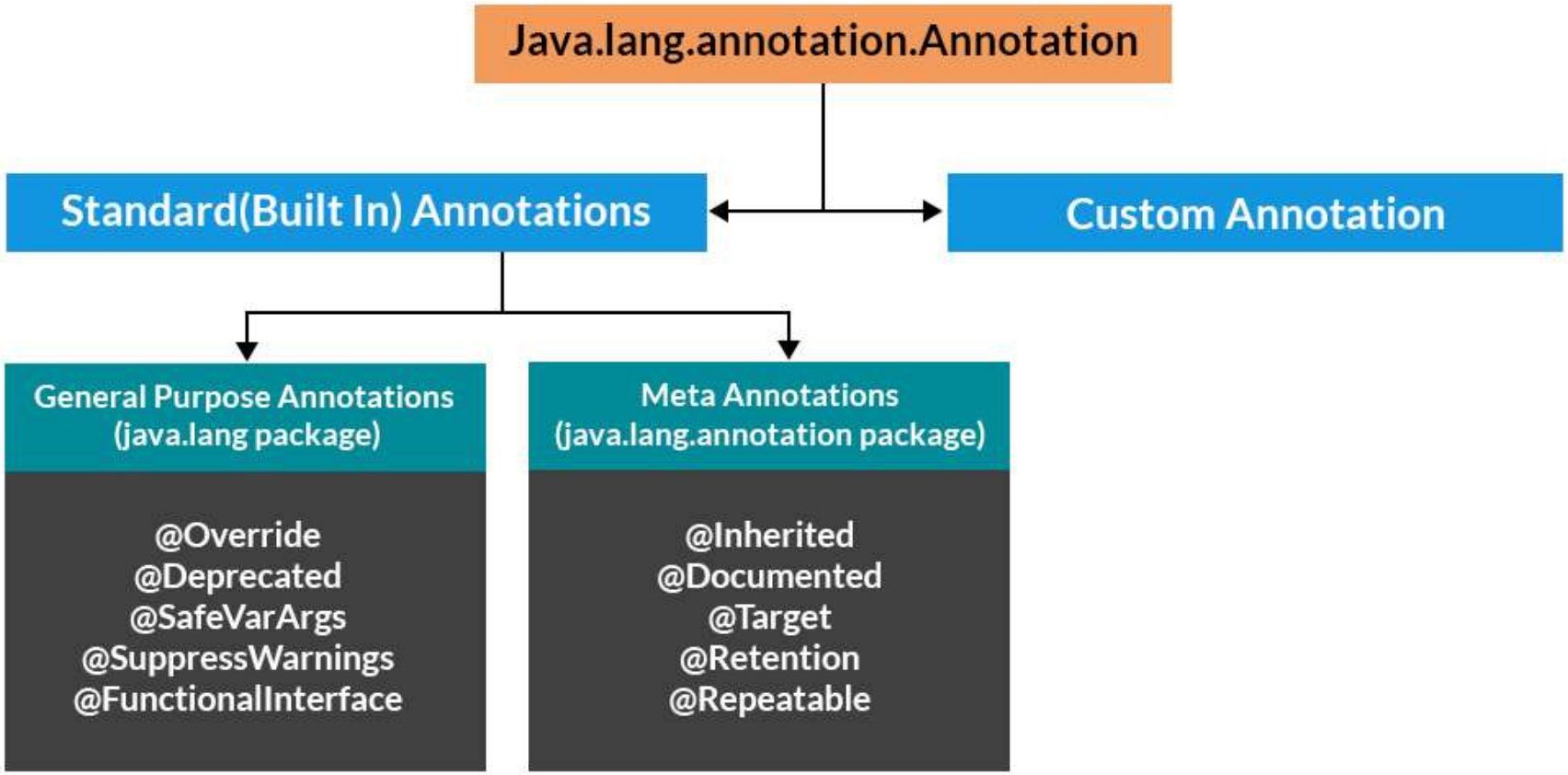
- Annotations are used to provide supplemental information about a program.
  - Annotations start with ‘@’.
  - Annotations do not change the action of a compiled program.
  - Annotations help to associate metadata (information) to the program elements i.e. instance variables, constructors, methods, classes, etc
  - Annotations basically are used to provide additional information, so could be an alternative to XML and Java marker interfaces.
- Annotatsiyalar dastur xaqida qo’shimcha ma’lumot berish uchun ishlatiladi.
  - Annotatsiyalar ‘@’ bilan boshlanadi.
  - Annotatsiyalar kompliyatsiya bo’lgan kodlarni o’zgartirmaydi.
  - Izohlar metadata (ma'lumot) ni dastur elementlari, ya'ni misol o'zgaruvchilari, konstruktorlar, usullar, sinflar va boshqalar bilan bog'lashga yordam beradi.
  - Annotatsiyalar asosan qo'shimcha ma'lumot berish uchun ishlatiladi, shuning uchun XML va Java marker interfeyslariga muqobil bo'lishi mumkin.

Annotation

- In Java, annotations are used to attach meta-data to a program element such as a class, method, instances, etc.
- Some annotations are used to annotate other annotations. These types of annotations are known as meta-annotations.

- Java-da annotatsiyalar sinf, usul, misollar va boshqalar kabi dastur elementiga meta-ma'lumotlarni biriktirish uchun ishlatiladi.
- Bazi annotatsiyalar boshqa annotatsiyalarni yaratish uchun ishlatilari. Bunday annotatsiyalar meta-annotation lar deyiladi.

Annotation


Java.lang.annotation.Annotation
Standard(Built In) Annotations
Custom Annotation
General Purpose Annotations
(java.lang package)
@Override
@Deprecated
@SafeVarArgs
@SuppressWarnings
@FunctionalInterface
Meta Annotations
(java.lang.annotation package)
@Inherited
@Documented
@Target
@Retention
@Repeatable

Categories of Annotations

- There are broadly 5 categories of annotations as listed:
  - Marker Annotations - marker
  - Single value Annotations – yagona qiymatli
  - Full Annotations – to’liq
  - Type Annotations - tip
  - Repeating Annotations - takrorlash

- Annotatsiyalarning umumiy 5 toifasi mavjud:

Category 1: Marker Annotations

- The only purpose is to mark a declaration. These annotations contain no members and do not consist of any data.
- **@Override** , @Deprecated are an example of Marker Annotation.
- Oddigina qilin nimadirni anglatish uchun ishlatilari. Bu annotatsiyalar hech narsani o’zgartirmaydi va hech narsani o’zgartirmaydi.

Category 2: Single value Annotations

- These annotations contain only one member and allow a shorthand form of specifying the value of the member. We only need to specify the value for that member when the annotation is applied and don’t need to specify the name of the member. However, in order to use this shorthand, the name of the member must be a value.

- @TestAnnotation(“testing”);

- Bu tipli annotatsiyada bitta o’zgaruvchisi bo’lib, u o’zgaruvchiga qiymat berishning qisqa usuli sifatida ishlatiladi.

Category 3: Full Annotations

- These annotations consist of multiple data members, names, values, pairs.
- @TestAnnotation(owner=”Rahul”, value=”Class Geeks”)

- Bu tipli annotatsiya lar birnechta o’zgaruvchilar, nomlar, qiymatlar juftligidan tashkil topgan.

Category 4: Type Annotations

- These annotations can be applied to any place where a type is being used. For example, we can annotate the return type of a method. We can declare these annotations with @Target annotation.

- Bu tipdagi annotatsiyalar har qanday joyda ishlatilishi mumkin. Masalan method ning return tipi sifatida ishlatsak bo’ladi. Biz ularni @Target annotatsiyasi orqali yaratishimiz mumkin.

```
// Using target annotation to annotate a type
@Target(ElementType.TYPE_USE)

// Declaring a simple type annotation
@interface TypeAnnoDemo{}

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args) {

        // Annotating the type of a string
        @TypeAnnoDemo String string = "I am annotated with a type annotation";
        System.out.println(string);
        abc();
    }

    // Annotating return type of a function
    static @TypeAnnoDemo int abc() {

        System.out.println("This function's  return type is annotated");

        return 0;
    }
}
```

Category 5: Repeating Annotations

- These are the annotations that can be applied to a single item more than once. For an annotation to be repeatable it must be annotated with the **@Repeatable** annotation, which is defined in the **java.lang.annotation** package.

- Bu tipdagi annotatsiyalar bitta itemga bir nechamarta ishlatish mumkin. Annotatsiyalar repeateable bo’lishi uchun ular @Repeatable annotatsiyasi bilan belgilangan bo’lishi kerak.

Category 5: Repeating Annotations Examole

```
@Words(word = "First", value = 1)
@Words(word = "Second", value = 2)
public static void newMethod(){
        ....
}
```

Predefined/ Standard Annotations

- Java popularly defines seven built-in annotations as we have seen up in the hierarchy diagram.
- Four are imported from java.lang.annotation: **@Retention**, **@Documented**, **@Target**, and **@Inherited**.
- Three are included in java.lang: **@Deprecated**, **@Override** and **@SuppressWarnings**

Annotation 1: @Deprecated

- It is a marker annotation. It indicates that a declaration is obsolete and has been replaced by a newer form.

```
@Deprecated
  public void Display()
  {
     System.out.println("Deprecatedtest display()");
  }
```

Annotation 2: @Override

- It is a marker annotation that can be used only on methods. A method annotated with **@Override** must override a method from a superclass.

```
class Base
{
    public void Display()
    {
       System.out.println("Base display()");
    }

}
 // Extending above class
class Derived extends Base
{
    @Override
    public void Display()
    {
       System.out.println("Derived display()");
    }
}
```

Annotation 3: @SuppressWarnings

- Use of @SuppressWarnings is to suppress or ignore warnings coming from the compiler, i.e., the compiler will ignore warnings if any for that piece of code.
- @SuppressWarning annotatsiyasi kompiler tomonidan kelayotgan ogohlantirishni bosti bosti qilish yoki e’tibor ga olmaslik uchun ishlatiladi.

```
@SuppressWarnings("unchecked")
  public class Calculator {
        }
```

@SuppressWarnings - 1

- - Here, it will ignore all unchecked warnings coming from that class. (All methods, variables, constructors).

```
@SuppressWarnings("unchecked")
  public class Calculator {
        }
```

- - It will stop warning from that function only, and not from other functions of Calculator class.

```
public class Calculator {
   @SuppressWarnings("unchecked")
      public int sum(x,y) {
        .
      }
  }
```

- We can use it on top of the code (inside method)

@SuppressWarnings - 2

- This annotation allows us to say which kinds of warnings to ignore. While warning types can vary by compiler vendor, **the two most common are *deprecation* and *unchecked***.

- ***deprecation*** **-** tells the compiler to ignore when we're using a deprecated method or type.
- ***unchecked*** **-** tells the compiler to ignore when we're using raw types (to suppress warnings relative to unchecked operations)
- *null* - potential missing or redundant null check

@SuppressWarnings - 3

```
public class Machine {
    private List versions;

    @SuppressWarnings("unchecked")
    // or
    @SuppressWarnings({"unchecked"})
    public void addVersion(String version) {
        versions.add(version);
    }
}
```

- **we can suppress the warning associated with our raw type usage:**

SuppressWarnings - 2

- This annotation is dangerous because a warning is something potentially wrong with the code. So if we're getting any warning, the first approach should be resolving those errors.

Annotation 4: @Documented

- It is a marker interface that tells a tool that an annotation is to be documented. Annotations are not included in ‘Javadoc’ comments. The use of @Documented annotation in the code enables tools like Javadoc to process it and include the annotation type information in the generated document.

Annotation 5: @Target

- It is designed to be used only as an annotation to another annotation. **@Target** takes one argument, which must be constant from the **ElementType** enumeration. This argument specifies the type of declarations to which the annotation can be applied.

- Bu annotatsiya boshqa annotatsiyalarni ishlatiladi. @Target annotatsiyasi bir yoki undan ko’p bo’lgan **ElementType** konstantalarini qabul qiladi. Bu annotatsiya hozirgi annotatsiya nimalarga ishlatish mumkin ekanligini elon qilish uchun ishlatiladi.

@Target 1

Target Constant	Annotations Can Be Applied To
ANNOTATION_TYPE	Another annotation
CONSTRUCTOR	Constructor
FIELD	Field
LOCAL_VARIABLE	Local variable
METHOD	Method
PACKAGE	Package
PARAMETER	Parameter
TYPE	Class, Interface, or enumeration

- We can specify one or more of these values in a **@Target**annotation. To specify multiple values, we must specify them within a braces-delimited list.

@Target 2

- For example:
- To specify that an annotation applies only to fields and local variables, you can use this

  @Target annotation: @Target({ElementType.FIELD, ElementType.LOCAL_VARIABLE})

- Masalan: Class o’zgaruvchisiga va local o’zgaruvchiga ishlatish mumkin bo’lgan annotatsiyani bildirish uchun quyidagilardan foydalaning.

Annotation 6: @Inherited 1

- @Inherited is a marker annotation that can be used only on annotation declaration.
- It affects only annotations that will be used on class declarations.
- **@Inherited** causes the annotation for a superclass to be inherited by a subclass.
- Therefore, when a request for a specific annotation is made to the subclass, if that annotation is not present in the subclass, then its superclass is checked. If that annotation is present in the superclass, and if it is annotated with **@Inherited,** then that annotation will be returned.
- The @Inherited annotation is used or annotated to the annotation

Annotation 6: @Inherited 2

- Java, by default, does not allow the custom annotations to be inherited.
- @inherited is a type of meta-annotation used to annotate custom annotations so that the subclass can inherit those custom annotations.
- Biz bitta class ga custom annotatsiya ishlatsak va shu class ning subclasslari parent classga ishlatilgan annotasiyani qiymatlarini extends ola olmaydi.
- Agar biz Custome annotation ga @inherited ni ishlatsak va shu annotatsiyani ishlatgan class larning subclasslari shu Custome annotatsiyani qiymatlarini ishlatishi mumkin.

Annotation 6: @Inherited exmaple

```
// Creating our single valued custom annotation
// with @inherited annotation
@Inherited
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@interface CustomAnnotation
{
    String value() default "GFG";
}


@CustomAnnotation(value = "Sky is limitless")
class Super {
  // Parent
}
```

Annotation 6: @Inherited exmaple

```
public class InheritedAnnotationDemo extends Super {

    public static void main(String[] arg) throws Exception
    {
        System.out.println(
            new InheritedAnnotationDemo()
                .getClass()
                .getAnnotation(CustomAnnotation.class));

        System.out.println(
            new Super().getClass().getAnnotation(
                CustomAnnotation.class));

    }

}
```

@Retention Annotations

- Retention - ushlab turish, saqlash deb tarjima qilinadi.
- @Retention is also a meta-annotation that comes with some retention policies.
- These retention policies determine at which point an annotation is discarded
- There are three types of retention policies: SOURCE, CLASS, and RUNTIME.

- @Retention annotatsiyasi meta-annotation hisoblanadi va unda to’xtatish xususiyatlari bor.
- Ushbu saqlash qoidalari qaysi nuqtada annotatsiya bekor qilinishini (to’xtatilishi) kerak ekanligini bildiradi.
- Quyidagi saqlash (to’xtatish) qoidalari mavjut : SOURCE, CLASS, va RUNTIME

@Retention Annotations - SOURCE

- **RetentionPolicy.SOURCE:** The annotations annotated using the SOURCE retention policy are discarded at runtime.

- Things like @SuppressWarnings, @Override are annotations used by the compiler - not needed at runtime. For those RetentionPolicy.SOURCE would make sense. Also annotations can be used to generate code (look at Spring ROO) - such annotation are also not required at run time.

- **RetentionPolicy.SOURCE:** SOURCE saqlash qoidasi bilan izohlangan annotatsiya lar RUNTIME vaqtida olib tashlanadi.
- @SuppressWarnings, @Override annotatsiyalari kompiler tomonidan ishlatiladi. Ular runtime da kerak emas. Shuningdek ular kod generatsiya qilish uchun ham ishlatilishi mumkin.

@Retention Annotations - CLASS

- **RetentionPolicy.CLASS:** The annotations annotated using the CLASS retention policy are recorded in the .class file but are discarded during runtime. CLASS is the default retention policy in Java.
- This is the default retention policy if you do not specify any retention policy at all.

- **RetentionPolicy.CLASS:** CLASS saqlash qoidasi yordamida izohlangan annotatsiyalar .class faylida yaratilganda bor bo'ladi, lekin RUNTIME vaqtida olib tashlanadi.
- Bu default saqlash qoidasi dir.

@Retention Annotations - RUNTIME

- **RetentionPolicy.RUNTIME:** The annotations annotated using the RUNTIME retention policy are retained during runtime and can be accessed in our program during runtime.

- **RetentionPolicy.RUNTIME:** RUNTIME saqlash qoidasi yordamida izohlangan annotatsiyalar runtime vaqtida saqlanib qoladi va ularga RUNTIME vaqtida dastur orqali olish mumkin.

User-defined (Custom)

- User-defined annotations can be used to annotate program elements, i.e. variables, constructors, methods, etc. These annotations can be applied just before the declaration of an element (constructor, method, classes, etc).

- User yaratgan annotatsiyalar class, class o’zgaruvchilari,constructorlari, metodlari ga ishlatilishi mumkin. Bu annotatsiyar class elementlaridan oldin yoziladi.

- **Syntax:** Declaration

```
[Access Specifier] @interface<AnnotationName>
{
   DataType <Method Name>() [default value];
}
```

User Defined Annotations Problems

- @Print annotation which prints object detail.
- @Json annoatation for making json String from object
- @Lombok /@Getter and @Setter annotations (can not be done by reflection)

Links

- https://www.geeksforgeeks.org/customize-java-annotation-with-examples/?ref=rp
- https://www.baeldung.com/java-custom-annotation