# JAVA: ANNOTATIONS.

*By Saurav Saxena.*

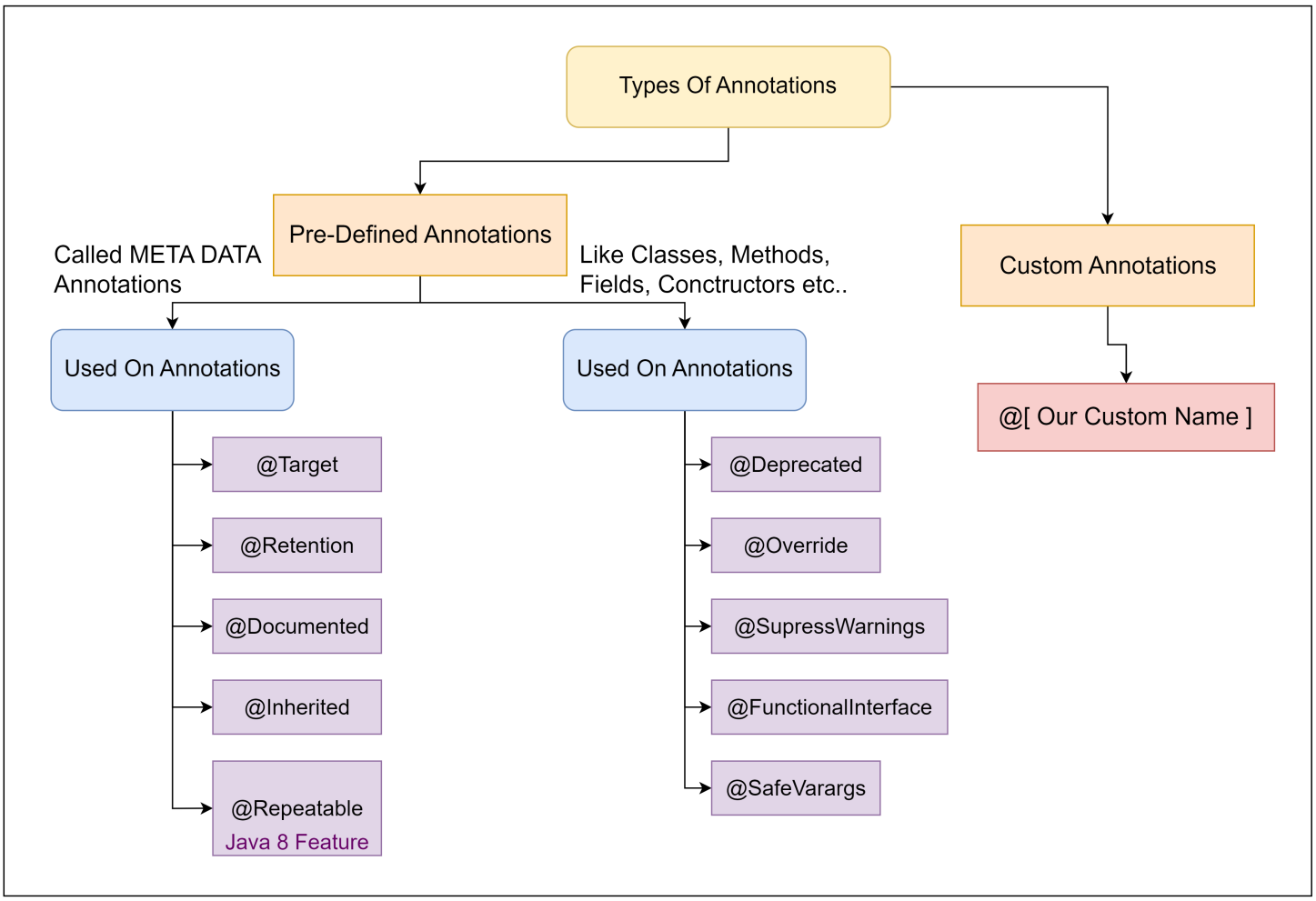
**1. What is Annotation?**

- It is kind of adding META DATA to the java code.
- Means it's usage is OPTIONAL.
- We can use this META DATA information at runtime and can add certain logic in our code if wanted.
- How to Read META DATA information.
- Using Reflection as already discussed in Reflection topic.
- Annotations can be applied at anywhere like classes, Methods, Interface, fields, parameters etc.

```
public interface Bird{
  public boolean fly();
}
public class Eagle implements Bird{
   //Annotation denoted using '@'
   //@Override annotation
   @Override
   public boolean fly(){
     return true;
   }
}
```

> *After adding meta data information like @Override the compiler perform certain logic to check all constrain or rule for @Override like Method signature should be same and many more.*

**2. Types Of Annotations.**




### 3. Annotations Used on Java Code.

#### @Deprecated

- Usage of Deprecated on Class or Method or Fields, shows you compile time WARNING .
- Deprecation means, no further improvements is happening on this and use new alternative method or fields instead.
- Can be used over: Constructor, Field, Local Variable, Method, Package, Parameter, Type(class, interface, enum)

```
public class Mobile{
  @Deprecated
  public void dummyMethod(){
    //defination
  }
}
public class Main{
  public static void main(String[] args){
    Mobile mobile = new Mobile();
    //Shows Warning
```

```
      mobile.dummyMethod();
    }
  }
```

**- @Override.**

- During Comile time, it will check that the method should be Overridden.
- And throws compile time error, If it do not match with the parent method.
- Can be used over: METHODS.

```
public interface Bird{
   public boolean fly();
}
public class Eagle implements Bird{
  //compile time error
  @Override
  public boolean fly1(){
    return true;
  }
}
```

**□ @SupressWarnings**

- It will tell compiler to IGNORE any compile time WARNING.
- Use it safely, could led to Run time exception if, any valid warning is ignored.
- Can be used over: Field, Method, Parameter, Constructor, Local Variable, Type[ Class or Interface or enum]

```
public class Mobile{
  @Deprecated
  public void dummyMethod(){
   //do
  }
}
//class level
//@SupressWarnings("deprecation")
//warning for not used field, methods etc.
//@SupressWarnings("unused")
//@SupressWarnings("all")
public class Main{
   //method level
  //@SupressWarnings("deprecation")
  public static void main(String[] args){
    Mobile mobile = new Mobile();
    //shows warning
```

```
      mobile.dummyMethod();
    }
  }
```

**@FunctionalInterface.**

- Restrict Interface to have only 1 abstract method.
- Throws Compilations error, if more than 1 abstract method found.
- can be used over: Type [ class or interface or enum]

**@SafeVarargs**

- Very very Similar @SpressWarnings
- Used to suppress "HEAP POLLUTION WARNING"
- Used over Methods and Constructor which has Variable Arguments as parameter.
- Method should be ether static or final [I.E. Method can not be overridden].
- In parent class we apply @SafeVarargs on some methods which can be overridden but in child class we forgot to apply these safe checks may causes hinder in our application.
- In java 9, we can also use it on private methods too.

**What is Heap Pollution?**

- Object of one Type [Example String], storing the reference of another type object [Example Integer ]

A obj[STRING] ---------->B obj [INTEGER]

```
public class Log{
  //@SafeVarargs
  public static void printLogValues(List<Integer> ...logNumberList){
     Object[] objectList = logNumberList;
     List<String> stringValueList = new ArrayList<>();
     stringValueList.add("Hello")l
     objectList[0] = stringValueList;
  }
}
```

**4. Annotation used over another Annotations [ META-ANNOTATIONS].**
**@Target**

- Thi meta-annotations will restrict, where to use the annotation.
- Either at method or constructor or fields etc..

```
@Target(ElementType.METHOD)
//this is the we create custom annotation
public @interface Override{
}


@Target({ElementType.CONSTRUCTOR, ElementType.METHOD})
public @interface SafeVarargs{}
```

Element Type:
TYPE: [class,Interface, Enum ]
Field,
METHOD,
PARAMETER,
CONSTRUCTOR,
LOCAL_VARIABLE,
ANNOTATION_TYPE,

```
@Documented
@Retention
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target{
  ElementType[] value();
}
```

PACKAGE,
TYPE_PARAMETER [ allow you to apply on generic type <T>,
TYPE_USE [JAVA 8 feature allow you to use annotation at all places, where type you can declare [ like List<@annotation String>]

 **@Retention:**

- This meta-annotation tells, how Annotation will be stored in java.
- **RetentionPolicy.SOURCE**: Annotations will be discarded by the compiler itself and it will not be recorded in .class file.
- **RetentionPolicy.CLASS**: Annotations will be recorded in .class file but will be ignore by JVM at run time.
- you cannot use reflection because it happens at run time.
- **RetentionPolicy.RUNTIME**: Annotations will be recorded in .class file + available during run time.
- Usage of reflection can be done.

### Example 1: SOURCE

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override{
}
public class Eagle implements Bird{
  //This annotation will not recoreded in
  //Eagle.class file
  @Override
  public void fly(){
  }
}
```

### Example 2: RUNTIME

```
//recoded in .class file and available for //JVM
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.CONSTRUCTOR, ElementType.METHOD})
public @interface SafeVarargs {}
public class Log{
  //@SafeVarargs
  public static void printLogValues(List<Integer> ...logNumberList){
     Object[] objectList = logNumberList;
     List<String> stringValueList = new ArrayList<>();
     stringValueList.add("Hello")l
     objectList[0] = stringValueList;
  }
}
```

### Example 3:

```
//recoded in .class file and available for //JVM
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface MyCustomAnnotationsWithInherited{}
```

```
@MyCustomAnnotationWithInherited
public class TestClass{
}
```

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

```
    //it gives warning because we are
    //trying to access annotation through
    //reflection at runtime but there is a
    //problem, These annotations may
    //presents or may not be at runtime.
       System.out.println(
          newTestClass().getClass().getAnnotation(MyCustomAnnotationWithInherited.class)
       );
  }
}
output:
@MyCustomAnnotationWithInherited()
```

### EXAMPLE 4: Custom Annotations is not present at run time because of RetentionPolicy

```
@Retention(RetentionPolicy.TYPE)
@Target(ElementType.TYPE)
public @interface MyCustomAnnotationsWithInherited{}
//This annotation is not available at //runtime and .class and ignore by JVM
@MyCustomAnnotationWithInherited
public class TestClass{
}
public class Main{
  public static void main(String[] args){
    //it gives warning because we are
    //trying to access annotation through
    //reflection at runtime but there is a
    //problem, These annotations may
    //presents or may not be at runtime.
    System.out.println(new TestClass().getClass()
       .getAnnotation(MyCustomAnnotationWithInherited.class)
    );
  }
}
output:
null
```

### @Documented:

- By default, Annotations are ignored when java Documentation is generated.
- With this meta-annotations even Annotations will come in JAVA DOCS.

### NOT DOCUMENTED:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override{
}
```

```
public class Eagle implements Bird{
  @Override
  public void fly(){
  }
}
```

### Example @SafeVarargs

```
//recoded in .class file and available for //JVM
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.CONSTRUCTOR, ElementType.METHOD})
public @interface SafeVarargs {}
public class Log{
  //@SafeVarargs
  public static void printLogValues(List<Integer> ...logNumberList){
     Object[] objectList = logNumberList;
     List<String> stringValueList = new ArrayList<>();
     stringValueList.add("Hello")l
     objectList[0] = stringValueList;
  }
}
```

### @Inherited:

- By default, Annotations applied on parent class are not available to child classes.
- But it is after this meta-annotation.
- This meta-annotations has no effect, if annotation is used other than class.

```
@Inherited
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface MyCustomAnnotationsWithInherited{}
@MyCustomAnnotationsWithInherited
public class ParentClass{
}
public class ChildClass extends ParentClass{
}
public class Main{
  public static void main(String[] args){
    //it gives warning because we are
    //trying to access annotation through
    //reflection at runtime but there is a
    //problem, These annotations may
    //presents or may not be at runtime.
    System.out.println(new
```

```
ChildClass().getClass().getAnnotation(MyCustomAnnotationWithInherited.class));
  }
}
output:
@MyCustomAnnotationWithInherited()
```

**//removing of @Ingerited while creating Custom annotations.**

```
//This custom annotations will not be
//available for inherted class.
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface MyCustomAnnotationsWithInherited{}
public class ChildClass extends ParentClass{
}
public class Main{
  public static void main(String[] args){
    //it gives warning because we are
    //trying to access annotation through
    //reflection at runtime but there is a
    //problem, These annotations may
    //presents or may not be at runtime.
    System.out.println(new
ChildClass().getClass().getAnnotation(MyCustomAnnotationWithInherited.class));
  }
}
output:
null
```

**□ @Repeatable Annotations:**

- Allow us to use the same annotations more than once at same places.

**WE CAN NOT DO THIS BEFORE JAVA 8**
Example using without **@Repeatable Annotations:**

```
//compile time error
@Deprecated
@Deprecated
public class Eagle{
  public void fly(){
  }
}
```

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Category{
```

```
  String name();
}
```

```
//We are trying to use repeative annotations
//compile time error
@Category(name="Bird")
@Category(name="LivingThing")
public class Eagle{
  public void fly(){
  }
}
```

### Using @Repeateable

```
@Repeatable(Categories.class)
@interface Category{
  String name();
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@interface Categories{
  Category[] value();
}
```

```
@Category(name="Bird")
@Category(name="LivingThing")
@Category(name = "carniVorous")
public class Eagle{
  public void fly(){
  }
}
public class Main{
  public static void main(String[] args){
      Category[] categoryAnnotationArray = new Eagle().getClass()
                                          .getAnnotationsByType(Category.class);

      for(Category annotation:categoryAnnotationArray){
         System.out.println(annotation.name());
      }
  }
}
output:
Bird
LivingThing
carniVorous
```

### 5. User Defined or Custom Annotations:

- We can use create our own ANNOTATIONS using keyword "@interface"

**Creating  an Annotations with empty body:**

```
public @interface MyCustomAnnotations{
}
```

**Creating an Annotations with method [ It's more like a field ]:**

- No parameter, no body
- Return type is restricted to Primitive, Class, String, enums, annotations and array of these types.

```
public @interface MyCustomAnnotations{
   String name() ;
}
```

```
@MyCustomAnnotations(name="testing")
public class Eagle{
  public void fly(){
  }
}
```

**Creating Annotations with an element with default values:**

```
public @interface MyCustomAnnotations{
   String name() default "hello" ;
}
```

```
//here default value apply
@MyCustomAnnotations
public class Eagle{
  public void fly(){
  }
}
```