Build COMPETENCY
across your TEAM
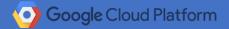
SYNERGETICS
GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Annotations

Google Cloud Platform

Windows Azure

amazon
web services

APACHE
kafka
A distributed streaming platform Apache
SOFTWARE FOUNDATION

APACHE
Spark

Java

hadoop

What's the use of Annotations?

Annotations basics

Built-in Annotations in Java
    @Override
    @Deprecated
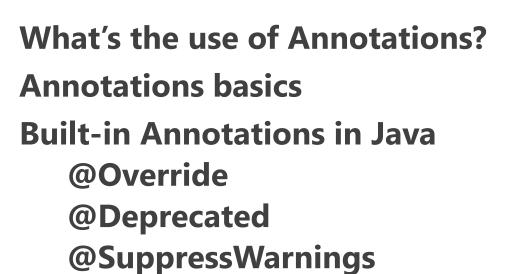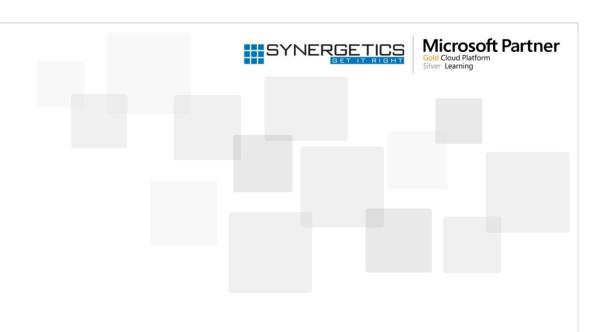   **@SuppressWarnings**

What's the use of Annotations

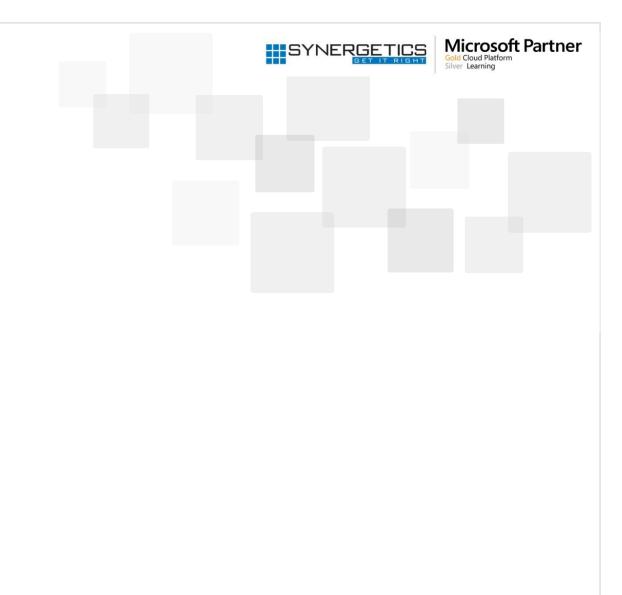Annotations are used to provide supplement information about a program.

- Annotations start with '**@**'.

- Annotations do not change action of a compiled program.

- Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.

- Annotations are not pure comments as they can change the way a program is treated by compiler

```java
class Base
{
    public void display()
    {
        System.out.println("Base display()");
    }
}
class Derived extends Base
{
    @Override
    public void display(int x)
    {
        System.out.println("Derived display(int )");
    }

    public static void main(String args[])
    {
        Derived obj = new Derived();
        obj.display();
    }
}
```

Categories of Annotations

- **Marker Annotations**
- **Single value Annotations**
- **Full Annotations**

Marker Annotations

- The only purpose is to mark a declaration. These annotations contain no members and do not consist any data. Thus, its presence as an annotation is sufficient. Since, marker interface contains no members, simply determining whether it is present or absent is sufficient. @Override is an example of Marker Annotation.

- Example: – @TestAnnotation()

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 value.

- Example: - @TestAnnotation("testing");

Full Annotations

- These annotations consist of multiple data members/ name, value, pairs.

- Example:- @TestAnnotation(owner="Rahul", value="Class Geeks")

Predefined/ Standard Annotations

- Java defines seven built-in annotations.

- Four are imported from java.lang.annotation: **@Retention**, **@Documented**, **@Target**, and **@Inherited**.

- Three are included in java.lang: **@Deprecated,** **@Override** and **@SuppressWarnings**

@Deprecated Annotation

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

- The Javadoc @deprecated tag should be used when an element has been deprecated.

- @deprecated tag is for documentation and @Deprecated annotation is for runtime reflection.

- @deprecated tag has higher priority than @Deprecated annotation when both are together used.

@Override Annotation

- It is a marker annotation that can be used only on methods.

- A method annotated with @Override must override a method from a superclass. If it doesn't, a compile-time error will result (see this for example).

- It is used to ensure that a superclass method is actually overridden, and not simply overloaded.

@SuppressWarnings

- It is used to inform the compiler to suppress specified compiler warnings.
- The warnings to suppress are specified by name, in string form.
- This type of annotation can be applied to any type of declaration.

@Documented Annotations

- It is a marker interface that tells a tool that an annotation is to be documented.

- Annotations are not included by Javadoc comments.

- Use of @Documented annotation in the code enables tools like Javadoc to process it and include the annotation type information in the generated document.

@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.
- The constants are shown below along with the type of declaration to which they correspond.

Constants

- The constants are shown below along with the type of declaration to which they correspond.

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 @Targetannotation. To specify multiple values, we must specify them within a braces-delimited list.
- 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})
- @Retention Annotation It determines where and how long the annotation is retent.

- The 3 values that the @Retention annotation can have:
- SOURCE: Annotations will be retained at the source level and ignored by the compiler.
- CLASS: Annotations will be retained at compile time and ignored by the JVM.
- RUNTIME: These will be retained at runtime.

@Inherited

- @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.

User-defined/ Custom Annotations

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

- Syntax of Declaration:-

[Access Specifier] @interface<AnnotationName>

{

  DataType <Method Name>() [default value];

}

User-defined/ Custom Annotations

- **AnnotationName** is an identifier.

- Parameter should not be associated with method declarations and **throws** clause should not be used with method declaration.

- Parameters will not have a null value but can have a default value.

- *default value* is optional.

- Return type of method should be either primitive, enum, string, class name or array of primitive, enum, string or class name type.

Q & A

Contact: amitmahadik@synergetics-india.com

Thank You