ANNOTATIONS IN JAVA

Java annotations are metadata (data about data) for our program source code.

They provide additional information about the program to the compiler but are not part of the program itself. These annotations do not affect the execution of the compiled program.

Annotations start with '@'. It's syntax is:

```
@AnnotationName
```

The @Override annotation specifies that the method that has been marked with this annotation overrides the method of the superclass with the same method name, return type, and parameter list.

It is not mandatory to use @Override when overriding a method. However, if we use it, the compiler gives an error if something is wrong (such as wrong parameter type) while overriding the method.

Annotation formats

Annotations may also include elements (members/attributes/parameters).

1. Marker Annotations

Marker annotations do not contain members/elements. It is only used for marking a declaration.

Its syntax is:

```
@AnnotationName()
```

Since these annotations do not contain elements, parentheses can be excluded. For example,

```
@Override
```

2. Single element Annotations

A single element annotation contains only one element.

Its syntax is:

```
@AnnotationName(elementName = "elementValue")
```

If there is only one element, it is a convention to name that element as 'value'.

```
@AnnotationName(value = "elementValue")
```

In this case, the element name can be excluded as well. The element name will be 'value' by default.

```
@AnnotationName("elementValue")
```

3. Multiple element Annotations

These annotations contain multiple elements separated by commas.

Its syntax is:

```
@AnnotationName(element1 = "value1", element2 = "value2")
```

Annotation placement

Any declaration can be marked with annotation by placing it above that declaration. As of Java 8, annotations can also be placed before a type.

1. Above declarations

As mentioned above, Java annotations can be placed above class, method, interface, field, and other program element declarations.

Example 2: @SuppressWarnings Annotation Example

```
import java.util.*;

class Main {
  @SuppressWarnings("unchecked")
  static void wordsList() {
    ArrayList wordList = new ArrayList<>();

// This causes an unchecked warning
    wordList.add("programiz");

    System.out.println("Word list => " + wordList);
  }

  public static void main(String args[]) {
    wordsList();
  }
}
```

Output

```
Word list => [programiz]
```

2. Type annotations

Before Java 8, annotations could be applied to declarations only. Now, type annotations can be used as well. This means that we can place annotations wherever we use a type.

Constructor invocations

```
new @Readonly ArrayList<>()
```

Types of Annotations

1. **Predefined annotations**
    1. `@Deprecated`
    2. `@Override`
    3. `@SuppressWarnings`
    4. `@SafeVarargs`
    5. `@FunctionalInterface`
2. **Meta-annotations**
    1. `@Retention`
    2. `@Documented`
    3. `@Target`
    4. `@Inherited`
    5. `@Repeatable`
3. **Custom annotations**

Use of Annotations

- **Compiler instructions** - Annotations can be used for giving instructions to the compiler, detect errors or suppress warnings. The built-in annotations `@Deprecated`, `@Override`,`@SuppressWarnings` are used for these purposes.

- **Compile-time instructions** - Compile-time instructions provided by these annotations help the software build tools to generate code, XML files and many more.

- **Runtime instructions** - Some annotations can be defined to give instructions to the program at runtime. These annotations are accessed using Java Reflection.