10.11-What is Annotation

Introduction:

Annotations in Java are a powerful feature introduced in Java 5. They act as a form of metadata, providing additional information to the compiler or runtime about the program elements (such as classes, methods, or variables). However, annotations do not alter the program's behaviour directly; instead, they can influence how the compiler processes the code.

- **What are Annotations?**
    - An annotation is essentially a tag or indicator attached to a class, interface, method, or field to provide supplementary data.
    - Annotations begin with the @ symbol.
    - They provide metadata that can be processed at compile-time or runtime by the compiler or the JVM.

Key Characteristics of Annotations:

- Annotations do not change the action of a compiled program.
- They help associate metadata (additional information) with program elements.
- Annotations can be applied to variables, methods, constructors, classes, etc.
- Unlike comments, annotations can be used by the compiler or runtime tools to change the way the program is processed.

Built-In Java Annotations

Java provides several built-in annotations that are widely used in coding. These annotations can either be applied directly to the code or to other annotations.

Common Built-In Annotations in Java:

1. **@Override**
   Indicates that a method overrides a method in the superclass. If the method does not actually override anything, a compile-time error will occur.

2. **@SuppressWarnings**
   Instructs the compiler to suppress specific warnings that it would otherwise generate. It's typically used to suppress warnings related to unchecked or deprecated operations.

3. **@Deprecated**
   Marks a method or class as outdated and discourages its use. When this annotation is present, the compiler issues a warning when the deprecated element is used.

Example: Understanding @Override Annotation

Here's a simple example that demonstrates the use of the @Override annotation:

```
class A {
    public void show() {
        System.out.println("In A's show");
    }
}

class B extends A {
    @Override
    public void show() {
        System.out.println("In B's show");
    }
}

public class Demo {
    public static void main(String[] args) {
        A obj = new B();
        obj.show();
    }
}
```

Output:

```
In B's show
```

Explanation:

- In this example, class B overrides the show() method from class A. By adding the @Override annotation above the method, the compiler checks if the method correctly overrides a superclass method.

- If the method name is incorrect, the @Override annotation will raise a compile-time error, helping to prevent potential bugs. For example, if the method in B were mistakenly named showw(), the compiler would notify us that the method does not override anything from the parent class.

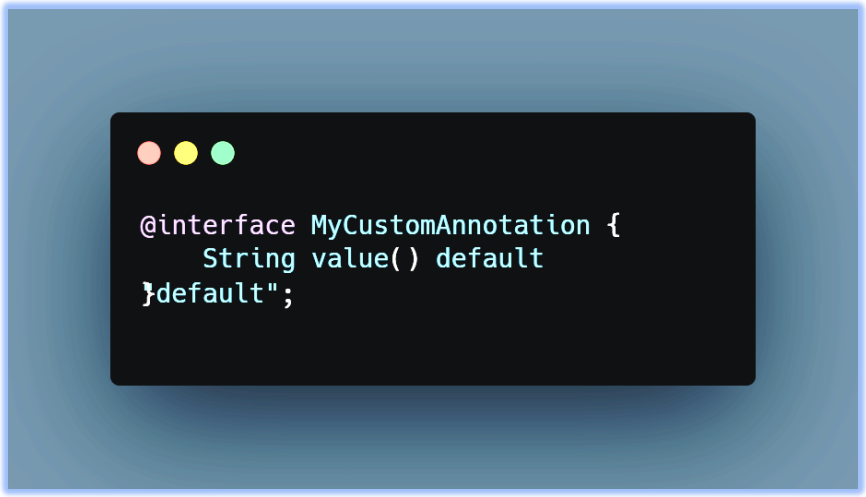
Why Use @Override?

- **Prevents Mistakes:** Helps detect errors when method names are mistyped or incorrectly defined.
- **Improves Code Readability:** Makes it clear to readers that the method is overriding a parent class method, which improves maintainability.

Custom Annotations in Java

Java also allows developers to create their own custom annotations. Custom annotations are declared using the @interface keyword. Although they are not commonly used in core Java, they become more relevant in advanced topics, particularly when dealing with frameworks like Spring or Hibernate.

Example of a Custom Annotation:



- The @interface keyword is used to define an annotation.
- The value element defines a default value for this annotation, which can be overridden when used.

Where to Use Annotations

Annotations can be applied at various levels within your code:

- **Class Level:** Can indicate that a class is deprecated or give other metadata about the class.
- **Method Level:** Used to show method overrides or suppress warnings.
- **Field/Variable Level:** Can mark fields as transient, inject dependencies, or associate metadata.

Annotation Usage in Java Frameworks

- While annotations may not be heavily used in basic Java programming, they are essential in Java frameworks. For example, annotations play a significant role in Spring, Hibernate, and other enterprise frameworks, helping automate tasks such as dependency injection, transaction management, and more.