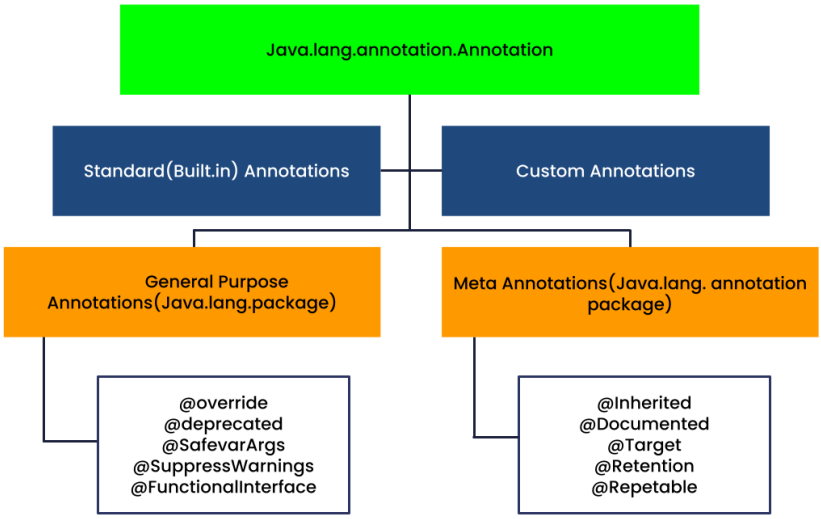
- **In Built Annotation**



Java provides several built-in annotations that can be used to provide additional information about code elements. Here are some of the most commonly used built-in annotations in Java:
@Override: This annotation is used to indicate that a method in a subclass is intended to override a method in the superclass.
@Deprecated: This annotation is used to mark a program element (such as a method or class) as deprecated, indicating that it is no longer recommended for use and may be removed in future versions of the code.
@SuppressWarnings: This annotation is used to suppress specific warnings generated by the Java compiler, such as unchecked cast warnings or deprecation warnings.
@FunctionalInterface: This annotation is used to indicate that an interface is intended to be a functional interface, which means it has a single abstract method and can be used with lambda expressions.
@Retention: This annotation is used to specify the retention policy for an annotation, indicating whether it should be retained at runtime or only used for compilation.
@Documented: This annotation is used to indicate that an annotation should be included in JavaDoc documentation.
@Target: This annotation is used to specify the types of program elements to which an annotation can be applied, such as classes, methods, or fields.
@Inherited: This annotation is used to indicate that an annotation should be inherited by subclasses of an annotated class or interface.

- **Custom Annotation:**

Custom Annotations
Java Custom annotations or Java User-defined annotations are easy to create and use. The @interface element is used to declare an annotation. For example:

```
@interface MyAnnotation{}
```

Here, MyAnnotation is the custom annotation name.

Types of Annotation
There are three types of annotations.

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation