Annotations:-

* Note of explanation or comment added to a text or diagram.
* In java, we need to add extra information to code, (Data about data or Meta Data)
* Organized data about data, embedded within the code.
* Decorative data
* Most probably you have seen one, on top of class, function, variables.
* Something with @, one top of function, variable, class.

eg. @Override
public void run() {}

@FunctionalInterface
public class Test {}

We already have comments for meta data
Why we need Annotations??

.java (High Level Language) → Compiler → .class (Bytecode) → execution

Compiler ⇝ Lexical Analysis → Tokenization
& Syntax ,, - Removes extra info,
- Removes comments
- Generates Token

So,
Comments won't be present after compilation.
" are for human not for JVM or Compiler.
let's say we want to inform compiler about something.

→ Annotation → introduced from Java 1.5.
Earlier we used to have XML, still in Spring, Hibernate, Struts.

- These XML files you can provide externally and fetch runtime.

XML file.

code → Runtime → O/P

XML vs Annotations

- Need to learn explicitly
- Data Outside of code, need to maintain, Location, Parsing etc.

Web.xml

```
@WebServlet("/getorder")
Public class GETOrder {
}
```

Types of Annotations

(1) Standard Annotation (Predefined/ Built in)
- Defined & provided by Java.
- java.lang.annotations.Annotations
- 8 annotation Under 2 categories.

General purpose	Meta annot. (Annotation about annotation)
@Override	@Documented
@SuppressWarnings	@Inheritated
@Deprecated	@Target
@FunctionalInterface	@Retention
→ Under java.lang package	Under java.lang.Annotation

(2) Custom Annotations / User defined Annotations
- We can create & add custom functionality.
- Created using predefined annotations.

eg. @Documented
@Retention(RetentionPolicy.RUNTIME)
@interface TestAnnotation {
String name();
String date();
}

* Annos can be used with Class, Method or Variable.

Based on no. of Annotation members :-

(1) Marker Annotations
- No values, No Members
- eg. @Override

(2) Single Valued Anno. -
- One member or Value
- eg. @SupressWarnings(value = "unchecked")
or " " ("unchecked")

(3) Multi valued - - -
- Multi values
- eg. @WebServlet(name = "abc.servlet",
urlPattern = "/abc.html")

Meta Annotation
(Anno. about Anno.)
- All annos in java need to have two anno. @target & @Retention.

- (1) Target
- Where you want to use it
- (class, variable/Method)
- @Target (Single Value anno.)
public @interface Target
{ ElementType[] value();
}

(2) Retention.
- Continute to hold
- Till what level, we want to hold Annotation.

ElementType is an Enum in java.lang.annotations package & possible values.

- ANNOTATION_TYPE
  CONSTRUCTOR
  FIELD
  LOCAL_VARIABLE
  METHOD
  PACKAGE
  PARAMETER
  TYPE (for class, interface or Enum)
- ElementType is an Array, Multiple values.

General Phases:-

source code (1) → compiler (2) → Runtime (3)
.java → .class

@Retention (Single value)

```
public @interface Retention
{
  RetentionPolicy value();
}
```

RetentionPolicy is an Enum in java.lang.annotations package
Not an array takes Single Value
SOURCE — compiler will remove annotation while compiling / source code will have
CLASS — keep in .class (Default) but remove while runtime
RUNTIME — Retain in all the phases

(3) @Inherited (Marker Annotation)

* To make our custom Annotation Inheritable
* By default in java annos are not inheritable.
* eg. @Inherited public @interface SecurityClearance {
  String level;
  }

```
@SecurityClearance(value="Level A")
public class Employee {
}
```

```
public class Manager extends Employee {
}
```

* This annotation only effects if custom Annotation is for CLASS

(4) @Documented (Marker annotation)

* Particular annotation is to be documented by JAVADOC or similar tools.

General Purpose.

(Cro for each in java doc).

(1) @Override (Marker Annos).

* To generate errors.

(2) @Deprecated (Marker annos. till 8)

* In java 9 2 field { Since - which version it is deprecated
  forRemoval - will be removed in upcoming version or not }.

* To generate warnings.

(3) @FunctionalInterface. (Marker Annos)

* To declare interface of functional type
* Compiler will generate warnings if interface doesn't fit into definition of functional interface.
* Only one abstract method
* Used with Lambda func^n.

(4) @SuppressWarnings (Single Valued Anos)

* To suppress warnings.
* eg. @SuppressWarnings ("unchecked") or ("deprecation")
  public class Calculator {

5