

Learning Java - A Foundational Journey

Session: 11

Annotations and Base64





- ◆ Explain declaring an annotation type in Java
- ◆ Describe predefined annotation types
- ◆ Explain Type annotations
- ◆ Explain Repeating annotations
- ◆ Describe Base64 encoding

For Aptech Centre Use Only



◆ Annotations:

- ◆ Are comments, notes, remarks, or explanations
- ◆ Affect the way a program is treated by tools and libraries, which in turn, can affect program semantics
- ◆ Represent specific use of the type
- ◆ An annotation declaration consists of '@' (at) followed by annotation type
- ◆ Can be added at the class level, field level, and method level

- ◆ Java SE 6 introduced an enhanced feature in annotations, called Pluggable Annotation Processing API. It is useful in Java 15 too.





- ◆ Common uses of annotations are as follows:

Information for the compiler:

Compiler uses annotations to produce notification or errors based on different rules.

Documentation:

Software applications use annotations to decide quality of code or generate reports automatically such as Jenkins.

Code generation:

Annotations can be used to create code or XML files automatically using metadata information available in the code.

Runtime processing:

Annotations that are observed in runtime are used for different purposes such as unit testing, dependency injection, validation, logging, data access, and so on.



The shortest manner in which you can declare or apply a Java annotation is by prefixing an @

Syntax

@<name>

Here,

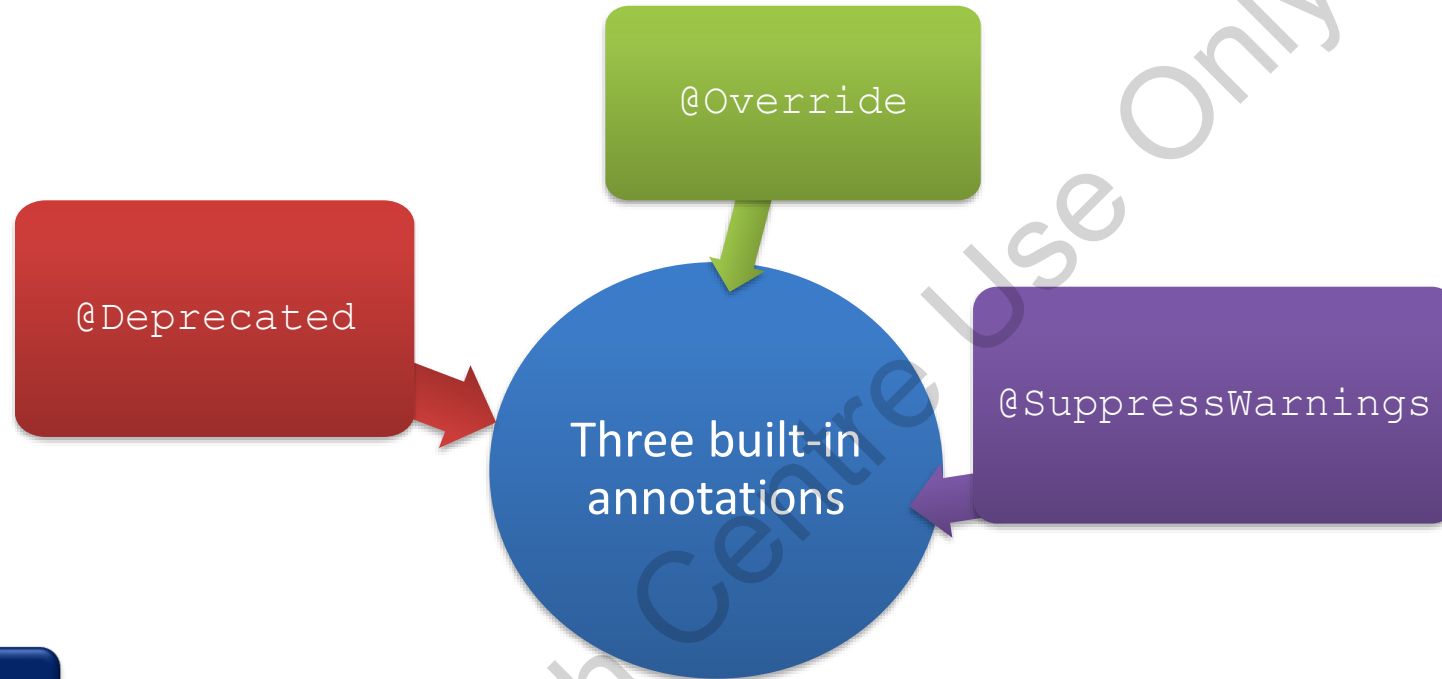
@(at) symbol signals to the compiler that this is an annotation.

The name following the @ symbol is the name of the annotation.

Example:

@Item

Here, the annotation name is Item



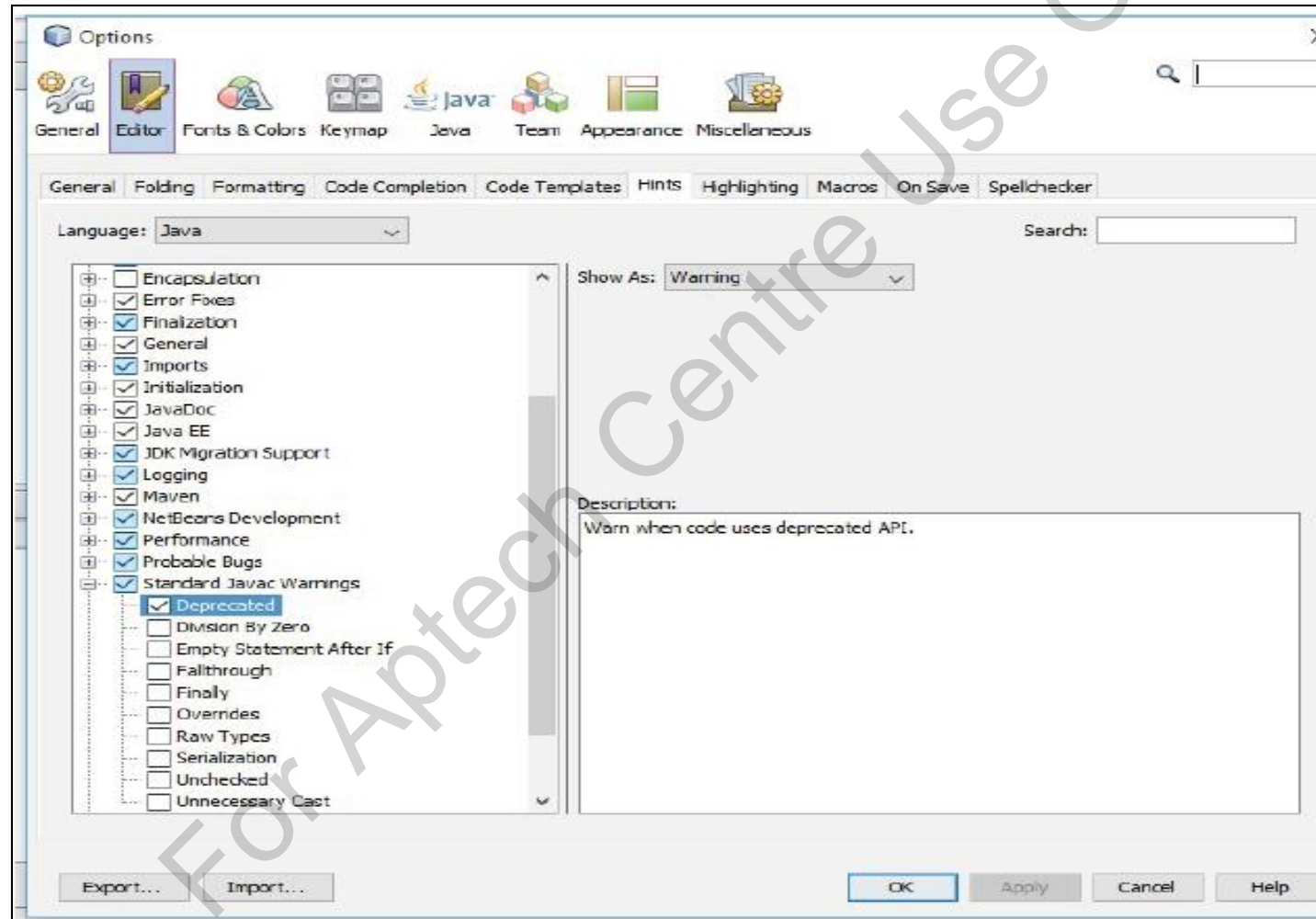
@Deprecated

Deprecates or marks a class, method, or field as deprecated, signifying that the part of code will no longer be used.

Predefined Annotation Types 2-4



- ◆ Configuring NetBeans to show Annotations:





@Override

- Used to create a compile time check to indicate that a method is being overridden
- Not essential in order to override a method in a superclass

```
public class ClassOne
{
    public void show(String testmsg) {
        System.out.println(testmsg);
    }
    public static void main(String args[]) {
        SubClass obj = new SubClass();
        obj.show("Good day!!");
    }
}
class SubClass extends ClassOne
{
    @Override
    public void show(String testmsg) {
        System.out.println("I want to say: "+testmsg);
    }
}
```

Output:

I want to say: Good day



```
@SuppressWarnings
```

- Can suppress compiler warnings in any available method
- For example, a warning will be generated, when a deprecated method is called by a method

Creating Custom Annotations 1-4



- ◆ Custom annotations can be created in Java. Similar to a Java class or interface, such annotations are defined in their own file.

```
@interface SampleAnnotate{  
    String samplValue();  
    String name();  
    int age();  
    String[] addNames();  
}
```

@Retention

This annotation tells the compiler how long annotations with the annotated type should be retained.

Syntax:

```
@Retention(RetentionPolicy.value)
```

where,

`RetentionPolicy` determines the stopping point of annotation



Enumeration with Three-Constant values

RetentionPolicy.RUNTIME

RetentionPolicy.CLASS

RetentionPolicy.SOURCE

@Target

```
@Target ({ElementType.METHOD})  
public @interface SampleAnnotate{...  
}
```

The ANNOTATION TYPE `target` means Java annotation definitions. Hence, the annotation can only be used to annotate further annotations such as `@Target` and `@Retention` annotations.

The TYPE `target` denotes any type such as a class, interface, enum, or annotation.

Creating Custom Annotations 3-4



@Inherited

Indicates that a custom Java annotation in a class is inherited by, subclasses inheriting from that class

```
import java.lang.annotation.Inherited;
@Inherited
public @interface SampleAnnotate {
}
@SampleAnnotate
class Person { ... }
public class Employee extends Person { ... }
```

@Documented

Utilized for informing the JavaDoc tool that custom annotations have to be visible in the JavaDoc for classes that are using custom annotation

```
import java.lang.annotation.Documented;
@Documented
@interface TestAnnotate {
    ...
}
@TestAnnotate
public class Employee { ... }
```

Creating Custom Annotations 4-4



Annotation Placement

Java annotations are placed on top of classes, interfaces, methods, method parameters, fields, and local variables.

```
@Entity  
public class Gadget {  
}
```

Declaring an Annotation Type

Various annotations can substitute comments in coding.



Type annotations are formed to maintain better analysis of Java programs, ensuring better type checking

Syntax

```
@NonNull String str;
```

where,

str is a String variable in the annotation.

Class instance creation expression:

Syntax:

```
new @Interned MyObject()
```

where,

MyObject is a class

Typecast:

Syntax:

```
myString = (@NonNull String) str;
```

where,

myString is a string variable



Required to apply same annotation to a declaration or type use

Declare a Repeatable Annotation Type

- ◆ This annotation type mentioned with the `@Repeatable` meta-annotation.

```
import java.lang.annotation.Repeatable;
@Repeatable(ScoreSchedules.class)
public @interface ScoreSchedule {
    String monthDay() default "1st";
    String weekDay() default "Monday";
    int hour() default 12;
}
```

Declare Containing Annotation Type

- ◆ The containing annotation type contains an array type element value. The array type value must be repeatable annotation type.

```
public @interface ScoreSchedules {
    ScoreSchedule[] value();
}
```



- ◆ Reflection API of Java can be used to access annotations on any type such as class or interface or methods.
- ◆ Several methods in Reflection API retrieve annotations.

@Functional Interface

- ◆ A functional interface is an interface that has one abstract method (not default). The compiler will operate the annotated element as a functional interface and generates an error if the element does not comply with the requirements.

```
@FunctionalInterface
interface MyCustomInterface
{
    ....
}
```




- ◆ Many libraries these days are utilizing annotations for various reasons such as:

Code Quality Analysis

Unit Testing

XML Parsing

Dependency Injection

- ◆ Some of these libraries include:

JUnit

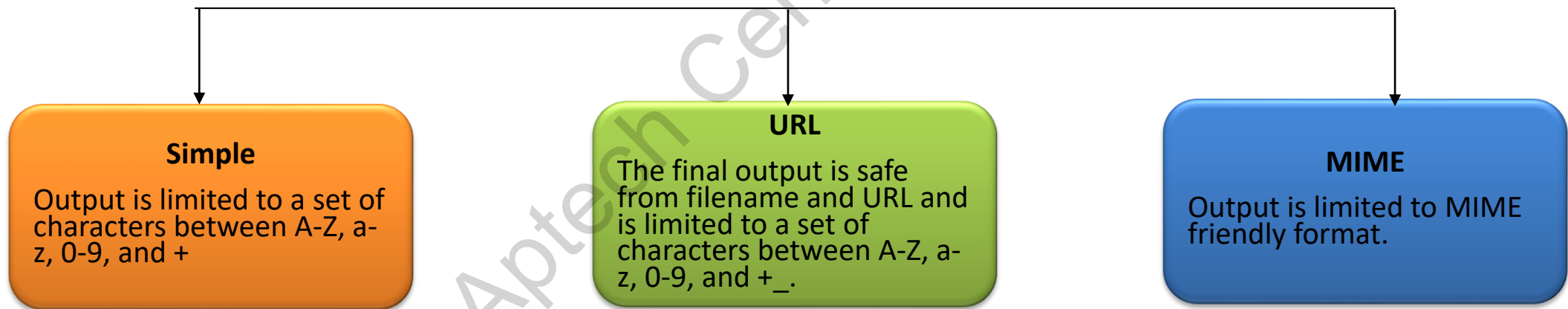
JAXB

FindBugs





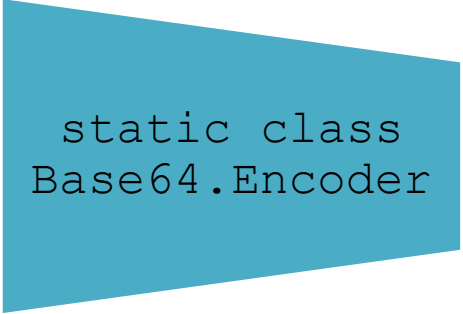
- ◆ Java comprises inbuilt encoder and decoder for Base64 encoding.
- ◆ There are three types of Base64 encoding:





Nested Classes

There are two nested classes in Base64:



```
static class  
Base64.Encoder
```



```
static class  
Base64.Decoder
```

Methods

- ◆ `static Base64.Decoder getDecoder()`
- ◆ `static Base64.Encoder getEncoder()`
- ◆ `getMimeDecoder()`
- ◆ `getMimeEncoder()`
- ◆ `getMimeEncoder(int lineLength, byte[] lineSeparator)`
- ◆ `getUrlDecoder()`
- ◆ `getUrlEncoder()`



- ◆ Annotations are comments, notes, remarks, or explanations.
- ◆ In Java, annotations help to associate additional information (also called metadata) to the program elements.
- ◆ Annotations can be determined from source files or class files at runtime.
- ◆ The `@Deprecated` annotation is used for deprecating or marking a class, method, or field as deprecated, signifying that the part of code no longer will no longer be used.
- ◆ The `@SuppressWarnings` annotation can suppress the compiler warnings in any available method. For example, a warning will be generated, when a deprecated method is called by a method.
- ◆ Base64 encoding has an in-built encoder and a decoder. There are three types of Base64 encoding namely, Simple, URL, and MIME.