JAVA PROGRAMMING BASICS

Module 3: Java Standard Edition

Training program

1. Java I/O Streams
2. Java Serialization
3. Java Database Connectivity
4. Java GUI Programming
5. The basics of Java class loaders
6. Reflections
7. Annotations
8. The proxy classes
9. Java Software Development
10. Garbage Collection

Module contents

1. Annotations
   - Annotations Basics
   - Annotation Types Used by the JavaSE
   - Custom annotations in Java

Module contents

1. Annotations
   - Annotations Basics
   - Annotation Types Used by the JavaSE
   - Custom annotations in Java

Annotations Basics 1/4

- *Annotations*, a form of metadata, provide data about a program that is not part of the program itself
- Annotations have no direct effect on the operation of the code they annotate
- Syntactically, annotations are a special kind of interfaces that are attached to various program elements in the source code, such as classes, variables, and methods
- Java annotations were added to Java from Java 5

Annotation Basics

- Annotations are tags inserted into source code for processing by software tools.
- Such tools can operate at the source code or bytecode level.
- Annotations can be processed at runtime using the Java Reflection API.
- There are some existing annotations that are processed by the Java compiler, and even more annotation support is provided by various frameworks like Java EE/Jakarta and the Spring Framework.

Annotations Basics 2/4

- Annotations have a number of uses, among them:
- **Information for the compiler**
- **Compile-time and deployment-time processing**
- **Runtime processing**

Annotations Basics 3/4

- An annotation looks like the following:
- @Entity
- The at sign character (@) indicates to the compiler that what follows is an annotation

Annotations Basics 4/4

- Annotations can be applied to declarations: declarations of classes, fields, methods, and other program elements.
- When used on a declaration, each annotation often appears, by convention, on its own line.

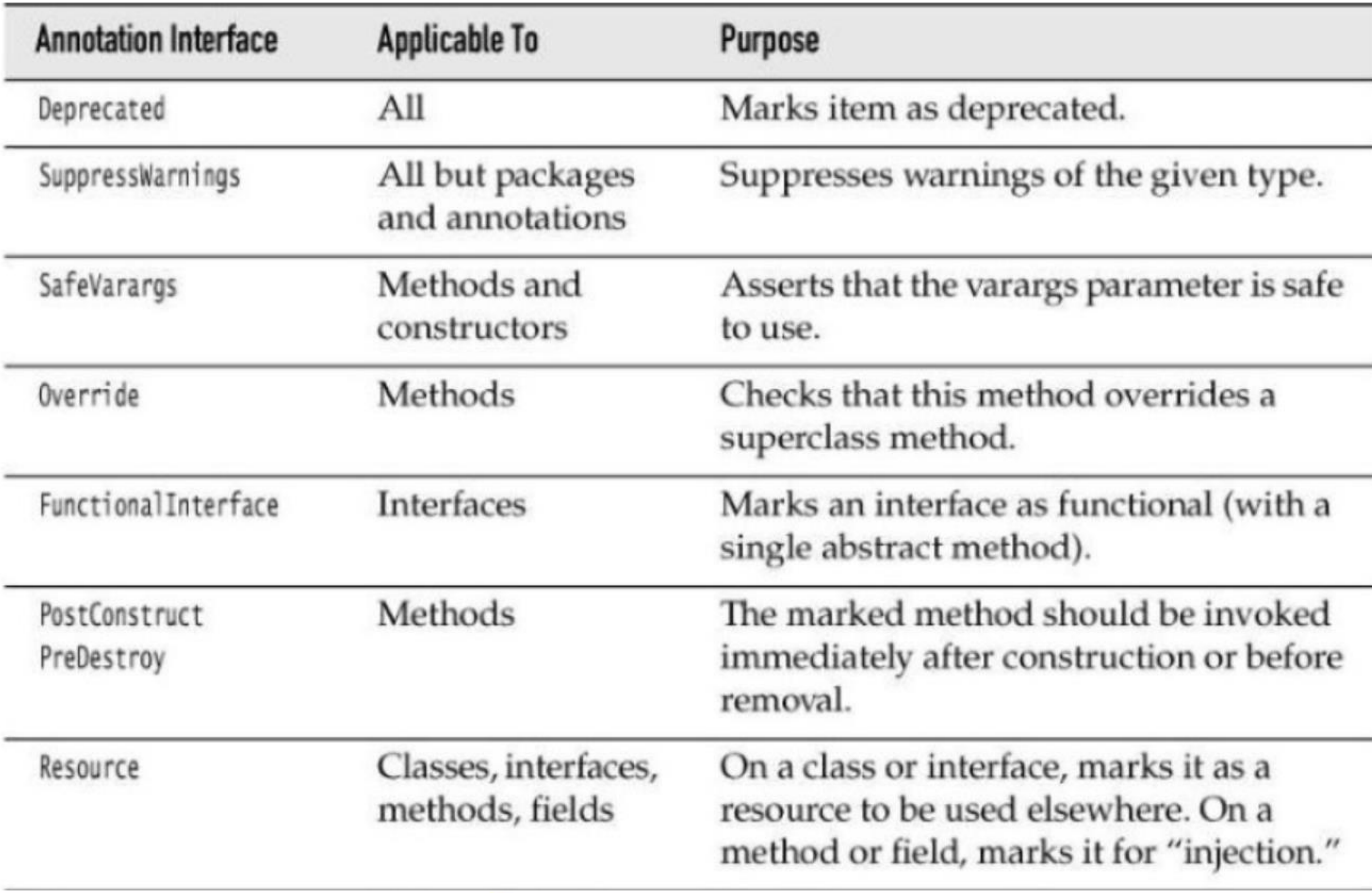
Module contents

1. Annotations
   - Annotations Basics
   - Annotation Types Used by the JavaSE
   - Custom annotations in Java

Annotation Types Used by the JavaSE 1/4

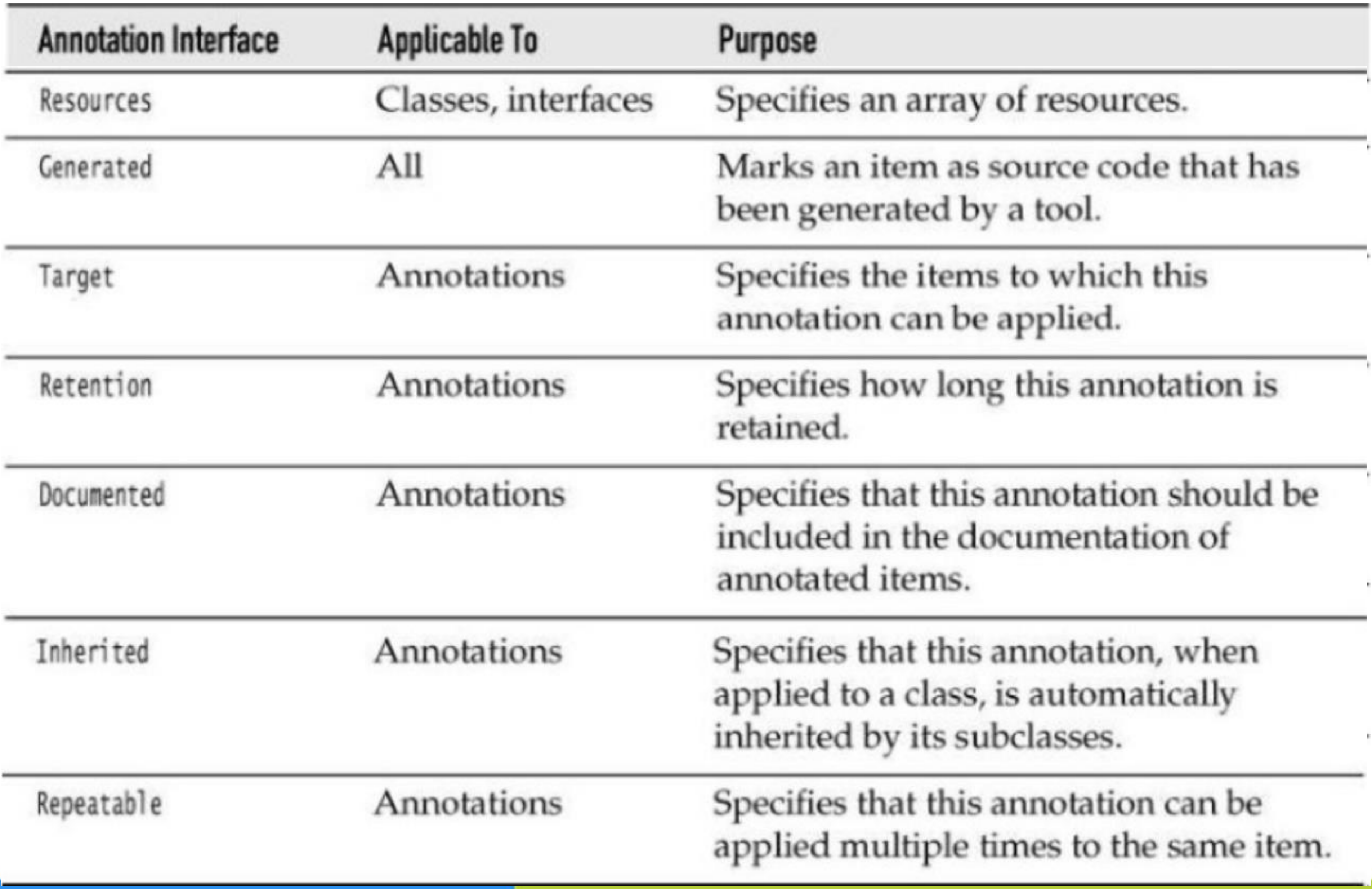
- A set of annotation types are predefined in the Java SE API.
- Some annotation types are used by the Java compiler, and some apply to other annotations
- The predefined annotation types defined in java.lang are
- @Deprecated, @FunctionalInterface
- @Override, @SafeVarargs
- and @SuppressWarnings

Standard annotations - 1



Annotation Interface	Applicable To	Purpose
`Deprecated`	All	Marks item as deprecated.
`SuppressWarnings`	All but packages and annotations	Suppresses warnings of the given type.
`SafeVarargs`	Methods and constructors	Asserts that the varargs parameter is safe to use.
`Override`	Methods	Checks that this method overrides a superclass method.
`FunctionalInterface`	Interfaces	Marks an interface as functional (with a single abstract method).
`PostConstruct` `PreDestroy`	Methods	The marked method should be invoked immediately after construction or before removal.
`Resource`	Classes, interfaces, methods, fields	On a class or interface, marks it as a resource to be used elsewhere. On a method or field, marks it for "injection."

Standard annotations - 2



Annotation Interface	Applicable To	Purpose
Resources	Classes, interfaces	Specifies an array of resources.
Generated	All	Marks an item as source code that has been generated by a tool.
Target	Annotations	Specifies the items to which this annotation can be applied.
Retention	Annotations	Specifies how long this annotation is retained.
Documented	Annotations	Specifies that this annotation should be included in the documentation of annotated items.
Inherited	Annotations	Specifies that this annotation, when applied to a class, is automatically inherited by its subclasses.
Repeatable	Annotations	Specifies that this annotation can be applied multiple times to the same item.

Annotation Types Used by the JavaSE 2/4

- **@Override** annotation informs the compiler that the element is meant to override an element declared in a superclass.

```
@Override
int overriddenMethod() {
…
}
```

see predefined.override.Square

Annotation Types Used by the JavaSE 3/4

- **@Deprecated** annotation indicates that the marked element is deprecated and should no longer be used.
- The compiler generates a warning whenever a program uses a method, class, or field with the @Deprecated annotation.

```
/** * @deprecated * explanation of why it was
deprecated */
@Deprecated static void deprecatedMethod() {
    //...
}
```

see predefined.deprecated.DeprecatedAnnoDemo

Annotation Types Used by the JavaSE 4/4

- **@SuppressWarnings** annotation tells the compiler to suppress specific warnings that it would otherwise generate

```
@SuppressWarnings("deprecation")
public static void main(String[] args) {
    long dt = Date.parse(args[0]);
}
```

- *"deprecation"* tells the compiler to ignore when we're using a deprecated method or type
- *"unchecked"* tells the compiler to ignore when we're using raw types

see predefined.suppresswarnings.Machine

Annotated Types used by Java SE

- Java 5 introduced the concept of varargs, or a method parameter of variable length, as well as parameterized types.
- Combining these can cause problems and compiler warns: "Possible heap pollution from parameterized vararg type"
- @SafeVarargs annotation disabled this warn.

see predefined.predefined.safevarargs

Module contents

1. Annotations
   - Annotations Basics
   - Annotation Types Used by the JavaSE
   - Custom annotations in Java

Custom annotations in Java 1/8

- The @interface declarations informs java that this is custom annotation

```
@Inherited
@Target(value= ElementType.METHOD)
@Retention(value= RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    String param1() default "some def value";
    String param2();
}
```

Custom annotations in Java

- Annotations are an interface and consist only of method declarations called annotation-type **elements**.
- Method declarations have no parameters and no exception declarations (throws).
- Return types can be primitives, objects of classes String, Class, enumerations, annotations and arrays of such types.
- The bodies of the methods are not defined, but are implemented by means of reflection.
- Usually the interface annotation is also annotated with so-called **meta-annotations**.

Custom annotations in Java 2/8

- @Target annotation restricts to which source code elements the custom annotation can be applied
- ElementType.ANNOTATION_TYPE
- ElementType.CONSTRUCTOR
- ElementType.FIELD
- ElementType.LOCAL_VARIABLE
- ElementType.METHOD
- ElementType.PACKAGE
- ElementType.PARAMETER
- ElementType.TYPE

Custom annotations in Java 3/8

- @Retention annotation specifies how the marked annotation is stored:
- RetentionPolicy.SOURCE – The marked annotation is retained only in the source level and is ignored by the compiler.
- RetentionPolicy.CLASS – The marked annotation is retained by the compiler at compile time, but is ignored by the Java Virtual Machine (JVM).
- RetentionPolicy.RUNTIME – The marked annotation is retained by the JVM so it can be used by the runtime environment.

Custom annotations in Java 4/8

- **@Inherited** annotation indicates that the annotation type can be inherited from the super class. (This is not true by default.)
- When the user queries the annotation type and the class has no annotation for this type, the class' superclass is queried for the annotation type. This annotation applies only to class declarations.

Custom annotations in Java 5/8

- **@Documented** annotation indicates that whenever the specified annotation is used those elements should be documented using the Javadoc tool

- By default, annotations are not included in Javadoc

Custom annotations in Java 6/8

-
  ```
  public class SomeClass {
      @MyAnnotation(param1 = "a1", param2 = "a2")
      public static void doJob1() {
          //...
      }
      @MyAnnotation(param2 = "b2")
      public static void doJob2() {
          //...
      }
  }
  ```

Custom annotations in Java 7/8

```
Class<?> cls = SomeClass.class;
Method[] methods = cls.getDeclaredMethods();
for (Method method : methods) {
   if (method.isAnnotationPresent(MyAnnotation.class)) {
      MyAnnotation ma =
            method.getAnnotation(MyAnnotation.class);
      System.out.println(method.getName()+"->“
            + ma.param1()+","+ma.param2());
   }
}
```

Custom annotations in Java 8/8

- **Console output**

```
doJob1->a1,a2
doJob2->some def value,a2
```