Java training

Annotations

Session overview

- **Annotations**
- **Hands-on** - using annotations

Annotations - what and why

- **Form of metadata** - information about a class, method or field
- Provide data about an entity that is not part of that entity
- Have no direct effect on the entity they are annotating
  - Processed by other classes, which determine their usage
- Use-cases:
  - **Information for the compiler** - used to detect errors / suppress warnings
  - **Compile- / deployment-time processing** — some tools & libraries process annotation information to generate code, XML files etc
  - **Runtime processing** — some annotations are specified to be examined at runtime

Scope (places of usage)

- Package
- Class
- Method
- Method parameters
- Field

JDK annotations samples

- Method overriding

```
@Override    // the preceding @ indicates the presence of an annotation
public void overridenMethod() { ... }
```

- Suppressing warnings

```
@SuppressWarnings("unused")     // suppresses warnings of unused code
public void methodWithUnusedVariables() { ... }
```

- Specifying deprecated methods and/or classes

```
@Deprecated      // marks methods which should no longer be used
public Product shouldNotBeUsedAnymore() { ... }
```

`@Override` - usage details

- **`@Override`** - specifies that a method overrides a (super) method
- Using it - instructs the compiler to *mark the annotated method as overriding a method* → more than a marker annotation, a safety check annotation
- Recommended to be used on overriden methods → if the overriden method is renamed or removed, the compiler will report an error

Hands-on →

Annotation elements (properties)

- Annotations can have *elements* (properties)
- Example:

```
@Table(      // the parenthesis are mandatory for annotations with elements
    name = “product”,
    schema = “main”
)
```

- They have a default value → applied if the element’s value is not specified

Annotation elements

- If the annotation uses a single element - its name can be omitted

```
@SuppressWarnings(“unused”)
void processProduct(Product product) {...}
```

- Can have any type, even arrays

```
@RequestMapping(method = {GET, POST})
```

- Can contain other annotations as elements (further discussed)

Multiple annotations

- **Multiple annotations** can be used on a class / field / method

```
@Autowired
@Qualifier(“productService”)
public ProductService productService;
```

- **Repeating annotations** → using the same annotation multiple times (1.8+)

```
@Author(“john.doe”)
@Author(“jane.doe”)
public class Product {...}
```

Predefined annotations

- `@Deprecated` indicates a deprecated element (should no longer be used)
- `@Override` compiler info on overrides of elements declared in a superclass
- `@SuppressWarnings` suppress specific warnings that it would otherwise generate
  - Values: “unused”, “deprecation”, “unchecked”
- `@SafeVarargs` no potentially unsafe operations on its varargs parameter
- `@FunctionalInterface` used on functional interfaces (Java 8+)

Create your own

```
public @interface TrainingSession {
    TrainingDay day();
    String topic();          // makes the element mandatory
    String[] subTopics() default {};       // using an array of elements
}

@TrainingSession(
    topic = “annotations”,
    subTopics = {“usage”, “creation”}
)
public class ProductService {
}
```

Recommended writing mode for annotations with many elements → improved readability

```
public enum TrainingDay {
   D01,D02,D03
}
```

Hands-on

- Trying some of the standard JDK annotations
  - `@Override, @SuppressWarnings, @Deprecated`

Further reading

- Java annotations tutorial
- https://docs.oracle.com/javase/tutorial/java/annotations/index.html
- https://en.wikipedia.org/wiki/Java_annotation

Q&A session

You ask, I answer :)