SWE 4301 - OBJECT ORIENTED CONCEPTS II

OOC-2 LAB 10 HANDOUT AND TASKS

Prepared by:
Md. Jubair Ibna Mostafa
Assistant Professor, IUT CSE

Department of Computer Science and Engineering
Islamic University of Technology

Contents

1 ANNOTATION

Java annotations are a mechanism for adding metadata information to source code. They're a powerful part of Java that was added in JDK5 and improved in the subsequent versions. We can use annotations by an "@" symbol.

Example:

```
@Override
public String toString() {
    return "Student{" + "ID='" + ID + '\'' + ", name='" + name + '}';
}
```

Here, @Override is an annotation. It ensures that you are overriding an method of a parent class.

1.1 Annotation's Type

Annotation is three types based on the number of values it accept.

- Marker annotation
- Single-value annotation
- Multi-value annotation

Marker annotation takes no parameters. Used to mark an element to process in a particular way. Built-in @Override is marker annotation. **Example** previous example.

Single-value annotation provides a single piece of data. It can be used as data=value pair or only value within parenthesis.

```
@SuppressWarnings("warningCategory")
public String something() {
    // code
}
```

Multi-value annotation has multiple data members, all values have to specify as data=value pair and comma separated within parenthesis.

```java
@SuppressWarnings("warningCategory". "again")
public String something() {
    // code
}
```

1.2 Custom Annotation

Custom annotations are declared if built-in annotations are not sufficient enough to meet business requirements. We can declare it as following:

Custom Annotation Example:

```java
public @interface Hello() {
   String greeting();
}
```

Here, @interface ensures that it is a custom annotation. We can give default value as well while declaring an annotation.

Custom Annotation Example with default value:

```java
public @interface Hello() {
   String greeting() default "World";
}
```

1.3 Meta-Annotation

Meta-annotations are annotations that can be applied to other annotations for configuration purpose. Example of these meta-annotations are - @Target @Retention @Inherited @Documented @Repeatable

1.3.1 Target Element

Target Element is used to define the scope of each annotation (i.e., you are restricting the use of annotation). Here, elements could be one or multiple of the following items.

- $@Target(ElementType.TYPE)$ it restricts the use of annotation to only class or interface (as we consider class and interface as type.)

- $@Target(ElementType.METHOD)$ it restricts the use of annotation to only method.
- $@Target(ElementType.FIELD)$ it restricts the use of annotation to only field of a class.
- $@Target(ElementType.CONSTRUCTOR)$ it restricts the use of annotation to only constructor of a class.
- $@Target(ElementType.PARAMETER)$ it restricts the use of annotation to only parameter of a class.
- $@Target(ElementType.LOCAL_VARIABLE)$ it restricts the use of annotation to only local variable of a class.
- $@Target(ElementType.ANNOTATION_TYPE)$ it restricts the use of annotation to only annotation.
- $@Target(ElementType.PACKAGE)$ it restricts the use of annotation to only package.

1.3.2 Retention Policy

Retention policy indicates where the annotation will be applied in the program's lifecycle. It could be one of three values.

- $@Retention(RetentionPolicy.SOURCE)$ – The annotation is used at compile time and discarded at runtime.
- $@Retention(RetentionPolicy.CLASS)$ – The annotation is stored in the class file at compile time and discarded at run time.
- $@Retention(RetentionPolicy.RUNTIME)$ – The annotation is retained at runtime.

2 REFLECTION

Reflection is then the ability to make modifications at runtime by making use of introspection. Java reflection allows us to inspect and/or modify runtime attributes of classes, interfaces,

fields and methods. we can instantiate new objects, invoke methods and get or set field values using reflection One very common use case in Java is the usage with annotations.

Introspection The ability to inspect code in the system or asking an object for its meta-information is called introspection. Introspection and annotations belong to what is called reflection and meta-programming.

Example:

```
Student student = new Student();
student.getClass()
Student.class.getAnnotations();
```

Here, we are accessing student object's information like annotations.

Example of Reflection

```
public class Person {
    private String name;
    private int age;
}

@Test
public void givenObject_whenGetsFieldNamesAtRuntime_thenCorrect() {
    Object person = new Person();
    Field[] fields = person.getClass().getDeclaredFields();

    List<String> actualFieldNames = getFieldNames(fields);

    assertTrue(Arrays.asList("name", "age")
      .containsAll(actualFieldNames));
}
```

3 TASKS AND INSTRUCTIONS

1. Create an annotation named **DevelopmentHistory**. It will satisfy the following constraints

- retained during runtime.
- multi-value annotation with default values (version=1, developer, tester="")
- can be used to Types.
- documented in JavaDoc

2. Create another annotation named **DevelopmentHistoryWithReviewer**. It will extends **DevelopmentHistory** annotation and additionally satisfy following constraints.
   - multi-value annotation with default values (version=1, developer, tester="", reviewers={})
   - can be used to constructors and methods.
3. Create a class **Faculty** class with the following details.
   - name, designation, salary and a list of **Course** as attribute. Course class has course code, name, credit, and type (theory or lab)
   - It has two methods teach(course) and research(topic) where course is a Course type parameter and topic is a String.
   - It has two constructors one having all attributes and another does not have list of courses.
   - Use early created annotations in class, constructor and methods. Use **DevelopmentHistory** in a constructor and both to another constructor. Use **DevelopmentHistory** and **DevelopmentHistoryWithReviewer** to both methods. If you use annotations in other elements, you will get an error.
4. Write 3 test cases that will asserts the use of annotations in class, constructors and methods.