FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 3 – SUPPORT PRESENTATION (II)

Reflection Techniques Examples

Problem: Given an instance of any object, extract the fields and the corresponding values

(1) Line 9 – obtain the class of the object and request the declared fields of that class

(2) For each declared field :

- Line 10 - Set the field accessible (most of them are private => need to change permissions)
- Line 13 - Return the value of the current field contained in the specified object
- Line 14 - Print the field's name and its value

```
5  public class ReflectionExample {
6
7      public static void retrieveProperties(Object object) {
8
9          for (Field field : object.getClass().getDeclaredFields()) {
10             field.setAccessible(true);
11             Object value;
12             try {
13                 value = field.get(object);
14                 System.out.println(field.getName() + "=" + value);
15
16             } catch (IllegalArgumentException e) {
17                 e.printStackTrace();
18             } catch (IllegalAccessException e) {
19                 e.printStackTrace();
20             }
21
22         }
23     }
24 }
```

Check **the implementation!!!**

Reflection Techniques Examples

Problem: Propose a smart solution for constructing the DAO classes using Generics and Reflection

- Line 26 - Define a class AbstractDAO that defines the common operations for accessing a table: Insert, Update, Delete, FindById, FindAll. Define the operations on the specified generic type ***<T> (! T can be any Java Model Class that is mapped to the Database, and has the same name as the table and the same instance variables and data types as the table fields)***
- Line 29; Line 33 - For each AbstractDAO object obtain the class of the generic type T

```
26 public class AbstractDAO<T> {
27     protected static final Logger LOGGER = Logger.getLogger(AbstractDAO.class.getName());
28
29     private final Class<T> type;
30
31     @SuppressWarnings("unchecked")
32     public AbstractDAO() {
33         this.type = (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];
34
35     }
36
```

Check **the implementation!!!**

Reflection Techniques Examples

Problem: Propose a smart solution for constructing the DAO classes using Generics and Reflection – *findById method*

- Line 54- A generic query is build using the id field name *(! The value "id" can be replaced in a generic way by defining a custom annotation for PK and obtaining the name of the annotated field)*
- Line37-44- Build the generic select query using the name of the class previously obtained *(! The approach considers that the name of the Java model class is the same with the table in db. For a more generic approach, define a custom annotation for the table name, where the developer can specify a name different from the name of the class)*
- Line 56- the connection is obtain from the ConnectionFactory previously defined
- Line 58- The id is considered to be of type int. *(! For a more generic approach, the type of the PK can be specified in a generic way, similarly with the generic Type T)*
- Line 61- Using the ResultSet obtained after executing the statement, we will obtain the list of objects. Only the first element is of interest, since the db should contain only one entry with the specified id

```
37    private String createSelectQuery(String field) {
38        StringBuilder sb = new StringBuilder();
39        sb.append("SELECT ");
40        sb.append(" * ");
41        sb.append(" FROM ");
42        sb.append(type.getSimpleName());
43        sb.append(" WHERE " + field + " =?");
44        return sb.toString();
45    }
```

```
50    public T findById(int id) {
51        Connection connection = null;
52        PreparedStatement statement = null;
53        ResultSet resultSet = null;
54        String query = createSelectQuery("id");
55        try {
56            connection = ConnectionFactory.getConnection();
57            statement = connection.prepareStatement(query);
58            statement.setInt(1, id);
59            resultSet = statement.executeQuery();
60
61            return createObjects(resultSet).get(0);
62        } catch (SQLException e) {
63            LOGGER.log(Level.WARNING, type.getName() + "DAO:findById " + e.getMessage());
64        } finally {
65            ConnectionFactory.close(resultSet);
66            ConnectionFactory.close(statement);
67            ConnectionFactory.close(connection);
68        }
69        return null;
70    }
```

Check **the implementation!!!**

Reflection Techniques Examples

Problem: Propose a smart solution for constructing the DAO classes using Generics and Reflection – *findById method (cntd.)*

- Line 72 – Given a result set, obtain the list of model objects of type T
- Line 76 - For each *result* from the ResultSet
  - Line 77 – Create a new instance of type T
  - Line 78 - For each *field* of the class T
    - Line 79 - Retrieve from the current *result* the value of the current *field*
    - Line 80-81 – Obtain the method for setting a value to the *field*
    - Line 82 - using the obtained method set the value to the field

```
72    private List<T> createObjects(ResultSet resultSet) {
73        List<T> list = new ArrayList<T>();
74
75        try {
76            while (resultSet.next()) {
77                T instance = type.newInstance();
78                for (Field field : type.getDeclaredFields()) {
79                    Object value = resultSet.getObject(field.getName());
80                    PropertyDescriptor propertyDescriptor = new PropertyDescriptor(field.getName(), type);
81                    Method method = propertyDescriptor.getWriteMethod();
82                    method.invoke(instance, value);
83                }
84                list.add(instance);
85            }
86        } catch (InstantiationException e) {
```

Check **the implementation!!!**

Reflection Techniques Examples

Problem: Propose a smart solution for constructing the DAO classes using Generics and Reflection- *Usage*

- Line 5 – Define a class (StudentDAO) that extends the AbstractDAO and specify the model class used: Student
- The generic methods are directly accessed through inheritance
- Other specific methods can be implemented at this level

```
5  public class StudentDAO extends AbstractDAO<Student> {
6
7
8
9      //uses basic CRUD methods from superclass
10
11     //TODO: create only student specific queries
12 }
```

Check **the implementation!!!**

Source code

Download the source code from:

- Simple layered project: https://gitlab.com/utcn_dsrl/pt-layered-architecture
- Reflection example: https://gitlab.com/utcn_dsrl/pt-reflection-example