REFLECTION WORKSHOP

# Create a new project

File - > New Project -> Maven Project - > Set the lines below -> Next -> Set project name and location -> Finish

*GroupId: ro.teamnet.zth*

*ArtifcatId: ZTH*

*Version: 1.0-SNAPSHOT*

# Update pom.xml

In pom.xml file you will copy the code below:

<dependencies>

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.2.0</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

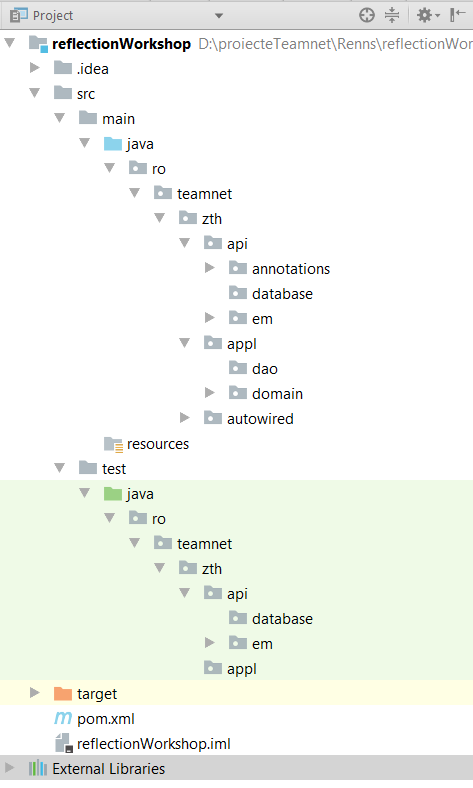
</plugin>

</plugins>

</build>

1. Create packages and folders

* in *src/main/java* create package *ro.teamnet.zth.api*;
* in package *ro.teamnet.zth.api* create the following folders: annotations, em, database;
* in *src/main/java* create package *ro.teamnet.zth.appl;*
* in package *ro.teamnet.zth.appl* create 2 folders: dao, domain;
* in *src/test/java* create package *ro.teamnet.zth.api* and *ro.teamnet.zth.appl*;
* in *ro.teamnet.zth.api* create the following folders: database, em;



1. Create annotations

Create in *src/main/java/ro/teamnet/zth/api/annotations* 3 annotations: “Table”, “Column”, “Id”.

**@Table** annotation:

* will be annotated with @Target(TYPE) and @Retention(RUNTIME);
* will have one method name() with default value “”;

**Example**:

@Target(TYPE)

@Retention(RUNTIME)

public @interface Table {

String name() default “”;

}

**@Column** annotation:

* will be annotated with @Target(FIELD) and @Retention(RUNTIME);
* will have one method name() with default value “”;

**@Id** annotation:

* will be annotated with @Target(FIELD) and @Retention(RUNTIME);
* will have one method name() with default value “id”;

# Create entities

In *src/main/java/ro/teamnet/zth/appl/domain* you will create 2 entities*: Department.java* and *Location.java.*

1. **Location.java**

* create a public class *Location.java* with the following private fields: id (Long), streetAddress (String), postalCode (String), city (String), stateProvince (String);
* generate getters and setters for fields above(ALT+INSERT -> Getter and Setter -> select all fields -> OK;
* override methods equals() and toString() (ALT+INSRT-> select equals()/toString()

-> select all fields -> OK

* annotate columns defined and the entity with the correct annotation

*Example*:

@Id(name = "location\_id")

private Long id;

! OBS: You need to set “name” with the name of DB field for Location entity

1. **Department.java**

* create a public class *Department.java,* with the following private fields: id (Long), departmentName (String), location (Location);
* create getters and setters for fields above;
* override methods equals() and toString() (ALT+INSRT-> select equals()/toString()

-> select all fields -> OK

* annotate columns defined and the entity with the correct annotation

*Example*:

@Id(name = "department\_id")

private Long id;

! OBS: You need to set “name” with the name of DB field for Location entity

1. Create **Job.java** and **Employee.java**mapped on database columns that you did yesterday.
2. Create utilities classes

* create in *src/main/java/ro/teamnet/zth/api/em* an enum(ref: <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html> ) *QueryType* with 4 values: *SELECT, INSERT, UPDATE, DELETE;*
* create in same folder a class *ColumnInfo* with following private fields: *columnName (type String), columnType (type Class), dbColumnName (type String), isId (type boolean), value (type Object).* Create getters and setters for these fields.
* create in same folder class *Condition* with following private fields: *columnName (type String), value (type Object).* Create getters and setters for these fields*;*

1. Create EntityUtils.java file (it’s a helper file used for getting information from an entity/class by annotations)

Create in *src/main/java/ro/teamnet/zth/api/em* a public class *EntityUtils.java* with the following methods:

* create a private constructor *EntityUtiles()*, which will throw a new *UnsupportedOperationException()*;
* create a public static method *getTableName (Class entity)* which will return the name of the DB table. In this method you will return DB table name from annotation (@Table) or entity name if there is no annotation. You will do that using Reflection;
* create a public static method *getColumns* (Class entity) which will return a list of ColumnInfo. In this method you will return a list of information about columns annotated with @Column. Steps:
* get all declared fields from class;
* go through each field and verify if it is annotated with @Column or @Id. If so, create a ColumnInfo object, set its fields (without value), and populate ColumnInfo list;
* return list;
* create a public static method *castFromSqlType*(Object value, Class wantedType), where value is the value from DB and wantedType is the type of value which you want to use. The return type is Object. This method is used to cast the type of “id” column from DB to id field from the entities. Steps:
* if value is BigDecimal and wantedType is Integer then you will return an Interger;
* if value is BigDecimal and wantedType is Long then you will return an Long;
* if value is BigDecimal and wantedType is Float then you will return an Float;
* if value is BigDecimal and wantedType is Double then you will return an Double;
* if value is different from BigDecimal then the method will return that value;
* create a public static method *getFieldsByAnnotations*(Class clazz, Class annotation), where clazz is for example Department and annotation is @Column. The return type is a list of fields. Steps:
* get declared fields for class “clazz”;
* search fields with the given annotation, and add the field in the list;
* return list;
* create a public static method *getSqlValue*(Object object), which will return an Object. Steps:
* if object class is annotated with @Table, get the field annotated with @Id, set it accessible, and return the object associated with the id field;
* if object class is not annotated with @Table, return the object

1. Create test class for EntityUtils.java

In *src/test/java/ro/teamnet/zth/api/em,* create *EntityUtilsTest.java* which will test all the methods defined in *EntityUtils.java.* Create 2 tests for each method.

*Example:*

*@Test*

*public void testGetTableNameMethod() {*

*String tableName = EntityUtils.getTableName(Department.class);*

*assertEquals("Table name should be departments!", "departments", tableName);*

*}*

# Create utility class used for generating SQL queries

Create in *src/main/java/ro/teamnet/zth/api/em* a public class *QueryBuilder.java* with following fields and methods:

* create a public method getValueForQuery(Object value) which returns a string object. If object type is String then return the value, between ‘’. If the object type is Date, then use:

DateFormat dateFormat = newSimpleDateFormat("mm/dd/yyyy");  
return "TO\_DATE('"+dateFormat.format((Date)value)+"','mm-dd-YYYY'"; If value is some other type, return value.toString().

* create private fields *tableName(type Object), queryColumns(type List<ColumnInfo>), queryType(type QueryType), conditions(type List<Condition>);*
* create a public method *addCondition(Condition condition)* which will return a *QueryBuilder (the current object),* and will add the given conditions to the already existing list of conditions necessary for a query (from the current object);
* create a public method *setTableName(Object tableName)* which will return a *QueryBuilder* (the current object) and will set the table name necessary for a query;
* create a public method *addQueryColumns(List<ColumnInfo> queryColumns)* which will return a *QueryBuilder* and will add the given query columns to the already existing list of query columns (from the current object);
* **create a public method *setQueryType(QueryType queryType)* which will return a *QueryBuilder* and will set the type of the query;**
* create 4 private methods: *createSelectQuery()*, *createDeleteQuery(), createUpdateQuery(), createInsertQuery(),* that will return a String object representing an SQL query*.* You will do this using StringBuilder class (ref: <https://docs.oracle.com/javase/tutorial/java/data/buffers.html> ), and all the properties from the current object (QueryBuilder): the query type, the table name, the query columns and the conditions of the query.
  + **!!! Use the getValueForQuery() method to set the condition values of the query;**
  + **!!! Be careful when adding the query columns (must be separated by , ) and the query conditions (don’t forget the WHERE clause);**
* create public method *createQuery()* which will return a String. In this method you will check the query type (using QueryType enum), and call one of the 4 methods defined above;

# Create test class for QueryBuilder.java

In *src/test/java/ro/teamnet/zth/api/em,* create *QueryBuilderTest.java* which will test all the methods defined in *QueryBuilder.java.* Create 2 tests for each of the following methods: createQuery(), createInsertQuery(), createUpdateQuery(), createDeleteQuery(), createSelectQuery().

# Implementation of @Autowired annotation in Spring framework

Read about @Autowired annotation in Spring framework, and understand what is does, so that you can make your own @Autowired annotation:

<https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/beans/factory/annotation/Autowired.html>

<https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm>

Create in *src/main/java/ro/teamnet/zth* another package named *autowired*, in which you will place all the classes needed for this implementation.

* Create a @MyAutowired annotation that has a value() method and class fields as target;
* Create a @MyQualifier annotation that has a value() method and classes as target;
* Create an interface with one void method;
* Create 2 classes that implement the interface above, and override the method such that each class has its own implementation of it (print some specific text);
* Give a qualifier to one of the classes created above, using @MyQualifier annotation;
* Create another class that has 2 properties of the types of the classes created by you. Annotate both of the properties with @MyAutowired annotation, but give a value to only one of them;
* Create a *AutowiredUtils.java* class that has 2 properties (Map type). One of them will store the program instances by their type(class), and another one will store the program instances by their qualifier;
* Create 2 methods, one that updates the type dictionary with a given object, and one that updates the qualifier dictionary with a given object;
* Create 2 methods that gets an object from the dictionaries by type(class) and by qualifier name, both received as arguments;
* Create a method that handles the autowiring fields of a given object; in this method, get the autowired fields and set their value with the one in the dictionaries;
* Call the method that updates the qualifier dictionary from the constructor of the class that’s annotated with @MyQualifier;
* Call the method that updates the type dictionary from the constructor of the other class;
* Call the method that handles that autowired fields from the constructor of the class that has them;
* Create a *Main.java* class, with a main() method in which you create an instance of the 2 classes that implement your interface;
* Create 2 instances of the class that autowires the other classes;
* Call the inherited method from the autowired fields from both instances; What do you observe?