

# Relational Databases with MySQL Week 11 Assignment

Points possible: 70

Category	Criteria	% of Grade
Functionality	Does the code work?	25
Organization	Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear.	25
Creativity	Student solved the problems presented in the assignment using creativity and out of the box thinking.	25
Completeness	All requirements of the assignment are complete.	25

**Instructions:** Complete the coding steps. Take screenshots of the code and of the running program (make sure to get screenshots of all required functionality) and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document to the repository. Additionally, push the Java project to the same repository. Add the URL for this week's repository to this document where instructed and submit this document to your instructor when complete.

## Coding Steps:

1. Create a class of whatever type you want (Animal, Person, Camera, Cheese, etc.).
  - a. Do not implement the Comparable interface.
  - b. Add a name instance variable so that you can tell the objects apart.
  - c. Add getters, setters and/or a constructor as appropriate.
  - d. Add a toString method that returns the name and object type (like "Pentax Camera").
  - e. Create a static method named `compare` in the class that returns an int and takes two of the objects as parameters. Return -1 if parameter 1 is "less than" parameter

2. Return 1 if parameter 1 is "greater than" parameter 2. Return 0 if the two parameters are "equal".
  - f. Create a static list of these objects, adding at least 4 objects to the list.
  - g. In another class, write a method to sort the objects using a Lambda expression using the compare method you created earlier.
  - h. Write a method to sort the objects using a Method Reference to the compare method you created earlier.
  - i. Create a main method to call the sort methods.
  - j. Print the list after sorting (System.out.println).
2. Create a new class with a main method. Using the list of objects you created in the prior step.
    - a. Create a Stream from the list of objects.
    - b. Turn the Stream of object to a Stream of String (use the map method for this).
    - c. Sort the Stream in the natural order. (Note: The String class implements the Comparable interface, so you won't have to supply a Comparator to do the sorting.)
    - d. Collect the Stream and return a comma-separated list of names as a single String. Hint: use Collectors.joining(", ") for this.
    - e. Print the resulting String.
3. Create a new class with a main method. Create a method (method a) that accepts an Optional of some type of object (Animal, Person, Camera, etc.).
    - a. The method should return the object unwrapped from the Optional if the object is present. For example, if you have an object of type Cheese, your method signature should look something like this:

```
public Cheese cheesyMethod(Optional<Cheese> optionalCheese) {...}
```
    - b. The method should throw a NoSuchElementException with a custom message if the object is not present.
    - c. Create another method (method b) that calls method a with an object wrapped by an Optional. Show that the object is returned unwrapped from the Optional (i.e., print the object).
    - d. Method b should also call method a with an empty Optional. Show that a NoSuchElementException is thrown by method a by printing the exception

message. Hint: catch the `NoSuchElementException` as parameter named "e" and do `System.out.println(e.getMessage())`.

- e. Note: your method should handle the `Optional` as shown in the video on `Optionals` using the `orElseThrow` method. For the missing object, you must use a `Lambda` expression in `orElseThrow` to return a `NoSuchElementException` with a custom message.

### Screenshots of Code:

```
1 package reactions;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 // Create a class of whatever type you want (Animal, Person, Camera, Cheese, etc.). Do not implement the
7 // Comparable interface.
8 public class Severe {
9
10     private String severe;
11
12     // Add a name instance variable so that you can tell the objects apart. Create a static list of these objects,
13     // adding at least 4 objects to the list.
14     public static List<Severe> severity = new ArrayList<> (List.of(
15         new Severe("esophageal stricture"),
16         new Severe("esophageal ulceration"),
17         new Severe("tardive dyskinesia"),
18         new Severe("laryngeal edema")));
19
20     public Severe(String severe) { this.severe = severe; }
```

```
21
22 // Add getters, setters and/or a constructor as appropriate.
23 public static List<Severe> getSeverity() {
24     return severity;
25 }
26
27 // Add a toString method that returns the name AND object type (like "Pentax Camera").
28 @Override
29 public String toString() {
30     return ("Severe Adverse Reaction: " + severe);
31 }
32
33 // Create a static method named compare in the parameters. Return -1 if parameter 1 is "less than" parameter
34 // 2. Return 1 if parameter 1 is "greater than" parameter 2. Return 0 if the two parameters are "equal".
35 public static int compare(Severe s1, Severe s2) { return s1.severe.compareTo(s2.severe); }
36
37
38 }
```

```

1 package reactions;
2
3 import java.util.List;
4
5 public class Sort {
6     // Create a main method to call the sort methods.
7     public static void main(String[] args) {
8         sortWithLambda();
9         sortWithMethodReference();
10    }
11
12    public static void sortWithLambda() {
13        // Write a method to sort the objects using a Lambda expression using the compare method you created earlier.
14        // Print the list after sorting (System.out.println).
15        List<Severe> severeList = Severe.getSevere();
16        severeList.sort((s1, s2) -> Severe.compare(s1, s2));
17        System.out.println("Lambda Sort: " + severeList);
18    }
19
20 }

```

```

19 public static void sortWithMethodReference() {
20     // Write a method to sort the objects using a Method Reference to the compare method you created earlier.
21     // Print the list after sorting (System.out.println).
22     List<Severe> severeList = Severe.getSevere();
23     severeList.sort(Severe::compare);
24     System.out.println("Method Reference Sort: " + severeList);
25 }
26 }

```

```

1 package reactions;
2
3 import java.util.stream.Collectors;
4
5 // Create a new class with a main method. Using the list of objects you created in the prior step.
6 public class Stream {
7
8     // note to self: don't need run method
9     public static void main(String[] args) {
10         new Stream().run();
11     }
12

```

```

13     private void run() {
14         // Create a Stream from the list of objects. Print the resulting String.
15         System.out.println(Severe.getSevere().stream().map(Severe::toString).collect(Collectors.joining(", ")));
16         // Turn the Stream of object to a stream of String (use the map method for this).
17         .map(Severe::toString).collect(Collectors.joining(", "));
18         // Sort the Stream in the natural order. (Note: The String class implements the Comparable interface, so you won't
19         // have to supply a Comparator to do the sorting.)
20         .sorted().collect(Collectors.joining(", "));
21         // Collect the Stream and return a comma-separated list of names as a single String. Hint: use Collectors.joining
22         // (" ", ", ") for this.
23         .collect(Collectors.joining(", "));
24     }

```

```

1 package reactions;
2
3 import java.util.NoSuchElementException;
4
5 // Create a new class with a main method.
6 public class Optional {
7
8     public static void main(String[] args) { new Optional().run(); }
9
10
11 // Create another method (Method B/Run) that calls Method A/Severe Method with an object wrapped by an Optional.
12 // Show that the object is returned unwrapped from the Optional (i.e., print the object).
13
14

```

```

15 private void run() {
16     Severe severe = severeMethod(java.util.Optional.of(new Severe("blaspharospas")));
17     System.out.println("Uh oh " + severe + " shouldn't be a severe reaction!");
18
19     try {
20 // Method B/Run should also call Method A/Severe with an empty Optional. Show that a NoSuchElementException is
    thrown
21 // by Method A/Severe by printing the exception message. Hint: catch the NoSuchElementException as parameter named
    "e"
22 // and do System.out.println(e.getMessage()).
23     severeMethod(severeOptional: java.util.Optional.empty());
24     } catch (Exception e) {
25         System.out.println(e.getMessage());
26     }
27 }
28

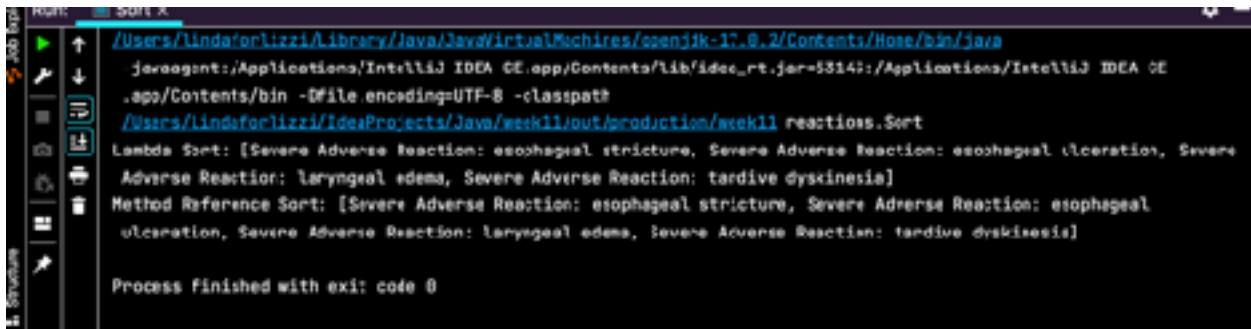
```

```

29 // Create a method (Method A/Severe Method) that accepts an Optional of some type of Animal, Person, Camera, etc.)
30 // The method should return the object unwrapped from the Optional if the object is present. For example, if you
    have an object of type Cheese, your method signature should look something like this: public Cheese cheesyMethod
    (Optional<Cheese> optionalCheese) {...}
31 private Severe severeMethod (java.util.Optional<Severe> severeOptional) {
32 // Note: your method should handle the Optional as shown in the video on Optionals using the orElseThrow method.
    For the missing object, you must use a Lambda expression in orElseThrow to return a NoSuchElementException with a
    custom message.
33     return severeOptional.orElseThrow(
34 // The method should throw a NoSuchElementException with a custom message if the object is not present.
35         () -> new NoSuchElementException("That adverse reaction does not exist!");
36     );
37 }
38

```

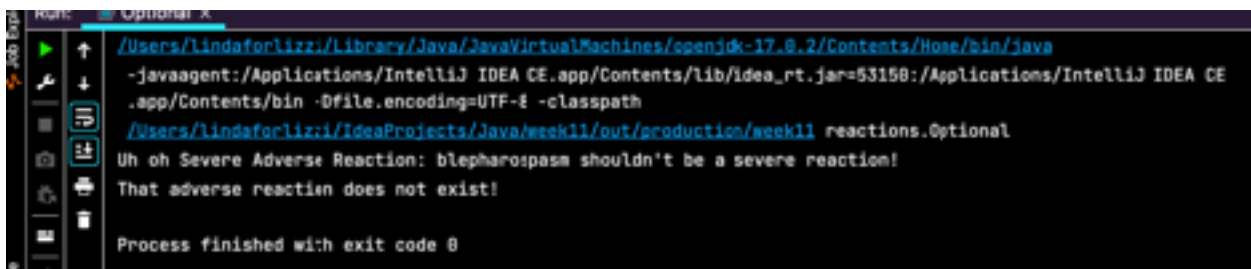
## Screenshots of Running Application Results:



```
Run: Sort x
/Users/lindaforlizzi/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=5314:/Applications/IntelliJ IDEA CE
.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/lindaforlizzi/IdeaProjects/Java/week11/out/production/week11 reactions.Sort
Lambda Sort: [Severe Adverse Reaction: esophageal stricture, Severe Adverse Reaction: esophageal ulceration, Severe
Adverse Reaction: laryngeal edema, Severe Adverse Reaction: tardive dyskinesia]
Method Reference Sort: [Severe Adverse Reaction: esophageal stricture, Severe Adverse Reaction: esophageal
ulceration, Severe Adverse Reaction: laryngeal edema, Severe Adverse Reaction: tardive dyskinesia]
Process finished with exit code 0
```



```
Run: Stream x
/Users/lindaforlizzi/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=53147:/Applications/IntelliJ IDEA CE
.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/lindaforlizzi/IdeaProjects/Java/week11/out/production/week11 reactions.Stream
Severe Adverse Reaction: esophageal stricture, Severe Adverse Reaction: esophageal ulceration, Severe Adverse
Reaction: laryngeal edema, Severe Adverse Reaction: tardive dyskinesia
Process finished with exit code 0
```



```
Run: Optional x
/Users/lindaforlizzi/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=53150:/Applications/IntelliJ IDEA CE
.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/lindaforlizzi/IdeaProjects/Java/week11/out/production/week11 reactions.Optional
Uh oh Severe Adverse Reaction: blepharospasm shouldn't be a severe reaction!
That adverse reaction does not exist!
Process finished with exit code 0
```

URL to GitHub Repository: <https://github.com/thisLinda/SQLWeek11Assignment>