

## 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:

- Create a class of whatever type you want (Animal, Person, Camera, Cheese, etc.).
  - Do not implement the Comparable interface.
  - Add a name instance variable so that you can tell the objects apart.
  - Add getters, setters and/or a constructor as appropriate.
  - Add a toString method that returns the name and object type (like "Pentax Camera").
  - 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".
  - Create a static list of these objects, adding at least 4 objects to the list.
  - In another class, write a method to sort the objects using a Lambda expression using the compare method you created earlier.
  - Write a method to sort the objects using a Method Reference to the compare method you created earlier.
  - Create a main method to call the sort methods.
  - Print the list after sorting (System.out.println).
- Create a new class with a main method. Using the list of objects you created in the prior step.
  - Create a Stream from the list of objects.
  - Turn the Stream of object to a Stream of String (use the map method for this).
  - 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.)
  - Collect the Stream and return a comma-separated list of names as a single String. Hint: use Collectors.joining(", ") for this.
  - Print the resulting String.
- 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.).
  - 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) {...}
```

- The method should throw a NoSuchElementException with a custom message if the object is not present.
- 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).

- 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()).
- 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:**

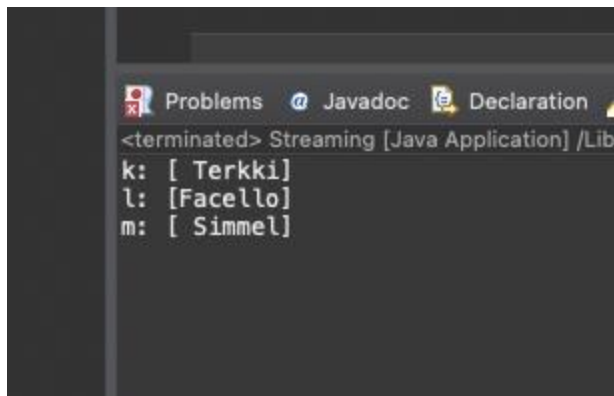
```
Streaming.java MyApp.java Optional.java
1 package assignment1;
2
3 import java.util.List;
4
5 public class MyApp {
6
7     private SortedAnimals sortAnml = new SortedAnimals();
8
9
10    public static void main(String[] args) {
11        new MyApp().run();
12    }
13
14
15    private void run() {
16        List<Animal> animals = sortAnml.getAnimal(SortType.METHOD_REFERENCE);
17        print(animals, SortType.METHOD_REFERENCE);
18    }
19
20    private void print(List<Animal> animals, SortType type) {
21        switch(type) {
22
23            case LAMBDA:
24                animals.forEach(animal -> System.out.println(animal.getBreed()));
25                break;
26
27            case METHOD_REFERENCE:
28                animals.forEach(System.out::println);
29                break;
30
31            default:
32                break;
33
34        }
35    }
36
37 }
38
```

```
Streaming.java MyApp.java Optional.java
1 package assignment2;
2 import java.util.List;
3
4
5
6
7 public class Streaming {
8     private RepeatService repeatService = new RepeatService();
9
10    public static void main(String[] args) {
11        new Streaming().run();
12    }
13
14
15    private void run() {
16
17        //group by repeating character goes into a map
18
19        Map<String, List<RepeatingName>> names = repeatService.findRepeatingNames();
20
21        names.entrySet().forEach(entry ->
22            System.out.println(entry.getKey() + ": " +
23                entry.getValue().stream()
24                    .map(RepeatingName::getName)
25                    .collect(Collectors.toList())));
26    }
27
28 }
29
```

```
Streaming.java  MyApp.java  Optionals.java X
1 import java.util.NoSuchElementException;
2
3
4 public class Optionals {
5
6     private Scanner scanner = new Scanner(System.in);
7     private OptionalService service = new OptionalService();
8
9     public static void main(String[] args) {
10
11         new Optionals().run();
12
13     }
14
15     private void run() {
16
17         boolean done = false;
18
19         while(!done) {
20             System.out.print("Enter something: ");
21             String search = scanner.nextLine();
22
23             if (search.isEmpty()) {
24                 done = true;
25             } else {
26                 try {
27                     String found = service.find(search);
28                     System.out.println("I found " + found + "! ");
29                 }
30                 catch(NoSuchElementException e) {
31                     System.out.println(e.getMessage());
32                 }
33             }
34         }
35     }
36 }
37
38 }
39
```

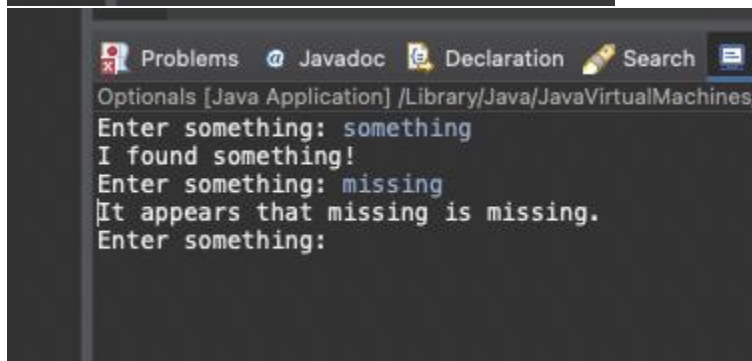
Screenshots of Running Application Results:

```
Problems  Javadoc  Declaration
<terminated> MyApp (5) [Java Application] /
Axolotl
Cat
Dog
Lion
```



This screenshot shows a console window from an IDE. The title bar includes icons for Problems, Javadoc, and Declaration. The text in the console indicates a terminated Java application named 'Streaming'. It displays three variables: 'k' with the value 'Terkki', 'l' with the value 'Facello', and 'm' with the value 'Simmel'.

```
<terminated> Streaming [Java Application] /Lib  
k: [ Terkki]  
l: [Facello]  
m: [ Simmel]
```



This screenshot shows a console window from an IDE. The title bar includes icons for Problems, Javadoc, Declaration, Search, and a message icon. The text in the console shows a Java application named 'Optionals' running. It prompts the user to 'Enter something:', and the user enters 'something'. The application responds with 'I found something!'. It then prompts 'Enter something:' again, and the user enters 'missing'. The application responds with 'It appears that missing is missing.' and then prompts 'Enter something:' a third time.

```
Optionals [Java Application] /Library/Java/JavaVirtualMachines/  
Enter something: something  
I found something!  
Enter something: missing  
It appears that missing is missing.  
Enter something:
```

URL to GitHub Repository:

<https://github.com/Asosa809/Week11>