

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:

```

1 package animalsort;
2
3 import java.util.List;
4
5 public class AnimalSort {
6     private AnimalService animalService = new AnimalService();
7
8     public static void main(String[] args) {
9         new AnimalSort().run();
10    }
11
12    private void run() {
13        List<Animals> animal = animalService.getAnimal(AnimalType.LAMBDA);
14        print(animal, AnimalType.LAMBDA);
15    }
16
17    private void print(List<Animals> animal, AnimalType type) {
18        switch(type) {
19            case LAMBDA:
20                animal.forEach(animals -> System.out.println(animals.getAnimal()));
21                break;
22            case METHOD_REFERENCE:
23                animal.forEach(System.out::println);
24                break;
25            case ANONYMOUS_INNER_CLASS:
26            case NORMAL_CLASS:
27                for(Animals animals : animal) {
28                    System.out.println(animals.getAnimal());
29                }
30                break;
31            default:
32                break;
33        }
34    }
35
36 }

```

```

1 package animalDao;
2
3 import java.util.ArrayList;
4
5 public class AnimalDao {
6     List<Animals> animals = new ArrayList<> (List.of(
7         new Animals("Dog"),
8         new Animals("Cat"),
9         new Animals("Lizard"),
10        new Animals("Fish"),
11        new Animals("Monkey")));
12
13     public List<Animals> getAnimal() {
14         return animals;
15     }
16 }

```

```

1 package animal.service;
2
3 import java.util.Comparator;
4
11
12 public class AnimalService {
13     private static AnimalDao animalDao = new AnimalDao();
14
15     public List<Animals> getAnimal(AnimalType type) {
16         List<Animals> animals = animalDao.getAnimal();
17         Comparator<Animals> comp = null;
18
19         switch(type) {
20             case ANONYMOUS_INNER_CLASS:
21                 comp = new Comparator<Animals>() {
22
23                     @Override
24                     public int compare(Animals p1, Animals p2) {
25                         return Animals.compareAnimals(p1, p2);
26                     }
27
28                 };
29                 break;
30
31             case LAMBDA:
32                 comp = (p1, p2) -> Animals.compareAnimals(p1, p2);
33
34                 break;
35
36             case METHOD_REFERENCE:
37                 comp = Animals ::compareAnimals;
38                 break;
39
40             case NORMAL_CLASS:
41                 comp = new AnimalSort();
42                 break;
43
44             default:
45                 throw new RuntimeException("Not a Animal Type" + type);
46
47         }
48
49         animals.sort(comp);
50         return animals;
51     }
52
53     static class AnimalSort implements Comparator<Animals> {
54
55         @Override
56         public int compare(Animals p1, Animals p2) {
57             // TODO Auto-generated method stub
58
59             return Animals.compareAnimals(p1, p2);
60         }
61     }
62 }

```

```

1 package animal.model;
2
3 public class Animals {
4     private String animals;
5
6     public Animals(String animalname) {
7         this.animals = animalname;
8     }
9
10    public Animals(Animals animals2) {
11        return;
12    }
13
14    @Override
15    public String toString() {
16        return animals;
17    }
18
19    public String getAnimal() {
20        return animals;
21    }
22
23    public static int compareAnimals(Animals p1, Animals p2) {
24        return p1.animals.compareTo(p2.animals);
25    }
26
27
28 }
29

```

```

1 package animal;
2
3 public enum AnimalType {
4
5     NORMAL_CLASS, ANONYMOUS_INNER_CLASS, LAMBDA, METHOD_REFERENCE
6 }
7

```

Stream:

```

1 package animalStream;
2
3 import java.io.IOException;
4
5
6
7 public class AnimalStream {
8
9     public static void main(String[] args) throws IOException {
10         Stream<String> streamOfString = Stream.of("Cat", "Dog", "Lizard", "Fish", "Monkey");
11
12         streamOfString = Stream.of("Meow", "Woof", "Hiss", "Bloop", "Monke");
13         String listOfStream = streamOfString.collect(Collectors.joining(", "));
14         System.out.println("The animals sounds like this : " + listOfStream);
15
16     }
17
18 }
19
20

```

Optional :

```

1 package animal.optional.dao;
2
3 import java.util.Optional;
4
5 public class OptionalDao {
6 public Optional<String> find(String search) {
7     if("Animal".equals(search)) {
8         return Optional.empty();    }
9     return Optional.of(search);
10 }
11
12 }
13

```

```

1 package animal.optional;
2
3 import java.util.NoSuchElementException;
4
5
6
7 public class AnimalOPService {
8     private OptionalDao dao = new OptionalDao();
9
10 public String find(String search) {
11
12     return dao.find(search).orElseThrow(() -> new NoSuchElementException("The keyword " + search + " is not found"));
13
14 }
15
16 }
17

```

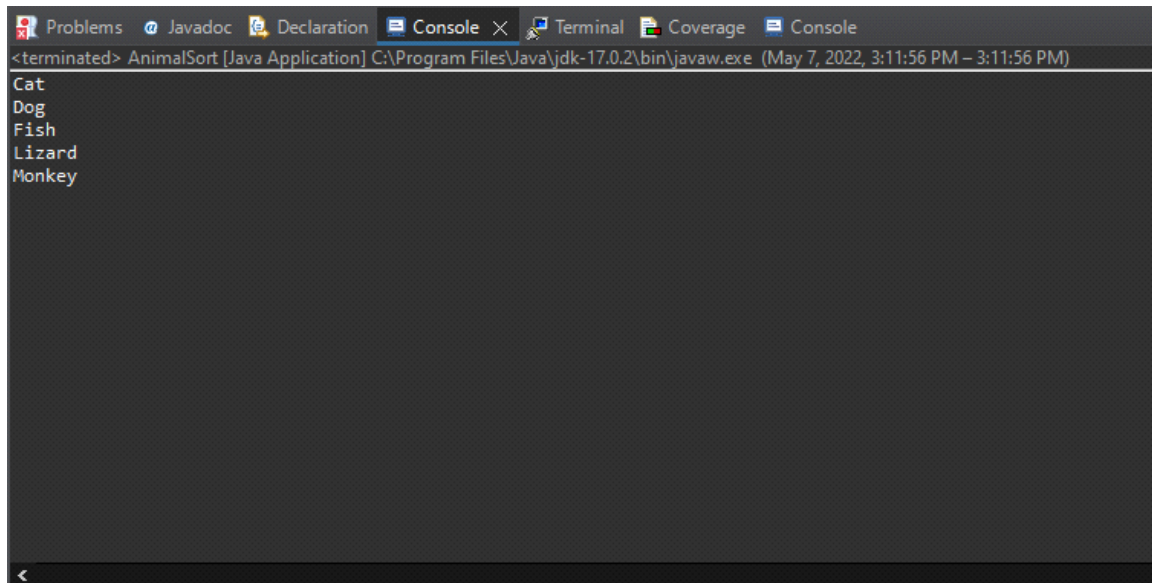
```

1 package animal.optional;
2
3 import java.util.NoSuchElementException;
4
5
6 public class AnimalOptional {
7     private Scanner scanner = new Scanner(System.in);
8     private AnimalOPService service = new AnimalOPService();
9     public static void main(String[] args) {
10         new AnimalOptional().run();
11     }
12
13     private void run() {
14         boolean done = false;
15         while(!done) {
16             System.out.print("Enter Optional : ");
17             String search = scanner.nextLine();
18
19             if(search.isEmpty()) {
20                 done = true;
21             }
22             else {
23                 try {
24                     String found = service.find(search);
25                     System.out.println("I found " + found + ".");
26                 }
27                 catch(NoSuchElementException e) {
28                     System.out.println(e.getMessage());
29                 }
30             }
31         }
32     }
33 }
34
35
36
37
38
39
40

```

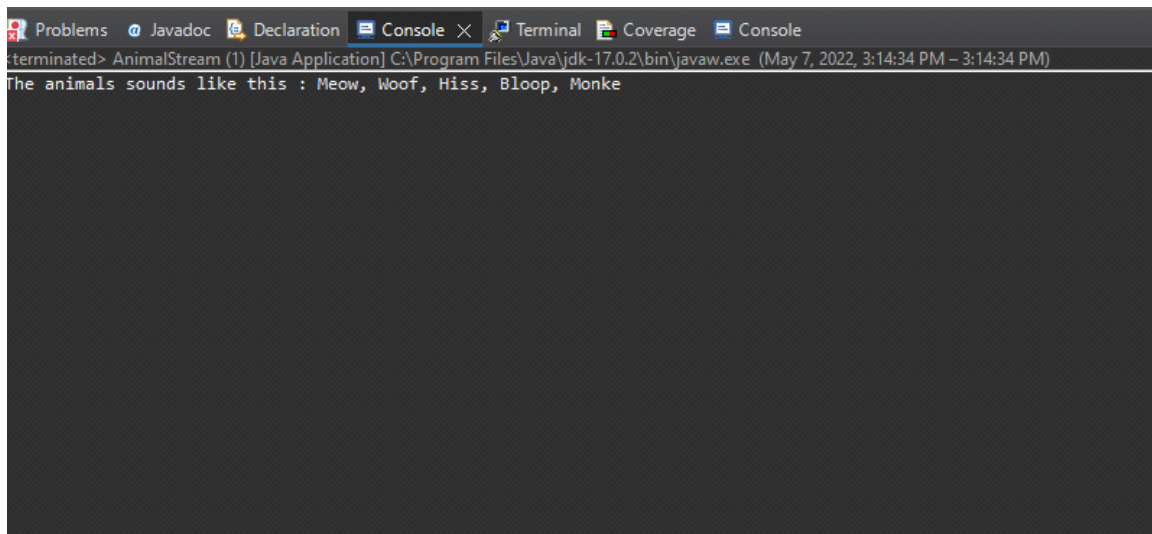
Screenshots of Running Application Results:

First Step: Lambda and Method Reference



```
Problems Javadoc Declaration Console X Terminal Coverage Console
<terminated> AnimalSort [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (May 7, 2022, 3:11:56 PM – 3:11:56 PM)
Cat
Dog
Fish
Lizard
Monkey
```

Stream:



```
Problems Javadoc Declaration Console X Terminal Coverage Console
<terminated> AnimalStream (1) [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (May 7, 2022, 3:14:34 PM – 3:14:34 PM)
The animals sounds like this : Meow, Woof, Hiss, Bloop, Monke
```

Optional:

```
Problems Javadoc Declaration Console X Terminal Coverage Console
AnimalOptional [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (May 7, 2022, 3:16:01 PM)
Enter Optional : Oof
I found Oof.
Enter Optional :
```

```
AnimalOptional [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (May 7, 2022, 3:16:52 PM)
Enter Optional : Animal
The keyword Animal is not found
Enter Optional :
```

URL to GitHub Repository: <https://github.com/Starssk1ttles/Week11Sql>