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:

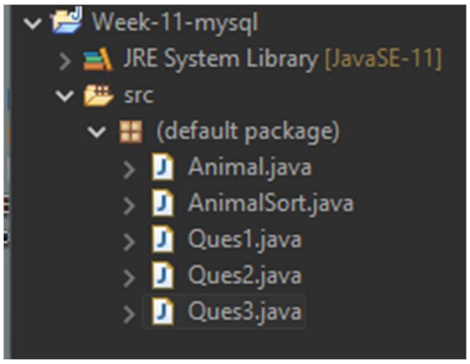
1. Create a class of whatever type you want (Animal, Person, Camera, Cheese, etc.).
    a. Do not implement the Comparable interface.
    b. Add a name instance variable so that you can tell the objects apart.
    c. Add getters, setters and/or a constructor as appropriate.
    d. Add a toString method 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:



Animal.java

```java
public class Animal {
    private String name;
    private int lifeSpan;


    //constructor
    public Animal(String name, int lifeSpan) {
        this.name = name;
        this.lifeSpan = lifeSpan;
    }


    //getters and setters
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getLifeSpan() {
        return lifeSpan;
    }

    public void setLifeSpan(int lifeSpan) {
        this.lifeSpan = lifeSpan;
    }


    //Add a toString method that returns the name and object type
    public String toString() {
        return getName() +" "+ this.getClass().getSimpleName();
    }

     //Create a static method named compare in the class that returns an int and takes two of the objects as parameters.
    public static int compare(Animal p1, Animal p2) {
        int p1LifeSpan = p1.getLifeSpan();
        int p2LifeSpan = p2.getLifeSpan();

        if(p1LifeSpan < p2LifeSpan) {
            return -1;
        } else if(p1LifeSpan >p2LifeSpan) {
            return 1;
        }

        return 0;
    }

}
```

AnimalSort.java

```java
import java.util.Arrays;

public class AnimalSort {

    //lambda expression
    public static void sortAnimalsByLifeSpan(Animal[] animalsAsArray) {
        Arrays.sort(animalsAsArray, (Animal a1, Animal a2) -> Animal.compare(a1, a2));
        displaySortedList(animalsAsArray);
    }

    //method reference
    public static void sortAnimalsByLifeSpanMethodReference(Animal[] animalsAsArray) {
        Arrays.sort(animalsAsArray, Animal::compare);
        displaySortedList(animalsAsArray);
    }

    //display sorted data
    public static void displaySortedList(Animal[] animalsAsArray) {
        for (int i = 0; i < animalsAsArray.length; i++) {
            System.out.println(animalsAsArray[i] + " ");
        }

    }

}

```

Ques1.java

```java
import java.util.ArrayList;

public class Ques1 {

    public static ArrayList<Animal> animals = new ArrayList<Animal>();

    public static void main(String[] args) {

    //adding at least 4 objects to the list
        animals.add(new Animal("Dog",22));
        animals.add(new Animal("Crocodile",45));
        animals.add(new Animal("Deer",35));
        animals.add(new Animal("Cat",25));

     System.out.println("Before Sorting the Animal data:");


     animals.forEach((s)-> System.out.println(s));

     Animal[] animalsAsArray =  animals.toArray(new Animal[animals.size()]);

    //sort using lambda expression
    System.out.println("\nAfter Sorting the Animal data by LifeSpan via LAMBDA EXPRESSION:");
    AnimalSort.sortAnimalsByLifeSpan(animalsAsArray);


    //sort using method reference
    System.out.println("\nAfter Sorting the Animal data by LifeSpan via METHOD REFERENCE:");
    AnimalSort.sortAnimalsByLifeSpanMethodReference(animalsAsArray);


    }

}

```

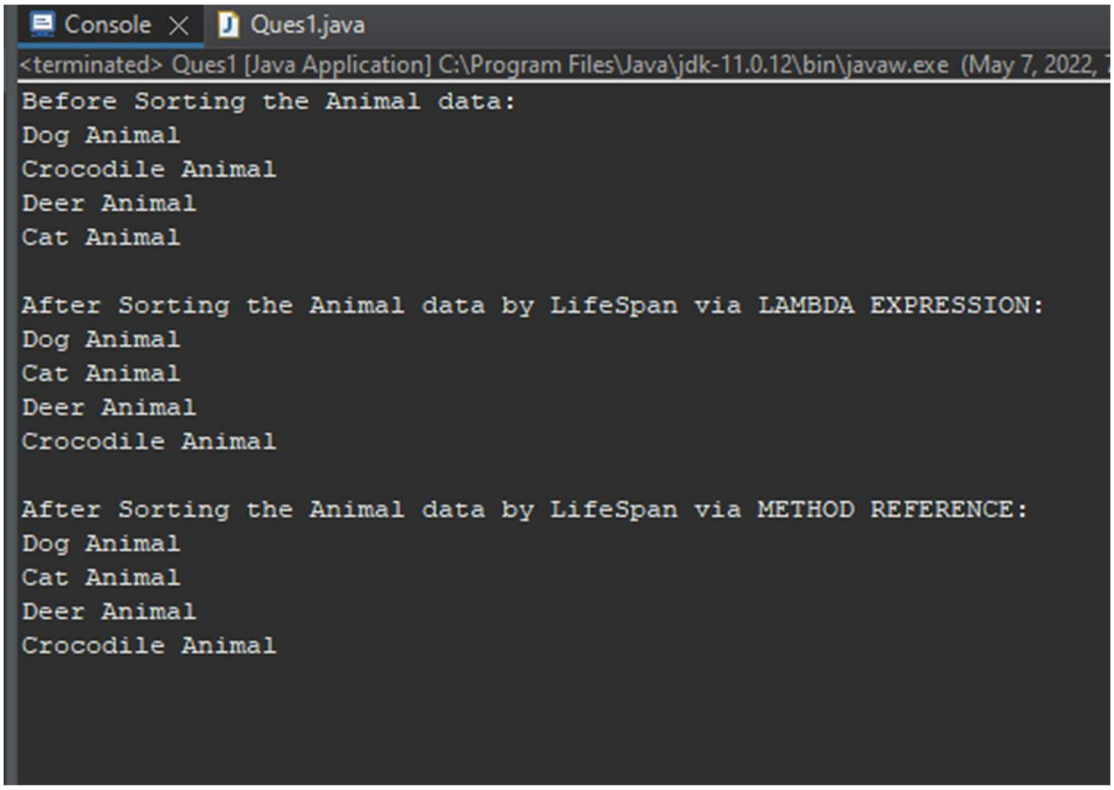
Ques2.java

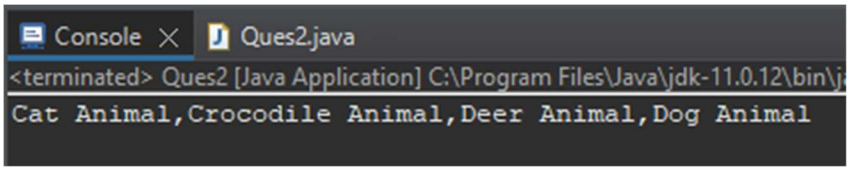
```java
import java.util.ArrayList;

public class Ques2 {


    public static ArrayList<Animal> animals = new ArrayList<Animal>();

    public static void main(String[] args) {

        //adding at least 4 objects to the list
        animals.add(new Animal("Dog",22));
        animals.add(new Animal("Crocodile",45));
        animals.add(new Animal("Deer",35));
        animals.add(new Animal("Cat",25));

        //Turn the Stream of object to a Stream of String (use the map method for this)
        String listOfStrings = animals.stream() // //Create a Stream from the list of object
                .map(String::valueOf) //Turn the Stream of object to a Stream of String (use the map method for this)
                .sorted() //Sort the Stream in the natural order
                .collect(Collectors.joining(",")); //Collect the Stream and return a comma-separated list of names as a single String

        System.out.println(listOfStrings); //Print the resulting String.

    }
}
```

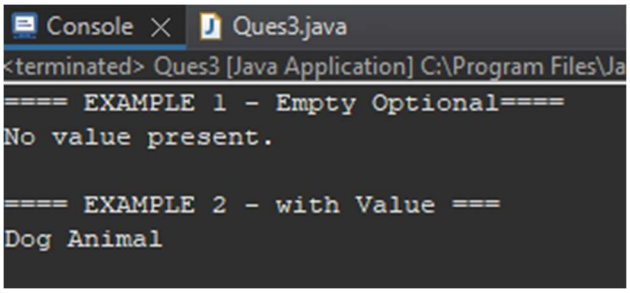
Ques3.java

```java
import java.util.NoSuchElementException;

public class Ques3 {

    public static void main(String[] args) {

        System.out.println("==== EXAMPLE 1 - Empty Optional====");
        Optional<Animal> animalEmpty = Optional.empty();
        animalMethodB(animalEmpty);  //Method b should also call method a with an empty Optional


        System.out.println("\n==== EXAMPLE 2 - with Value ===");
        Animal al = new Animal("Dog", 22);
        Optional<Animal> animalOptional = Optional.of(al);
        animalMethodB(animalOptional);

    }


    public static Animal animalMethodA(Optional<Animal> optionalAnimal) {

        optionalAnimal.orElseThrow(() -> new NoSuchElementException("No value present."));
        return optionalAnimal.get();
    }


    public static void animalMethodB(Optional<Animal> optionalAnimal) {
           try {
                System.out.println(animalMethodA(optionalAnimal));
            }
            catch (NoSuchElementException e) {
                System.out.println(e.getMessage());
            }

    }

}
```

Screenshots of Running Application Results:







URL to GitHub Repository:

https://github.com/ailimutan/week-11mysql