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:

```java
package week11Assignment;

import java.util.LinkedList;
import java.util.List;

public class Shoes {

    private String name;
    /*
     * List of shoe types created to be used with this assignment's concepts.
     */
    // @formatter:off
            private static final List<Shoes> Shoes = List.of(
                    new Shoes("Runners"),
                    new Shoes("Trainers"),
                    new Shoes("Sneakers"),
                    new Shoes("Cycling"));
    // @formatter:on

    // Getter and constructor.

    public Shoes(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    // Method to return understandable types of objects.

    @Override
    public String toString() {
        return name + " shoes";
    }

    // Method for list retrieval by other classes.

    public static List<Shoes> getShoes() {
        return new LinkedList<>(Shoes);
    }

    // Compare method for sorting operations.

    public int compare(Shoes that) {
        return this.name.compareTo(that.name);
    }
}
```

```java
package week11Assignment;

import java.util.List;
import java.util.Scanner;

public class SortShoes {
    public Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        new SortShoes().run();
    }
// Run method for selection of which sorting option user wants.

    private void run() {
        System.out.println(
                "Enter option: \n1 for sorting with Lamda"
        + "\nOr any other input for sorting with Method Reference");
        String search = scanner.nextLine();
        List<Shoes> shoes;
        String name;

        System.out.println("Original List: " + Shoes.getShoes()); //Prints what the original list is.

        /*
         * User enters 1 then Lambda sorting is used.
         * if any other entry is entered, Method Reference will
         * be used.
         */
        if (search.equals("1")) {
            shoes = sortWithLambda();
            name = "Lambda Sort: ";
        } else {
            shoes = (List<Shoes>) sortWithMethodReference();
            name = "Method Reference Sort: ";
        }

        System.out.println(name + shoes); // List is printed to console of the sorting results.
    }

    // Method for Method Reference Sorting.

    private Object sortWithMethodReference() {
        List<Shoes> shoes = Shoes.getShoes();

        shoes.sort(Shoes::compare);

        return shoes;
    }

    // Method for Lambda Sorting.

    private List<Shoes> sortWithLambda() {
        List<Shoes> shoes = Shoes.getShoes();

        shoes.sort((p1, p2) -> p1.compare(p2));

        return shoes;
    }

}

```

```java
package week11Assignment;

import java.util.stream.Collectors;

public class StreamShoes {

    public static void main(String[] args) {
        new StreamShoes().run();
    }


     // Run Method for streaming the list sorted and the type amended to each item.

    private void run() {
        String names = Shoes.getShoes()
        //@formatter:off
            .stream()
            .map(c -> c.toString())
            .sorted()
            .collect(Collectors.joining(", \n"));
        //@formatter:on

        System.out.println("Stream:\n" + names); // Stream list printed to console.
    }

}

```

```java
package week11Assignment;

import java.util.NoSuchElementException;
import java.util.Optional;

public class OptionalShoes {
    public static void main(String[] args) {
        new OptionalShoes().run();

    }

    /*
     * Optional Method for handling input values of non-null nature
     * where the handling of a null can break a program if not prepared properly.
     * Instead a value is checked to see if missing.
     */
    private void run() {
        Optional<Shoes> value = Optional.of(new Shoes("Hiking"));
        Shoes shoes = shoesMethod(value);
        System.out.println("I want " + shoes + ".");

        try {
         Optional<Shoes> empty = Optional.empty();
         shoesMethod(empty);

        }
        catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }

    private Shoes shoesMethod(Optional<Shoes> shoesOptional) {

    return shoesOptional.orElseThrow(()
    -> new NoSuchElementException("The shoes do not exist."));

    }
}

```

Screenshots of Running Application Results:

SortShoes:

```
Enter option:
1 for sorting with Lamda
Or any other input for sorting with Method Reference
1
Original List: [Runners shoes, Trainers shoes, Sneakers shoes, Cycling shoes]
Lambda Sort: [Cycling shoes, Runners shoes, Sneakers shoes, Trainers shoes]
```

```
Enter option:
1 for sorting with Lamda
Or any other input for sorting with Method Reference
2
Original List: [Runners shoes, Trainers shoes, Sneakers shoes, Cycling shoes]
Method Reference Sort: [Cycling shoes, Runners shoes, Sneakers shoes, Trainers shoes]
```

StreamShoes:

```
Stream:
Cycling shoes,
Runners shoes,
Sneakers shoes,
Trainers shoes
```

OptionalShoes:

```
I want Hiking shoes.
The shoes do not exist.
```

URL to GitHub Repository:

https://github.com/PierceIsaacson/week-11-MySQL-week5