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:

Step 1.

```
1  package com.promineotech.Fruit;
2
3  import java.util.List;
7
8  public class MyFruit {
9    private SortService sortService = new SortService();
10
11   public static void main(String[] args) {
12     new MyFruit().run();
13
14 }
15
16   private void run() {
17     List<Fruit> fruits = sortService.getFruits(SortType.METHOD_REFERENCE);
18     print(fruits,SortType.METHOD_REFERENCE);
19   }
20
21   private void print(List<Fruit> fruits, SortType type) {
22     switch(type) {
23       case LAMBDA:
24         fruits.forEach(fruit -> System.out.println(fruit.getFruit()));
25         break;
26
27       case METHOD_REFERENCE:
28         fruits.forEach(System.out::println);
29         break;
30
31       case NORMAL_CLASS:
32         for(Fruit fruit : fruits) {
33           System.out.println(fruit.getFruit());
34         }
35         break;
36
37       default:
38         break;
39
40     }
41
42   }
43 }
```

```java
package com.promineotech.Fruit.dao;

import java.util.ArrayList;

public class SortDao {
  List<Fruit> fruits = new ArrayList<>(List.of(new Fruit("Apple"), new Fruit("Banana"),
      new Fruit("Cherries"), new Fruit("Orange"), new Fruit("Mango"),
      new Fruit("Lemon"), new Fruit("Grapes"), new Fruit("Watermelon")));



  public List<Fruit> getFruits() {
    return fruits;
  }

}

```

```java
package com.promineotech.Fruit.model;

public class Fruit {
  private String fruit;

  public Fruit(String name) {
    this.fruit = name;
  }

  @Override
  public String toString() {
    return fruit;
  }

  public String getFruit() {
    return fruit;
  }

  public static int compareFruits(Fruit f1, Fruit f2) {
    return f1.fruit.compareTo(f2.fruit);
  }

}

```

```java
package com.promineotech.Fruit.service;

public enum SortType {
  NORMAL_CLASS,LAMBDA,METHOD_REFERENCE
}

```

```java
package com.promineotech.Fruit.service;

import java.util.Comparator;

public class SortService {
  private SortDao sortDao = new SortDao();

  public List<Fruit> getFruits(SortType type) {
    List<Fruit> fruits = sortDao.getFruits();
    Comparator<Fruit> comp = null;

    switch(type) {
      case LAMBDA:
        comp = (f1, f2) -> Fruit.compareFruits(f1, f2);

        break;

      case METHOD_REFERENCE:
        comp = Fruit::compareFruits;
        break;

      case NORMAL_CLASS:
        comp = new MyFruit();
        break;

      default:
        throw new RuntimeException("Unhandled sort type: " + type);


  }
    fruits.sort(comp);
    return fruits;
}

  static class MyFruit implements Comparator<Fruit> {

    @Override
    public int compare(Fruit f1, Fruit f2) {
      return Fruit.compareFruits(f1, f2);
    }
  }
 }

```

Step 2.

```java
package com.promineotech.Fruit;

import java.util.List;


public class FruitStreaming {
  private RepeatService repeatService = new RepeatService();
  public static void main(String[] args) {
    new FruitStreaming().run();
  }

  private void run() {
    Map<String, List<RepeatingFruit>> names = repeatService.findRepeatingFruit();
    //@formatter:off
    names.entrySet().forEach(entry ->
    System.out.println(entry.getKey() + ": " +
        entry.getValue().stream()
            .map(RepeatingFruit::getFruit)
            .collect(Collectors.toList())));
    //@formatter:on
  }
}

```

```java
package com.promineotech.Fruit.dao;

import java.util.List;


public class RepeatDao {
  List<String> Fruits = List.of("Apple","Banana","Cherries","Orange","Mango",
                                "Lemon","Grapes","Watermelon","Banana","Cherries",
                                "Blackberry","Pinapple","Litchee","Cape Gooseberry",
                                "Passionfruit");

  public List<String> getFruits() {
    return Fruits;
  }
}

```

```java
package com.promineotech.Fruit.service;

import java.util.List;


public class RepeatService {
private RepeatDao repeatDao = new RepeatDao();

  public Map<String, List<RepeatingFruit>> findRepeatingFruit() {
    // @formatter:off
    return repeatDao.getFruits().stream()
          .filter(RepeatingFruit::hasRepeatingCharacter)
          .map(RepeatingFruit::new)
          .sorted()
          .collect(Collectors.groupingBy(RepeatingFruit::getRepeatingChar));
    //@formatter:on
  }
}

```

```java
package com.promineotech.Fruit.model;

import java.util.Objects;

public class RepeatingFruit implements Comparable<RepeatingFruit> {
    private String repeatingChar;
    private String fruit;

    public RepeatingFruit(String fruit) {
      this.fruit = Objects.requireNonNull(fruit, "Name must not be null!");
      this.repeatingChar = findRepeatingChar(fruit);
      Objects.requireNonNull(this.repeatingChar,
          "The fruit " + fruit + " must have one or more repeating characters!");
    }

    public static boolean hasRepeatingCharacter(String name) {
      return findRepeatingChar(name) != null;
    }

    public String getRepeatingChar() {
      return repeatingChar;
    }

    public String getFruit() {
      return fruit;
    }

    private static String findRepeatingChar(String fruit) {
      if(Objects.isNull(fruit)|| fruit.trim().isEmpty() ) {
        throw new RuntimeException("Name must not be null or empty");
      }

      String trimmed = fruit.trim();
      char lastChar = fruit.charAt(0);

      for(int index = 1; index < trimmed.length(); index++) {
       char thisChar = fruit.charAt(index);

       if(lastChar == thisChar) {
         return Character.toString(thisChar);
       }
       lastChar = thisChar;
      }
      return null;
    }

    @Override
    public int compareTo(RepeatingFruit that) {
      int cmp = this.repeatingChar.compareTo(that.repeatingChar);

      if(cmp == 0) {
        cmp = this.fruit.compareTo(that.fruit);
      }
```

Step3.

```java
package com.promineotech.Fruit;

import java.util.NoSuchElementException;


public class Optionals {
  private Scanner scanner = new Scanner(System.in);
  private OptionalServices service = new OptionalServices();
  public static void main(String[] args) {
    new Optionals().run();
  }

  private void run() {
    boolean done = false;

    while(!done) {
      System.out.println("Enter a fruit: ");
      String search = scanner.nextLine();

      if(search.isEmpty()) {
        done = true;
      }
      else {
        try {
        String found = service.find(search);
        System.out.println("I found " + found + "!");
      }
        catch(NoSuchElementException e) {
          System.out.println(e.getMessage());
        }
      }
    }
  }

}
```

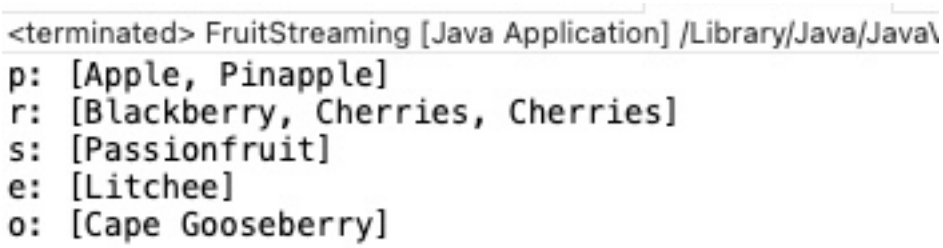
```java
package com.promineotech.Fruit.dao;

import java.util.Optional;

public class OptionalDao {
    public Optional<String> find(String search) {
      if(" ".equals(search)) {
        return Optional.empty();
      }
      return Optional.of(search);
    }
  }


```

```java
package com.promineotech.Fruit.service;

import java.util.NoSuchElementException;

public class OptionalServices {
  private OptionalDao dao = new OptionalDao();
  public String find(String search) {
    return dao.find(search).orElseThrow(() -> new NoSuchElementException(
        "It appears that a name of a fruit is missing."));

  }


}
```

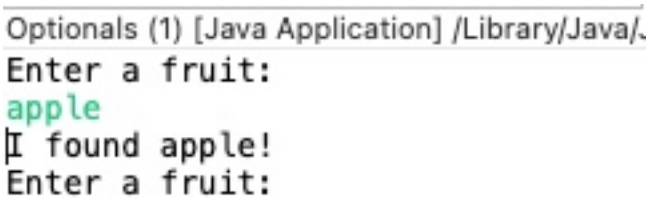
Screenshots of Running Application Results:

Step 1.

Step 2.



Step 3.



URL to GitHub Repository:

https://github.com/AaronL1981/Java-Week-11-MySQL-Assignment-.git