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:

Problem 1

```java
package entity;

import java.util.Arrays;

// 1. Create a class of whatever type you want(Animal, Person, Camera, etc).
// a. Do NOT implement the Comparable interface
public class Cookies {
  //b. Add a name instance variable so that you can tell the objects apart.
  private String name;

  //c. Add getters, setters and/or a constructor as appropriate
  public Cookies(String name) {
    this.setName(name);
  }

  public String getName() {
    return name;
  }

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

  //d. Add a toString method that returns the name and object type (like "Pentax Camera")
  public String toString() {
    return (name + " Cookie");
  }

  // e. Create a static method names 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
  //    are "equal"
  // the compareTo method from the String class accomplishes this. Checks alphabetically
  public static int compare(Cookies c1, Cookies c2) {
    return c1.getName().compareTo(c2.getName());
  }

//f. Create a static list of these objects, adding at least 4 objects the the list.
  public static List<Cookies> cookieList = Arrays.asList(
      new Cookies("Chocolate Chip"),
      new Cookies("Snickerdoodle"),
      new Cookies("Sugar"),
      new Cookies("Peanutbutter"),
      new Cookies("Macademia Nut"));

}
```

```java
package sort;

import java.util.List;

public class CookiesSort {

  // g. In another class, write a method to sort the objects using a Lamnda expression
  //    using the compare method you created earlier.
  public List<Cookies> sortedListLamda(List<Cookies> cookieList){
    cookieList.sort((c1, c2) -> Cookies.compare(c1, c2));
    return cookieList;
  }

  // h. Write a method to sort the objects using a Method Reference to the compare method
  public List<Cookies> sortedListMethodRef(List<Cookies> cookieList){
    cookieList.sort(Cookies::compare);
    return cookieList;
  }

```

```java
package application_main_methods;

import entity.Cookies;

public class ProblemOne {

  public static void main(String[] args) {
    CookiesSort cookiesDao = new CookiesSort();

    // i. Create a main method to call the sort methods
    cookiesDao.sortedListLamda(Cookies.cookieList);

    cookiesDao.sortedListMethodRef(Cookies.cookieList);

    // j. Print the list after sorting (System.out.println).
    System.out.println(Cookies.cookieList);

  }

}

```

Problem 2

```java
package application_main_methods;

import java.util.stream.Collectors;

public class ProblemTwo {

  public static void main(String[] args) {
    //2. Create a new class with a main method. Using the list of objects you created
    //   in the prior problem -

    //a. Create a Stream from the list of objects.
    //b. Turn the Stream of object into Stream of String(use the map method for this).
    //c. Sort the stream in the natural order
    //d. Collect the Stream and return a comma-separated list of names as a single String
    String cookieStream = Cookies.cookieList.stream().map(Cookie -> Cookie.toString())
        .sorted().collect(Collectors.joining(", "));

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

  }

}
```

Problem 3

```java
package application_main_methods;

import java.util.NoSuchElementException;

public class ProblemThree {
  static Scanner input = new Scanner(System.in);

  //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.
  public static Cookies cookieMethod(Optional<Cookies> optionalCookies) {
    return optionalCookies.orElseThrow(() -> new NoSuchElementException("Hmmm..no cookie"));
  }

  // 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()).
  public static void findCookies(Cookies cookie) {
    // Code done with mentor
    Optional<Cookies> optional = Optional.of(Cookies.cookieList.get(0)); // 0 is chocolate chip
    try {
      System.out.println("I found a " + cookieMethod(optional) + "!");
    }catch (NoSuchElementException e) {
      System.out.println(e.getMessage());
    }

//   for user input: Just making sure it would work with input if I wanted it to.
    Optional<Cookies> optionalInput = null;
    if(cookie.getName().isBlank()) {
       optionalInput = Optional.empty();
    }else{
      optionalInput = Optional.of(cookie);
    }
```

```
46
47      System.out.println("*From user input*");
48      try {
49        System.out.println("I found a " + cookieMethod(optionalInput) + "!");
50      }catch (NoSuchElementException e) {
51        System.out.println(e.getMessage());
52      }
53
54  //  for controlled tests. Just in case the code I did with the mentor wasn't adequate for assignment
55      System.out.println("_____");
56      System.out.println("*Cookie Method with object wrapped in optional: *");
57      System.out.println("I found a " + cookieMethod(Optional.of(cookie)) + "!");
58
59      System.out.println("_____");
60      System.out.println("*With empty optional: *");
61      try {
62        cookieMethod(Optional.empty());
63      }catch (NoSuchElementException e) {
64        System.out.println(e.getMessage());
65      }
66    }
67
68    public static void main(String[] args) {
69      System.out.println("Please enter the name of the cookie: ");
70      String name = input.nextLine();
71      Cookies cookie = new Cookies(name);
72      findCookies(cookie);
73    }
74
75
```

Screenshots of Running Application Results:

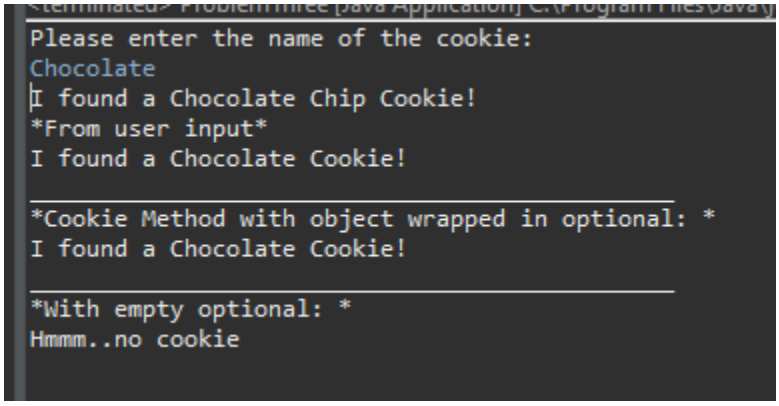
Problem 1 running

```
<terminated> ProblemOne [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe  (May 6, 2022, 10:30:47 AM – 10:30:47 AM)
[Chocolate Chip Cookie, Macademia Nut Cookie, Peanutbutter Cookie, Snickerdoodle Cookie, Sugar Cookie]
```

Problem 2 running

```
<terminated> ProblemTwo [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe  (May 6, 2022, 10:31:08 AM – 10:31:08 AM)
Chocolate Chip Cookie, Macademia Nut Cookie, Peanutbutter Cookie, Snickerdoodle Cookie, Sugar Cookie
```

Problem 3 running



URL to GitHub Repository:

https://github.com/ANGRAYSON/Week-11-Coding-Assingment