Relational Databases with MySQL Week 4 Coding 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 steps 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) {...}
    ```

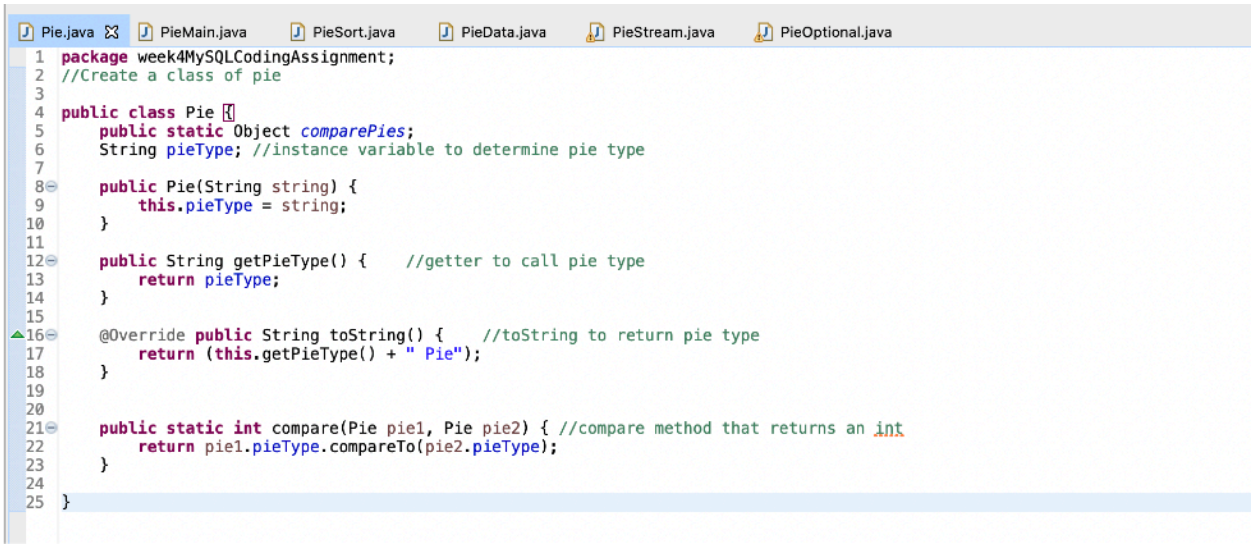
    b. The method should throw a NoSuchElementException with a custom message if the object is not present.

    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:

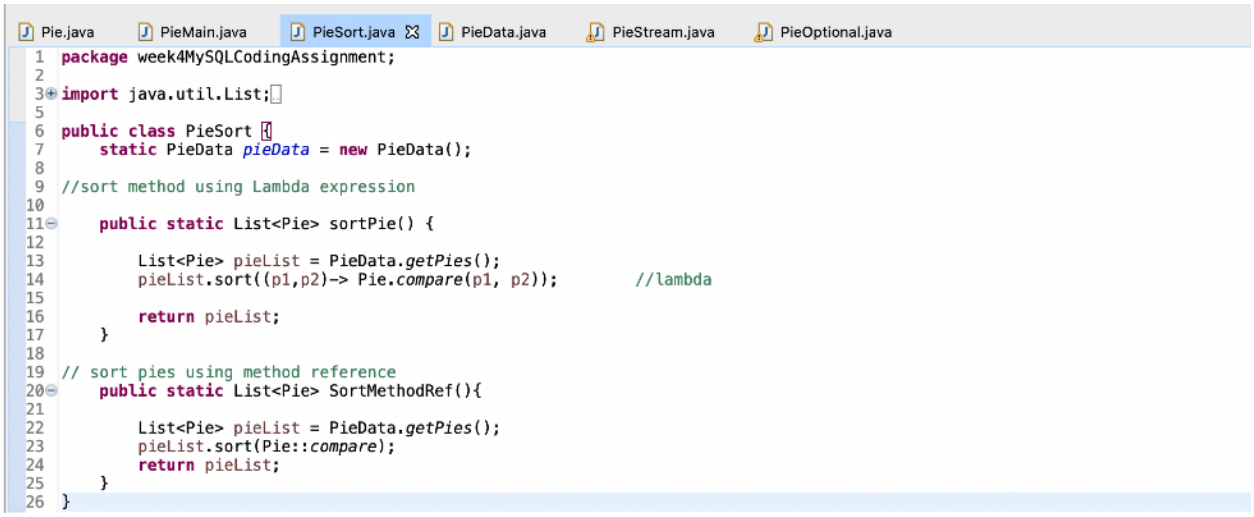


Pie.java | PieMain.java | PieSort.java | PieData.java | PieStream.java | PieOptional.java

```
1  package week4MySQLCodingAssignment;
2  //Create a class of pie
3
4  public class Pie {
5      public static Object comparePies;
6      String pieType; //instance variable to determine pie type
7
8      public Pie(String string) {
9          this.pieType = string;
10     }
11
12     public String getPieType() {    //getter to call pie type
13         return pieType;
14     }
15
16     @Override public String toString() {    //toString to return pie type
17         return (this.getPieType() + " Pie");
18     }
19
20
21     public static int compare(Pie pie1, Pie pie2) { //compare method that returns an int
22         return pie1.pieType.compareTo(pie2.pieType);
23     }
24
25 }
```

Pie.java | PieMain.java | PieSort.java | PieData.java | PieStream.java | PieOptional.java

```
1  package week4MySQLCodingAssignment;
2
3  import java.util.List;
4
5  public class PieMain {
6
7  //main method that calls sorting method
8      public static void main(String[] args) {
9          List<Pie> sortedPieList = PieSort.sortPie();
10         System.out.println(sortedPieList);  //prints list after sorting pies
11     }
12 }
```



Pie.java | PieMain.java | PieSort.java | PieData.java | PieStream.java | PieOptional.java

```
1  package week4MySQLCodingAssignment;
2
3  import java.util.List;
5
6  public class PieSort {
7      static PieData pieData = new PieData();
8
9  //sort method using Lambda expression
10
11     public static List<Pie> sortPie() {
12
13         List<Pie> pieList = PieData.getPies();
14         pieList.sort((p1,p2)-> Pie.compare(p1, p2));        //lambda
15
16         return pieList;
17     }
18
19 // sort pies using method reference
20     public static List<Pie> SortMethodRef(){
21
22         List<Pie> pieList = PieData.getPies();
23         pieList.sort(Pie::compare);
24         return pieList;
25     }
26 }
```

Pie.java | PieMain.java | PieSort.java | PieData.java | PieStream.java | PieOptional.java

```java
package week4MySQLCodingAssignment.dao;

import java.util.ArrayList;

public class PieData {
//list of objects
     public static List<Pie> pies = new ArrayList<Pie>(List.of(new Pie("Apple"),
            new Pie("Blueberry"), new Pie("Cherry"), new Pie("Chocolate Cream"),
            new Pie("Huckleberry"), new Pie("Rhubarb"), new Pie("Banana cream"),
            new Pie("Key Lime"), new Pie("Shoofly"), new Pie("Mississippi Mud"),
            new Pie("French Silk"), new Pie("Minced Meat"), new Pie("Lemon Meringue"),
            new Pie("Banoffee"), new Pie("Blackberry"), new Pie("Coconut Cream"),
            new Pie("Buttermilk"), new Pie("Peach"), new Pie("Pumpkin"),
            new Pie("Raspberry"), new Pie("Tamale"), new Pie("Pecan")));

    public static List<Pie> getPies(){
        return pies;
    }
}
```

Pie.java | PieMain.java | PieSort.java | PieData.java | PieStream.java | PieOptional.java

```java
package week4MySQLCodingAssignment.streaming;

import java.util.List;
import java.util.stream.Collectors;
import week4MySQLCodingAssignment.Pie;
import week4MySQLCodingAssignment.dao.PieData;


public class PieStream {

    public static void main(String[] args) {
        PieData pieOne = new PieData();
        List<Pie> pies = pieOne.getPies();
        String pieString = pies.stream()    //stream of pies
                .map(String::valueOf)       //stream of pies to stream of string
                .sorted()                   //sort stream
                .collect(Collectors.joining(", ")); //collect stream and return comma-separated string
        System.out.println(pieString);      //print string
    }
}
```

Pie.java | PieMain.java | PieSort.java | PieData.java | PieStream.java | PieOptional.java

```java
package week4MySQLCodingAssignment.dao;


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

import week4MySQLCodingAssignment.Pie;

public class PieOptional {

    public static void main(String[] args) {
        methodB();

    }

    public static Pie pieMethod(Optional<Pie> optionalPie) {    //returns unwrapped pie
        return optionalPie.orElseThrow(() -> new NoSuchElementException("Unable to find pie!")); //exception w/ custom message
    }

    public static void methodB() {  //calls pieMethod with object wrapped by an optional
        Optional<Pie> pies = Optional.of(PieData.pies.get(15)); //15 == coconut cream pie
        System.out.println(pieMethod(pies));    //unwrapped from optional

        try {
            pieMethod(pies.empty());        //empty optional
        }
        catch (Exception e) {
            System.out.println(e.getMessage()); //exception message from empty optional
        }
    }
}
```

Console

```
<terminated> PieOptional [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.jdk/Contents/Home/bin/java  (Aug 28, 2021, 9:04:55 PM – 9:04:55 PM)
Coconut Cream Pie
Unable to find pie!
```

URL to GitHub Repository: