

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:

Beer.java:

```

Beer.java × BeerSort.java StreamBeer.java BeerOptional.java
1 package beer;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Beer {
7
8     public static Object beerTypes;
9     private String beerName;
10
11     public Beer() {}
12     public Beer(String name) {
13         this.beerName = name;
14     }
15
16     @Override
17     public String toString() {
18         return (this.getName() + " Beer");
19     }
20     //getters and setters
21     public String getName() {
22         return beerName;
23     }
24     public void setName(String name) {
25         this.beerName = name;
26     }
27
28     //compare method
29
30     //compare method
31     public static int compare(Beer b1, Beer b2) {
32         return b1.beerName.compareTo(b2.beerName);
33     }
34
35     public static List<Beer> beers = new ArrayList<Beer>(List.of
36         (new Beer("Pilsner"), new Beer("Lager"), new Beer("Shandy"),
37         new Beer("Stout"), new Beer("Ale"), new Beer("Porter")));
38
39     public static List<Beer> getBeers() {
40         return beers; //returns list of beer
41     }

```

BeerSort.java:

```

1 package beer;
2
3 import java.util.List;
4
5 public class BeerSort {
6     static Beer beer = new Beer();
7
8
9     public static List<Beer> sorted() {
10         List<Beer> beerType = Beer.getBeers();
11         beerType.sort((b1, b2) -> Beer.compare(b1, b1));
12         return beerType;
13     }
14
15     public static List<Beer> MethodRefSort() {
16         List<Beer> beerType = Beer.getBeers();
17         beerType.sort(Beer::compare);
18         return beerType;
19     }
20
21     public static void main(String[] args) {
22         List<Beer> sortedList = BeerSort.sorted();
23         System.out.println(sortedList);
24
25         List<Beer> mSortedList = BeerSort.MethodRefSort();
26         System.out.println(mSortedList);
27
28         System.out.println(mSortedList);
29
30     }
31 }

```

StreamBeer.java:

```

1 package beer;
2
3 import java.util.List;
4 import java.util.stream.Collectors;
5
6 public class StreamBeer {
7
8     public static void main(String[] args) {
9
10         List<Beer> beerType = Beer.getBeers();
11         String beerStream = beerType.stream()
12             .map(Beer::toString)
13             .sorted()
14             .collect(Collectors.joining(" , "));
15         System.out.println(beerStream);
16     }
17 }
18
19

```

BeerOptional:

```

2
3 import java.util.List;
4 import java.util.NoSuchElementException;
5 import java.util.Optional;
6
7 public class BeerOptional {
8
9     public static void main(String[] args) {
10
11     }
12
13     public static List<Beer> beerMethodA(Optional<List<Beer>> beer) {
14         return beer.orElseThrow(() -> new NoSuchElementException("Value is not present!"));
15     }
16
17     public static void beerMethodB() {
18         Optional<List<Beer>> beer = Optional.of(Beer.getBeers());
19         System.out.println(beerMethodA(beer));
20         //unwrapped
21
22         try {
23             beerMethodA(Optional.empty());
24         } catch (NoSuchElementException e) {
25             System.out.println(e.getMessage());
26         }
27     }
28 }
29

```

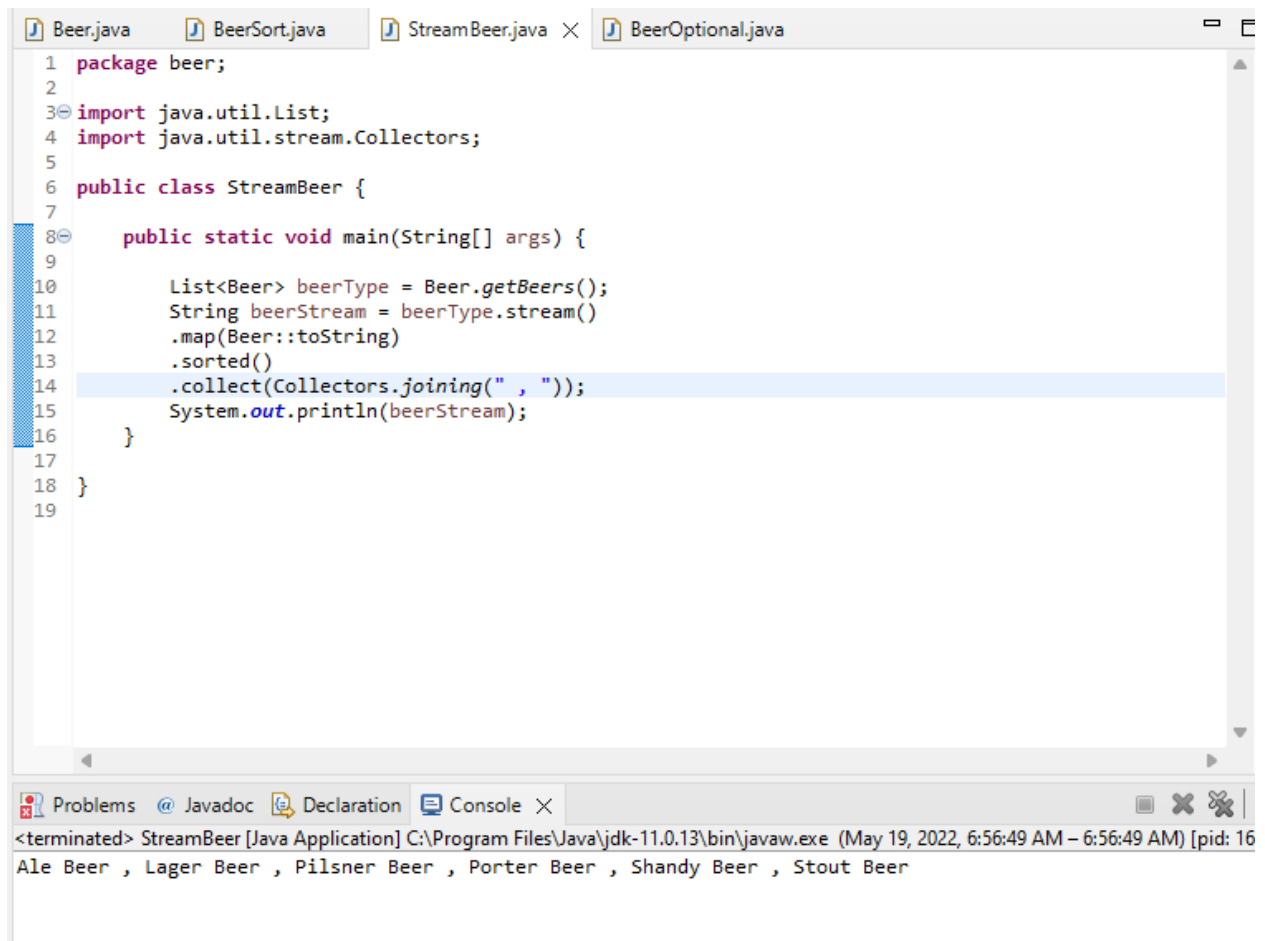
Screenshots of Running Application Results:

The screenshot shows an IDE with four tabs: Beer.java, BeerSort.java (active), StreamBeer.java, and BeerOptional.java. The BeerSort.java file contains the following code:

```
1 package beer;
2
3 import java.util.List;
4
5 public class BeerSort {
6     static Beer beer = new Beer();
7
8
9     public static List<Beer> sorted() {
10         List<Beer> beerType = Beer.getBeers();
11         beerType.sort((b1, b2) -> Beer.compare(b1, b1));
12         return beerType;
13     }
14
15     public static List<Beer> MethodRefSort() {
16         List<Beer> beerType = Beer.getBeers();
17         beerType.sort(Beer::compare);
18         return beerType;
19     }
20
21     public static void main(String[] args) {
22         List<Beer> sortedList = BeerSort.sorted();
23         System.out.println(sortedList);
24
25         List<Beer> mSortedList = BeerSort.MethodRefSort();
26         System.out.println(mSortedList);
27
28     }
```

The Outline view on the right shows a package 'beer' containing a class 'BeerSort' with methods 'sorted', 'MethodRefSort', and 'main'. The Console view at the bottom shows the output of the program:

```
<terminated> BeerSort [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (May 19, 2022, 6:55:42 AM - 6:55:42 AM) [pid: 13644]
[Pilsner Beer, Lager Beer, Shandy Beer, Stout Beer, Ale Beer, Porter Beer]
[Ale Beer, Lager Beer, Pilsner Beer, Porter Beer, Shandy Beer, Stout Beer]
```



The screenshot shows an IDE with four tabs: Beer.java, BeerSort.java, StreamBeer.java, and BeerOptional.java. The StreamBeer.java tab is active, displaying the following code:

```
1 package beer;
2
3 import java.util.List;
4 import java.util.stream.Collectors;
5
6 public class StreamBeer {
7
8     public static void main(String[] args) {
9
10         List<Beer> beerType = Beer.getBeers();
11         String beerStream = beerType.stream()
12             .map(Beer::toString)
13             .sorted()
14             .collect(Collectors.joining(" , "));
15         System.out.println(beerStream);
16     }
17 }
18
19
```

The console output at the bottom shows the execution of the StreamBeer application, resulting in the following text:

```
<terminated> StreamBeer [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (May 19, 2022, 6:56:49 AM - 6:56:49 AM) [pid: 16]
Ale Beer , Lager Beer , Pilsner Beer , Porter Beer , Shandy Beer , Stout Beer
```

URL to GitHub Repository:

<https://github.com/harvisd/week11assignment>