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:

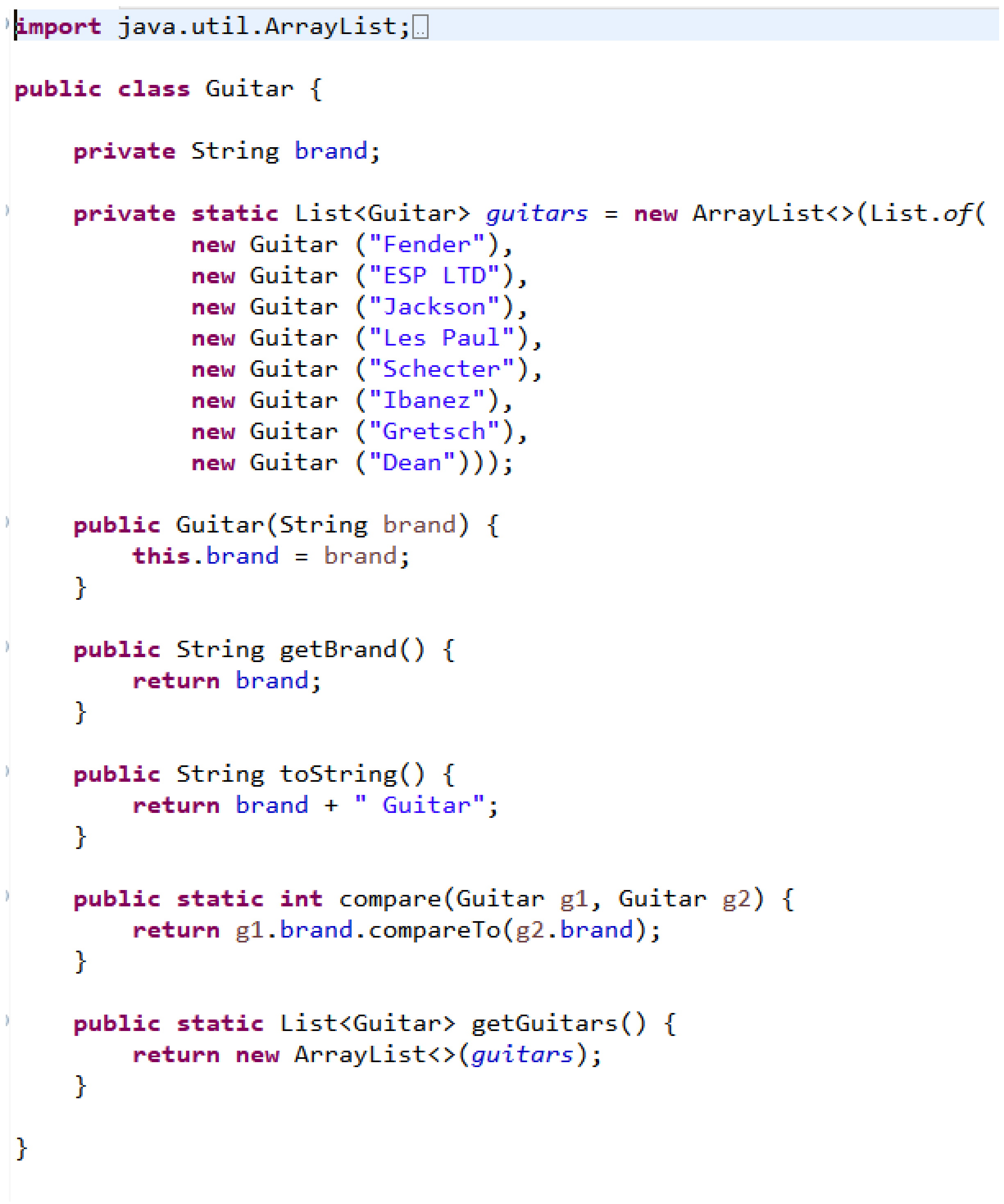
1. Create a class of whatever type you want (Animal, Person, Camera, Cheese, etc.).
    a. Do not implement the Comparable interface.
    b. Add a name instance variable so that you can tell the objects apart.
    c. Add getters, setters and/or a constructor as appropriate.
    d. Add a toString method 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:

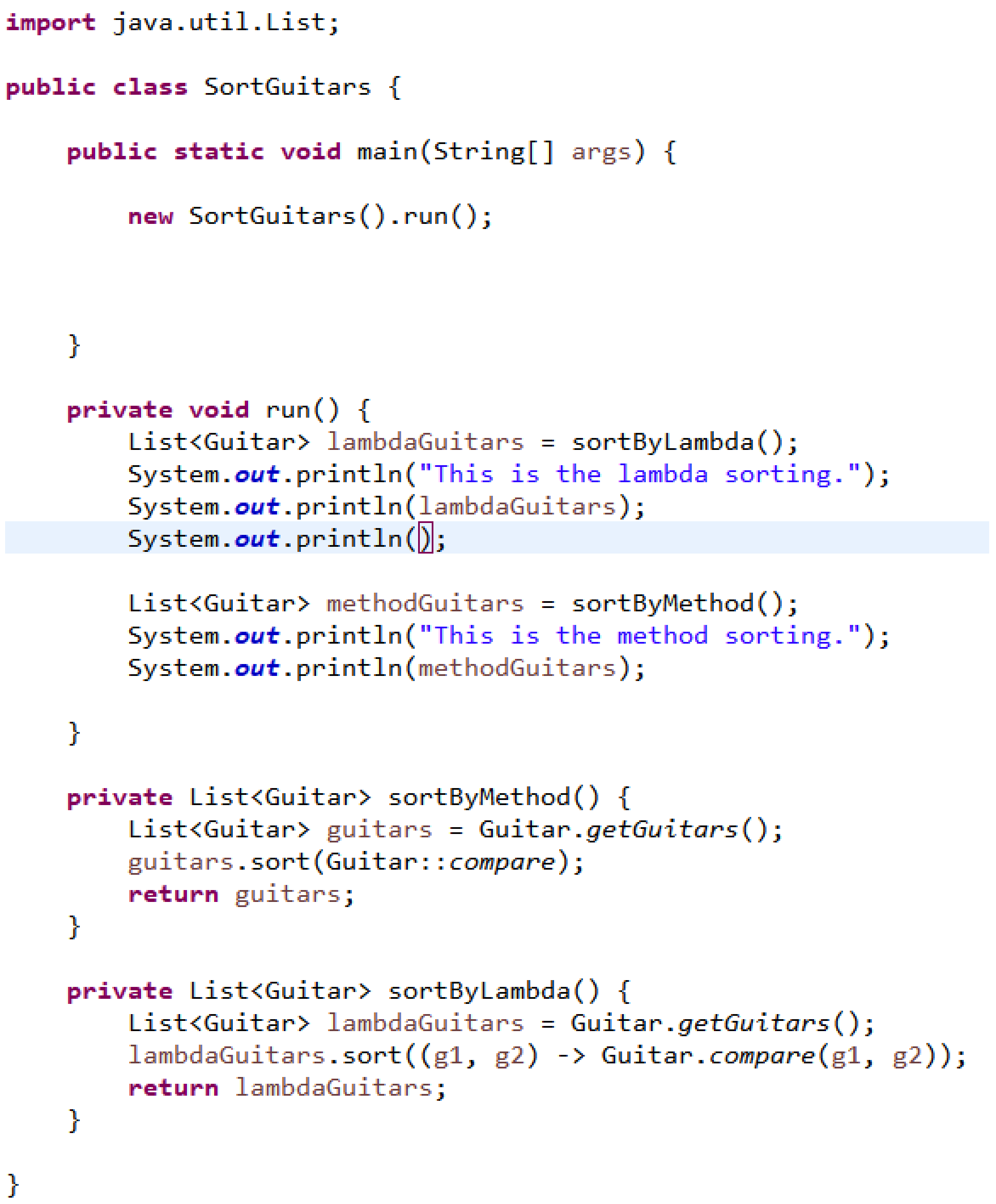
       ```
       public Cheese cheesyMethod(Optional<Cheese> optionalCheese) {...}
       ```

    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.

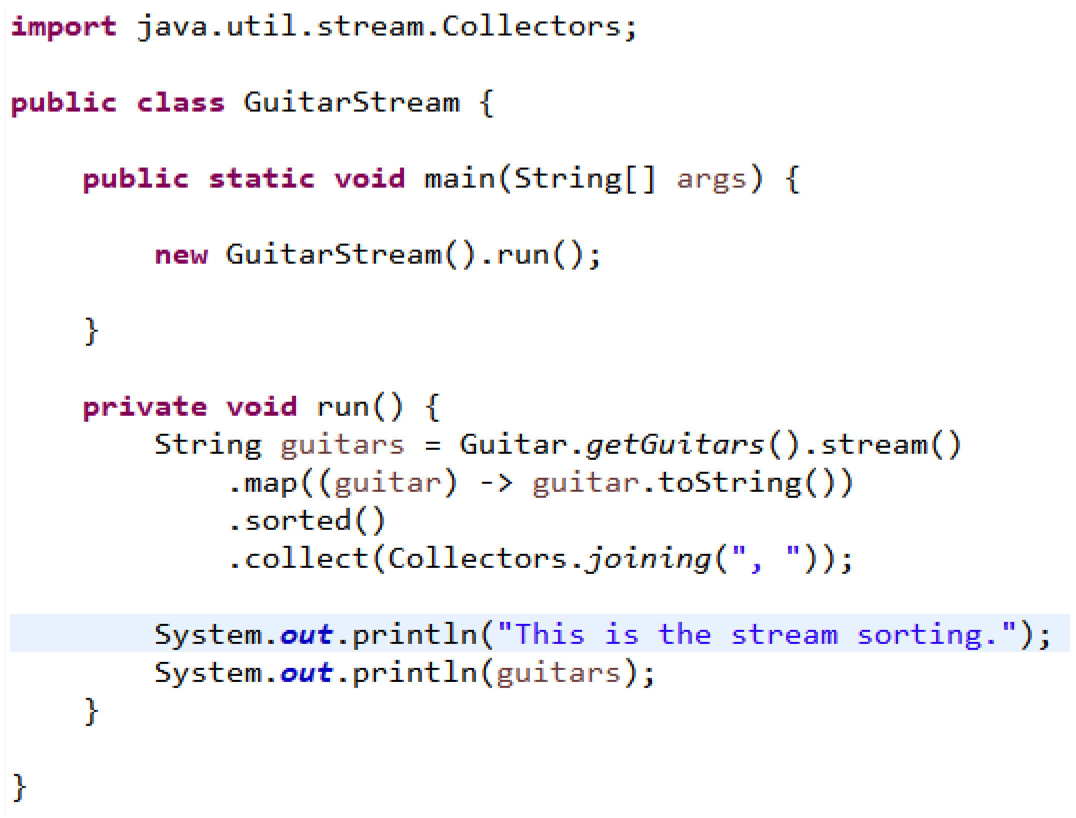
Screenshots of Code:



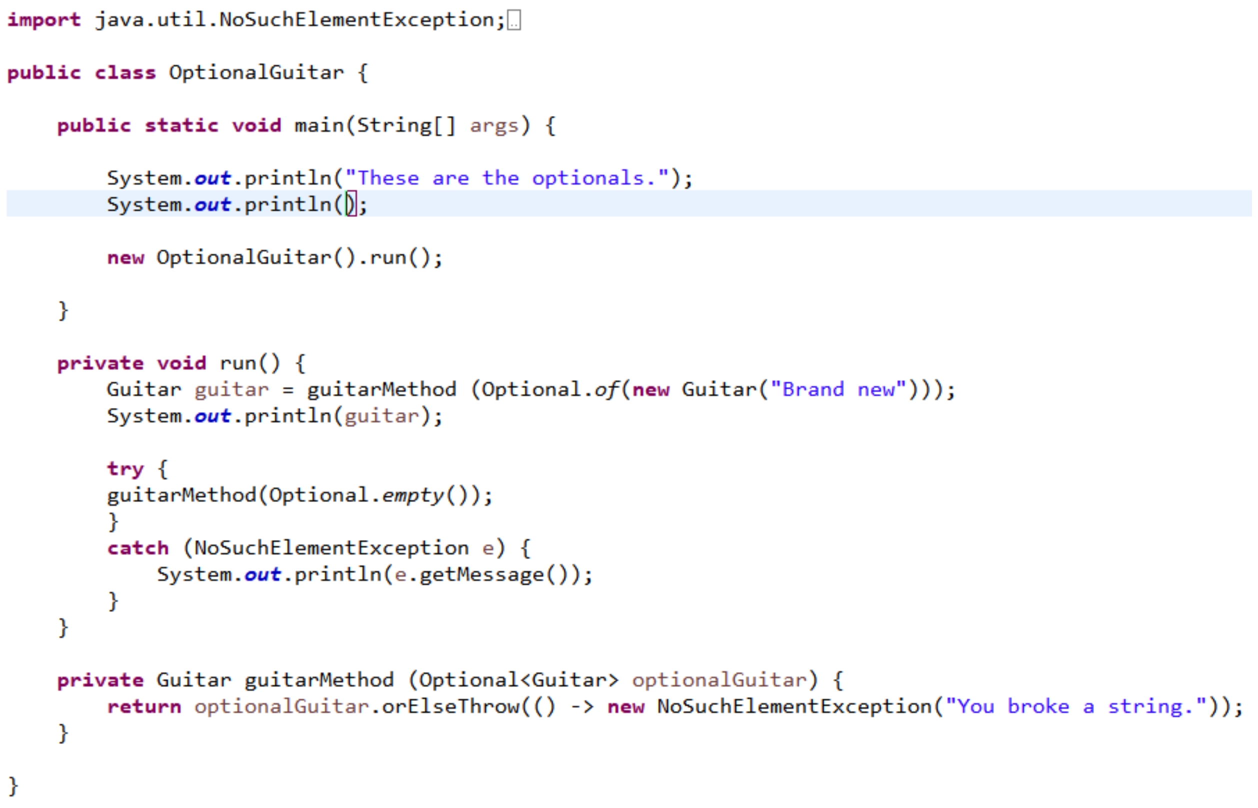
```java
import java.util.ArrayList;

public class Guitar {

    private String brand;

    private static List<Guitar> guitars = new ArrayList<>(List.of(
            new Guitar ("Fender"),
            new Guitar ("ESP LTD"),
            new Guitar ("Jackson"),
            new Guitar ("Les Paul"),
            new Guitar ("Schecter"),
            new Guitar ("Ibanez"),
            new Guitar ("Gretsch"),
            new Guitar ("Dean")));

    public Guitar(String brand) {
        this.brand = brand;
    }

    public String getBrand() {
        return brand;
    }

    public String toString() {
        return brand + " Guitar";
    }

    public static int compare(Guitar g1, Guitar g2) {
        return g1.brand.compareTo(g2.brand);
    }

    public static List<Guitar> getGuitars() {
        return new ArrayList<>(guitars);
    }

}
```



```java
import java.util.List;

public class SortGuitars {

    public static void main(String[] args) {

        new SortGuitars().run();



    }

    private void run() {
        List<Guitar> lambdaGuitars = sortByLambda();
        System.out.println("This is the lambda sorting.");
        System.out.println(lambdaGuitars);
        System.out.println();

        List<Guitar> methodGuitars = sortByMethod();
        System.out.println("This is the method sorting.");
        System.out.println(methodGuitars);

    }

    private List<Guitar> sortByMethod() {
        List<Guitar> guitars = Guitar.getGuitars();
        guitars.sort(Guitar::compare);
        return guitars;
    }

    private List<Guitar> sortByLambda() {
        List<Guitar> lambdaGuitars = Guitar.getGuitars();
        lambdaGuitars.sort((g1, g2) -> Guitar.compare(g1, g2));
        return lambdaGuitars;
    }

}
```

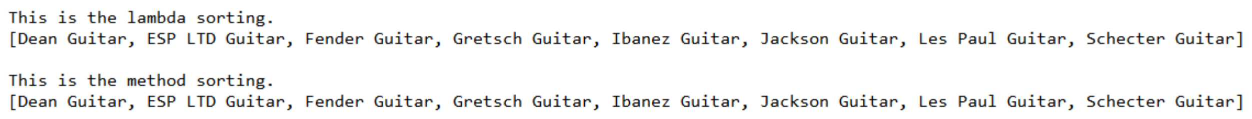


```java
import java.util.stream.Collectors;

public class GuitarStream {

    public static void main(String[] args) {

        new GuitarStream().run();

    }

    private void run() {
        String guitars = Guitar.getGuitars().stream()
            .map((guitar) -> guitar.toString())
            .sorted()
            .collect(Collectors.joining(", "));

        System.out.println("This is the stream sorting.");
        System.out.println(guitars);
    }

}
```

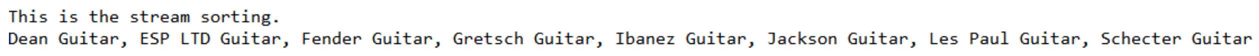


```java
import java.util.NoSuchElementException;

public class OptionalGuitar {

    public static void main(String[] args) {

        System.out.println("These are the optionals.");
        System.out.println();

        new OptionalGuitar().run();

    }

    private void run() {
        Guitar guitar = guitarMethod (Optional.of(new Guitar("Brand new")));
        System.out.println(guitar);

        try {
        guitarMethod(Optional.empty());
        }
        catch (NoSuchElementException e) {
            System.out.println(e.getMessage());
        }
    }

    private Guitar guitarMethod (Optional<Guitar> optionalGuitar) {
        return optionalGuitar.orElseThrow(() -> new NoSuchElementException("You broke a string."));
    }

}
```

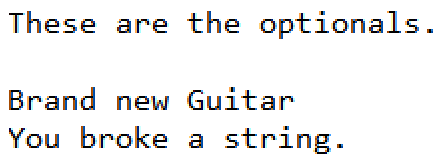
Screenshots of Running Application Results:



```
This is the lambda sorting.
[Dean Guitar, ESP LTD Guitar, Fender Guitar, Gretsch Guitar, Ibanez Guitar, Jackson Guitar, Les Paul Guitar, Schecter Guitar]

This is the method sorting.
[Dean Guitar, ESP LTD Guitar, Fender Guitar, Gretsch Guitar, Ibanez Guitar, Jackson Guitar, Les Paul Guitar, Schecter Guitar]
```



```
This is the stream sorting.
Dean Guitar, ESP LTD Guitar, Fender Guitar, Gretsch Guitar, Ibanez Guitar, Jackson Guitar, Les Paul Guitar, Schecter Guitar
```



```
These are the optionals.

Brand new Guitar
You broke a string.
```

URL to GitHub Repository: https://github.com/Faudeth/Week-11