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.

URL to GitHub Repository:

https://github.com/ladymanj/Week11CodingAssignment.git

Screenshots of Code:

DreamWorksFilm.java:

```java
package dreamworks_films;

import java.util.LinkedList;
import java.util.List;

public class DreamWorksFilm {
    private String name;

    private static final List<DreamWorksFilm> DREAMWORKS_FILMS = List.of(
            new DreamWorksFilm("Shrek"),
            new DreamWorksFilm("Kung Fu Panda"),
            new DreamWorksFilm("How to Train Your Dragon"),
            new DreamWorksFilm("Antz"),
            new DreamWorksFilm("Trolls"),
            new DreamWorksFilm("Megamind"),
            new DreamWorksFilm("Madagascar"),
            new DreamWorksFilm("Bee Movie")
            );


    public DreamWorksFilm(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return name + " - a DreamWorks Film";
    }

    public static int compare(DreamWorksFilm film1, DreamWorksFilm film2) {
        return film1.name.compareTo(film2.name);
    }

    public static List<DreamWorksFilm> getDreamWorksFilms() {
        return new LinkedList<>(DREAMWORKS_FILMS);
    }

    public static void printFilms(List<DreamWorksFilm> films) {
        for(DreamWorksFilm film : films) {
            System.out.println("\t" + film);
        }
        System.out.println();
    }


}

```

DreamWorksFilmSorter.java:

```java
package dreamworks_films;

import java.util.List;

public class DreamWorksFilmSorter {

    public static void main(String[] args) {
        List<DreamWorksFilm> unsortedFilms = DreamWorksFilm.getDreamWorksFilms();
        List<DreamWorksFilm> lambdaFilms;
        List<DreamWorksFilm> methodReferenceFilms;

        System.out.println("Original:");
        DreamWorksFilm.printFilms(unsortedFilms);

        lambdaFilms = sortWithLambda();
        System.out.println("Sort using lambda expression:");
        DreamWorksFilm.printFilms(lambdaFilms);

        methodReferenceFilms = sortWithMethodReference();
        System.out.println("Sort using method reference:");
        DreamWorksFilm.printFilms(methodReferenceFilms);
    }

    private static List<DreamWorksFilm> sortWithLambda() {
        List<DreamWorksFilm> films = DreamWorksFilm.getDreamWorksFilms();

        films.sort((p1, p2) -> DreamWorksFilm.compare(p1, p2));
        return films;
    }

    private static List<DreamWorksFilm> sortWithMethodReference() {
        List<DreamWorksFilm> films = DreamWorksFilm.getDreamWorksFilms();

        films.sort(DreamWorksFilm::compare);
        return films;
    }
}
```

DreamWorksFilmStreamer.java:

```java
package dreamworks_films;

import java.util.stream.Collectors;

public class DreamWorksFilmStreamer {

    public static void main(String[] args) {
        String names = DreamWorksFilm.getDreamWorksFilms()
                .stream()
                .map(p -> p.toString())
                .sorted()
                .collect(Collectors.joining(", "));

        System.out.println(names);
    }

}
```

DreamWorksFilmOptional.java:

```java
package dreamworks_films;

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

public class DreamWorksFilmOptional {

    public static void main(String[] args) {
        DreamWorksFilm film = filmMethod(Optional.of(new DreamWorksFilm("The Bad Guys")));
        System.out.println("The newest and hottest film is: " + film);

        try {
            Optional<DreamWorksFilm> empty = Optional.empty();
            filmMethod(empty);
        }
        catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }

    private static DreamWorksFilm filmMethod(Optional<DreamWorksFilm> filmOptional) {
        return filmOptional.orElseThrow(
                () -> new NoSuchElementException("The film does not exist."));
    }
}
```

Screenshots of Running Application Results:

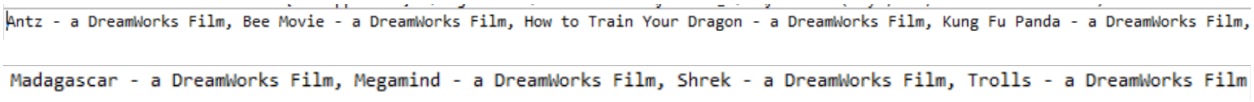
DreamWorksFilmSorter.java running application results:

```
Original:
        Shrek - a DreamWorks Film
        Kung Fu Panda - a DreamWorks Film
        How to Train Your Dragon - a DreamWorks Film
        Antz - a DreamWorks Film
        Trolls - a DreamWorks Film
        Megamind - a DreamWorks Film
        Madagascar - a DreamWorks Film
        Bee Movie - a DreamWorks Film

Sort using lambda expression:
        Antz - a DreamWorks Film
        Bee Movie - a DreamWorks Film
        How to Train Your Dragon - a DreamWorks Film
        Kung Fu Panda - a DreamWorks Film
        Madagascar - a DreamWorks Film
        Megamind - a DreamWorks Film
        Shrek - a DreamWorks Film
        Trolls - a DreamWorks Film

Sort using method reference:
        Antz - a DreamWorks Film
        Bee Movie - a DreamWorks Film
        How to Train Your Dragon - a DreamWorks Film
        Kung Fu Panda - a DreamWorks Film
        Madagascar - a DreamWorks Film
        Megamind - a DreamWorks Film
        Shrek - a DreamWorks Film
        Trolls - a DreamWorks Film
```

DreamWorksFilmStreamer.java running application results:



DreamWorksFilmOptional.java running application results: