R·I·T Information Sciences and Technologies

Name: _____

ISTE-120 Lab 12: Interfaces

This lab uses a series of classes that compare the cost of travel by plane, train and auto from Rochester to some other destination. An interface is required and input is provided through command line arguments.

Exercise 1 – The cost of car travel (3 points)

Step 1: Creating the interface - Create an interface, `TravelCost`, that has the following characteristics:

- A constant `double AGENT_FEE` value in US dollars ($'s) of `10.00`
- An abstract method `getDuration` that returns a double trip length in hrs
- An abstract method `getTotalCost` that returns a double trip cost in $'s
- An abstract method `getLodgingCost` that returns a double lodging cost in $'s
- An abstract method `getDestination` that returns a String destination
- An abstract method `toString` that returns a string formatted by `String.format()`

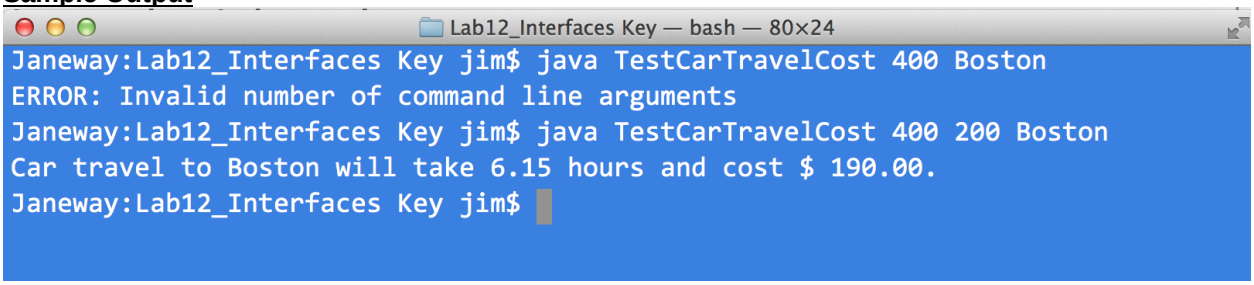
Step 2: Create a class, `CarTravelCost`, with the characteristics:

- Attributes are trip distance `int numMiles`, nightly lodging cost `double hotelCost` and trip destination `String destination`
- A normal accessor for `destination`
- A constructor that accepts and updates all the attributes
- `getDuration` returns: `numHours=(double)numMiles/65`, (uses car speed of 65 mph)
- The `getLodgingCost()` returns: `numDays*hotelCost. numDays`, which is the number of days on the road, is calculated as `getDuration()/8.0` (8 hours of driving per day). There is no need to include fractional days as a whole day (why?); the `Math.floor()` method will help.
- `getTotalCost()` returns: `(numMiles * 0.45) + getLodgingCost() + AGENT_FEE.` This represents a mileage allowance of 45 cents per mile plus lodging.
- A `toString` method returns: `destination, getDuration()` and `getTotalCost()` formatted by `String.format()` (see example below).

Step 3: Create a test class , `TestCarTravelCost`, that takes in three command line arguments: number of miles, hotel cost per night and destination. Issue an error message if there are not three arguments: `ERROR: Invalid number of command line arguments.` All data conversions should be done in the test class (e.g. `String` to `double`). If there is a valid number of arguments then:

- Instantiate an object of the `CarTravelCost` class using the three arguments
- Print out the `toString`

Sample Output



Signature: _____ **Date:** _____

Exercise 2 – Add train travel (2 points)

Step 1: Create a class, `TrainTravelCost`, with the following characteristics:

- Attributes of `double duration`, `double trainFare`, and `String destination`.
- A constructor that accepts and updates all the attributes
- Normal accessors for `duration` and `destination`
- The `getLodgingCost` method returns: `0.0` (passengers can sleep on the train)
- The `getTotalCost` method returns: `travelFare + AGENT_FEE`
- A `toString` method returns: "Train travel to *description* will take *duration* and cost *totalCost"* formatted by `String.format()` … see example below.

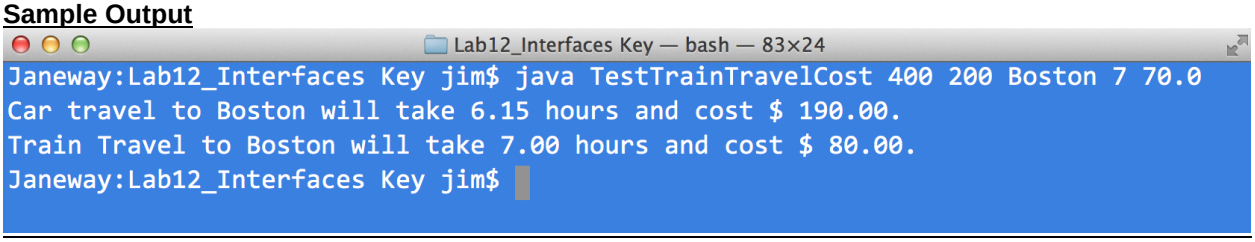
Step 2: Modify the test class `TestCarTravelCost`
Add two more command line arguments, train duration and train fare, for a total of 5 arguments. Put out an error message if there are not five arguments (`ERROR: Invalid number of command line arguments`)

All data conversions should be done in the test class (e.g. `String` to `double`).

If there is a valid number of arguments then:

- Instantiate an object of the `CarTravelCost` class using the arguments (as before)
- Add an instantiation of the class `TrainTravelCost` using the necessary arguments
- Print out the `toString` from the `CarTravelCost` object (as before)
- Add a `toString` method for the `TrainTravelCost` object and print it out; as before, formatted by `String.format()`

Sample Output



Signature: _____ **Date:** _____

Exercise 3 – Add air travel (5 points)

If not completed during the lab period, bring the completed work by next week.

Introduction: Assume you finally made a lot of money and grew tired of taking the train and car everywhere. You begin to wonder how much it will cost to fly to Boston as a result. Calculate the cost of flying and compare it with the costs of traveling by car and train before boarding your jet without a care in the world.

Step 1: Create a class, `AirTravelCost`, that has the following characteristics:

- Attributes of departure/arrival date (`String` in the form `YYYYMMDD`), departure/arrival time (`String` in the form `HHMM`), `destination`, `travelFare`, and `hotelCost` (cost per night).
- A constructor that accepts and updates all the attributes
- A normal accessor for destination
- Implement a getDuration() method in which the `Duration` of the flight can be calculated. This time, instead of the `GregorianCalendar` class that was used in previous labs, use the `LocalDateTime` class from the java.time.* package together with the `DateTimeFormatter` class from the java.time.format.* package. Note: These classes are used to parse the current dates stored in strings into a specific format, from which the timespan between two can be calculated.
    - Create two `String` objects, one for arrival and one for departure. Each will store the flight date and time together. To store the date and time together in a string, concatenate two strings with a space between them
    - Create two `LocalDateTime` objects formatted with `DateTimeFormatter` from newly created string objects, which are stored in the format: **`"yyyyMMdd HHmm"`**
    - An example of storing a string as a LocalDateTime with a format of "yyyy-MM-dd HH:mm:ss" is:

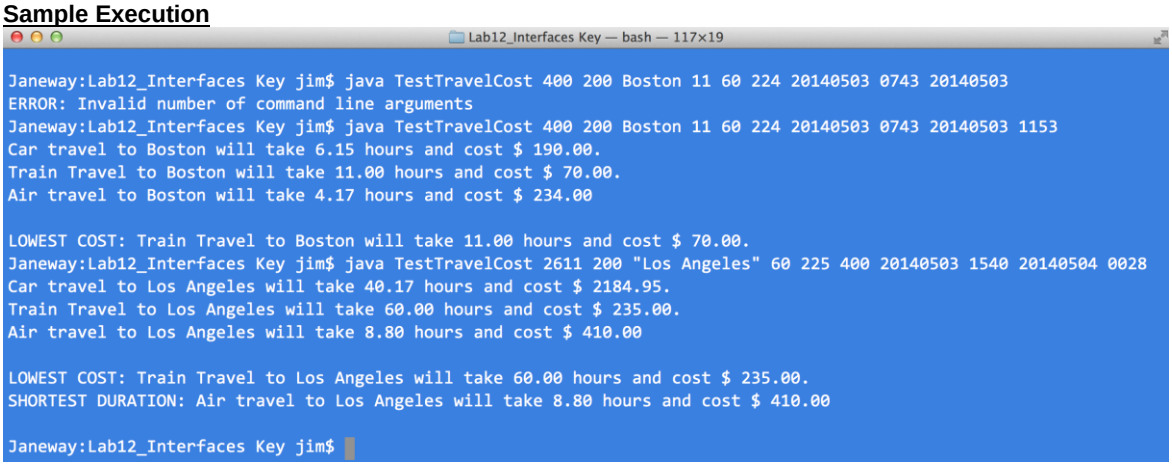
```
LocalDateTime localDateTime = LocalDateTime.parse(dateToFormat,
    DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
```

    - Subtract the two values (for the departure and arrival date/time) to yield duration in milliseconds. To achieve that use the `between` method of the `Duration` object which calculates the timespan of two dates. Convert the result to milliseconds with the `toMillis()` method. Note: time in milliseconds in stored in a long number data type
    - Divide the result by `60,000` to get minutes and divide again by `60` to get hours, etc.
    - Return the result (duration of flight in hours)
- In another method calculate the lodging (hotel) cost by dividing the `duration` by `24` hours and use the number of whole days only (no partial days). Multiply the number of days by the hotel cost per night.
- Create a method that returns the, total cost which can be calculated as `travelFare + getLodgingCost() + AGENT_FEE`
- A `toString` method returns a string formatted by `String.format()` who's output looks like "Air travel to *destination* will take *duration* and cost *totalCost*"

Step 2: Modify the test class `TestTravelCost` to add five more command line arguments, `airFare`, `departureDate` (`String` formatted as `YYYYMMDD`), `departureTime` (`String` formatted as `HHMM` – - in 24 hour time), `arrivalDate`, `arrivalTime` for a total of 10 arguments. All data conversions should be done in the test class (e.g. `String` to `double`). Put out an error message if there are not ten arguments: `ERROR: Invalid number of command line arguments.` If there is a valid number of arguments then:

- Instantiate an object of the `CarTravelCost` class using the arguments (as before)
- Instantiate an object of the `TrainTravelCost` using the necessary arguments (as before)
- Add an instantiation of the class `AirTravelCost` using the necessary arguments
- Add each of these objects to an `ArrayList`. Think about the datatype (i.e. class used to define the `ArrayList` objects).
- Print out the return of the `toString` method from the `CarTravelCost` object (as before)
- Print out the return of the `toString` method from the `TrainTravelCost` object (as before)
- Add a print of the return of the `toString` method from the `AirTravelCost` object
- Search the `ArrayList` to see which travel method is cheapest and which is of shortest duration and print out those results.

Sample Execution



Signature: _____ Date: _____