Scenario

You are working for a software company that has been contracted to develop a GUI-based scheduling desktop application. The contract is with a global consulting organization that conducts business in multiple languages and has main offices in Phoenix, Arizona; White Plains, New York; Montreal, Canada; and London, England. The consulting organization has provided a MySQL database that the application must pull data from. The database is used for other systems, so its structure cannot be modified.

The organization outlined specific business requirements that must be met as part of the application. From these requirements, a system analyst at your company created solution statements for you to implement in developing the application. These statements are listed in the requirements section.

Your company acquires Country and First-Level-Division data from a third party that is updated once per year. These tables are prepopulated with read-only data. Please use the attachment "Locale Codes for Region and Language" to review division data. Your company also supplies a list of contacts, which are pre populated in the Contacts table; however, administrative functions such as adding users are beyond the scope of the application and done by your company's IT support staff. Your application should be organized logically using one or more design patterns and generously commented using Javadoc so your code can be read and maintained by other programmers.

Requirement

A. Create a GUI-based application for the company in the scenario. Regarding your file submission—the use of non-Java API libraries are not allowed with the exception of JavaFX SDK and MySQL JDBC Driver. If you are using the NetBeans IDE, the custom library for your JavaFX .jar files in your IDE must be named JavaFX.

Note: If you are using IntelliJ IDEA, the folder where the JavaFX SDK resides will be used as the library name as shown in the "JavaFX SDK with IntelliJ IDEA webinar.

1. Create a log-in form with the following capabilities:
   - accepts username and password and provides an appropriate error message
   - determines the user's location (i.e., ZoneId) and displays it in a label on the log-in form
   - displays the log-in form in English or French based on the user's computer language setting to translate all the text, labels, buttons, and errors on the form
   - automatically translates error control messages into English or French based on the user's computer language setting

Note: Some operating systems require a reboot when changing the language settings.

2. Write code that provides the following customer record functionalities:
   - Customer records and appointments can be added, updated, and deleted.

- When deleting a customer record, all of the customer's appointments must be deleted first, due to foreign key constraints.

• When adding and updating a customer, text fields are used to collect the following data: customer name, address, postal code, and phone number.

- Customer IDs are auto-generated, and first-level division (i.e., states, provinces) and country data are collected using separate combo boxes.

Note: The address text field should not include first-level division and country data. Please use the following examples to format addresses:

- *U.S. address: 123 ABC Street, White Plains*
- *Canadian address: 123 ABC Street, Newmarket*
- *UK address: 123 ABC Street, Greenwich, London*

- When updating a customer, the customer data auto populates in the form.

• Country and first-level division data is pre-populated in separate combo boxes or lists in the user interface for the user to choose. The first-level list should be filtered by the user's selection of a country (e.g., when choosing U.S., filter so it only shows states).
• All of the original customer information is displayed on the update form.
- Customer_ID must be disabled.
• All of the fields can be updated except for Customer_ID.
• Customer data is displayed using a TableView, including first-level division data. A list of all the customers and their information may be viewed in a TableView, and updates of the data can be performed in text fields on the form.
• When a customer record is deleted, a custom message should display in the user interface.

3. Add scheduling functionalities to the GUI-based application by doing the following:

a. Write code that enables the user to add, update, and delete appointments. The code should also include the following functionalities:

• A contact name is assigned to an appointment using a drop-down menu or combo box.
• A custom message is displayed in the user interface with the Appointment_ID and type of appointment canceled.
• The Appointment_ID is auto-generated and disabled throughout the application.
• When adding and updating an appointment, record the following data: Appointment_ID, title, description, location, contact, type, start date and time, end date and time, Customer_ID, and User_ID.
• All of the original appointment information is displayed on the update form in the local time zone.
• All of the appointment fields can be updated except Appointment_ID, which must be disabled.

b. Write code that enables the user to view appointment schedules by month and week using a TableView and allows the user to choose between these two options using tabs

or radio buttons for filtering appointments. Please include each of the following requirements as columns:

- Appointment_ID
- Title
- Description
- Location
- Contact
- Type
- Start Date and Time
- End Date and Time
- Customer_ID
- User_ID

c. Write code that enables the user to adjust appointment times. While the appointment times should be stored in Coordinated Universal Time (UTC), they should be automatically and consistently updated according to the local time zone set on the user's computer wherever appointments are displayed in the application.

Note: There are up to three time zones in effect. Coordinated Universal Time (UTC) is used for storing the time in the database, the user's local time is used for display purposes, and Eastern Standard Time (EST) is used for the company's office hours. Local time will be checked against EST business hours before they are stored in the database as UTC.

d. Write code to implement input validation and logical error checks to prevent each of the following changes when adding or updating information; display a custom message specific for each error check in the user interface:

- scheduling an appointment outside of business hours defined as 8:00 a.m. to 10:00 p.m. EST, including weekends
- scheduling overlapping appointments for customers
- entering an incorrect username and password

e. Write code to provide an alert when there is an appointment within 15 minutes of the user's log-in. A custom message should be displayed in the user interface and include the appointment ID, date, and time. If the user does not have any appointments within 15 minutes of logging in, display a custom message in the user interface indicating there are no upcoming appointments.

*Note: Since evaluation may be testing your application outside of business hours, your alerts must be robust enough to trigger an appointment within 15 minutes of the **local time set on the user's computer,** which may or may not be EST.*

f. Write code that generates accurate information in *each* of the following reports and will display the reports in the user interface:

Note: You do not need to save and print the reports to a file or provide a screenshot.

- the total number of customer appointments by type and month
- a schedule for each contact in your organization that includes appointment ID, title, type and description, start date and time, end date and time, and customer ID
- an additional report of your choice that is different from the two other required reports in this prompt and from the user log-in date and time stamp that will be tracked in part C

B. Write *at least* **two** different lambda expressions to improve your code.

C. Write code that provides the ability to track user activity by recording all user log-in attempts, dates, and time stamps and whether each attempt was successful in a file named login_activity.txt. Append each new record to the existing file, and save to the root folder of the application.

D. Provide descriptive Javadoc comments for at least 70 percent of the classes and their members throughout the code, and create an index.html file of your comments to include with your submission based on Oracle's guidelines for the Javadoc tool best practices. Your comments should include a justification for each lambda expression in the method where it is used.

Note: The comments on the lambda need to be located in the comments describing the method where it is located for it to export properly.

E. Create a README.txt file that includes the following information:
- title and purpose of the application
- author, contact information, student application version, and date
- IDE including version number (e.g., IntelliJ Community 2020.01), full JDK of version used (e.g., Java SE 17.0.1), and JavaFX version compatible with JDK version (e.g. JavaFX-SDK-17.0.1)
- directions for how to run the program
- a description of the additional report of your choice you ran in part A3f
- the MySQL Connector driver version number, including the update number (e.g., mysql-connector-java-8.1.23)

F. Demonstrate professional communication in the content and presentation of your submission.