Design Report - Project 1

By Colleen Venhuis, Kevin Gordon, and Jillian Lovas

This is a simple client-server Auto Registration application in which the data will be stored in the Oracle Server. The system is used for five different purposes: Vehicle Registration, Auto Transaction, Driver's Licence Registration, Violation Record, and Search. A user may search for driver information, violations, and vehicle history.

Program Classes:

These classes run the auto registration program, where the user can choose five different programs to add and navigate through the system.

mainMenu

void menu();
Description: Calls the main menu where a variety of submenus can be accessed. Pressing numbers 1-6 allows access to the different submenus or to exit the program.
Parameters: None
Errors: catches NumberFormatException; only numbers are allowed as input. The menu will restart if this error occurs.

---

void login();
Description: Creates the connection to the user's database by asking for their username and password. A global variable is set with the connection information.
Parameters: None
Errors: Catches SQLException. If the connection cannot be made a printout is provided with the error code and the function restarts for the user to try again.

Helpers

void checkValidity(String input, int length, String type, boolean canBeNull);
Description: Checks if the user's input has the right format required for entry or examination of the database, analyzing the input's length, the type of input (String or integer) and if the input is allowed to be null (ie, the user didn't enter any value).
Parameters:

input: The user's inputted data from the terminal.
length: The allotted length for the column in question.
type: The type of input. Can be STRING_TYPE or NUM_TYPE, indicating either letter input or number input, respectively.
canBeNull: A boolean value for indicating if the user's input is allowed to be null or not.

Errors: throws the following exceptions:

TooLongException:	Occurs when the input > length allowed
NumberFormatException:	Occurs when a number input is expected and a character is entered instead.
CantBeNullException:	Input is null but it is required to have a value.

String checkFK(String enteredValue, String fTable, String col, boolean isNum, boolean canBeNull);

Description: Checks if the foreign key value of the provided table and column is valid. Checks a single foreign key. Returns a matching foreign key from within the database.

Parameters:

enteredValue:	The foreign key value to be checked.
fTable:	The table in which to find the foreign key in (it will be a primary key in this table).
col:	The column where the foreign key will be checked for.
isNum:	A boolean value indicating whether or not the foreign key to find is a String or a number.
canBeNull:	A boolean value indicating whether the foreign key can be left null or not (if the table is not required to refer to the foreign key).

Errors: throws the following exceptions:

FKException: If there is no foreign key found in the foreign table when a value was expected.

---

boolean checkTwoPK(String col1, String col2, String entry1, String entry2, String table, boolean isSame);

Description: Checks if two primary key values of the provided table and columns are valid. Check is used when there is more than one primary key. Returns true if the two primary keys exist, false otherwise.

Parameters:

col1:	First column in the provided table to check.
col2:	Second column in the provided table to check.
entry1:	First value to check in the first column and table.
entry2:	Second value to check in the second provided column and table.
table:	The table in which to check the provided entries.
isSame:	A boolean used to indicate if the two table values are both character inputs or if one is a character input and the other is a number input.

Errors: None, but checkValidity should be ran on the entries before running this function.

---

void checkDate(String input, int length, String type);

Description: Checks the validity of the user's entered date. As long as the date is valid, no exceptions will be thrown (future dates or very unrealistic past dates will pass through this function just fine). The format MUST be yyyy-mm-dd, hyphen included.
Parameters:

input: The user's entered date. Must be in the form yyyy-mm-dd to pass.
length: The length of the date input (set as 10).
type: The type of input for this function must be TYPE_DATE.

Errors: throws the following exceptions:

TooLongException: Occurs when the input > length allowed
NumberFormatException: Occurs when a number input is expected and a character is entered instead.
DateIsNullException: Occurs when the user has entered a null String. This exception
should be used to set a flag and pass a java.sql.Date object set
to null instead of a null String type.
DateFormatException: Occurs when the date is not in the form yyyy-mm-dd.

---

boolean checkPhoto(String f);
Description: Checks if the inputted photo file location is real.
Parameters:

f: the inputted location of the photo file.

Errors: None.

---

boolean checkLicenceExists(String licenceNo);
Description: A helper that, when given a licence_no, returns true if a licence_no exists, false otherwise.
Parameters:

licenceNo: The user's inputted licence, which will be checked.

Errors: None.

---

boolean checkSinExists(String sin);
Description: A helper that when given a licence_no, returns true if a sin exists, false otherwise.
Parameters:

sin: The user's inputted sin number, which will be checked.

Errors: None.

---

void checkDecimal(String input, int length1, int length2, boolean canBeNull);
Description: A function that checks if a NUMBER(x,y) is valid.
Parameters:

input: The user's inputted value to be checked.
length1: The x length of NUMBER(x, y).
length2: The y length of NUMBER(x, y).

canBeNull: Set to true for if the value can be null, false for possible null value.

Errors: throws the following exceptions:

TooLongException:	Occurs when the input > length1 or length2 allowed
NumberFormatException:	Occurs when a number input is expected and a character is entered instead.
CantBeNullException:	Input is null but it is required to have a value.

New Vehicle Registration

void vehicleRegMenu();

Description: Displays the menu for vehicle registration.

Parameters: None.

Errors: None.

---

void confirmEntries(VehicleObj v);

Description: Does the orderly stuff for the vehicle registration. Note that this function has the same title as the one for Owner, but Owner has its own.

Parameters:

v: The Vehicle object with the user's inputted entries.

Errors: None.

---

void confirmEntries(Owner o);

Description: Does the orderly stuff for the owner registration. Note that this function has the same title as the one for Vehicle, but Vehicle has its own.

Parameters:

o: The Owner object with the user's inputted entries.

Errors: None.

---

void addVehicle();

Description: Displays the information for the user to enter appropriate information about adding a new vehicle to the database.

Parameters: None.

Errors: None.

---

boolean checkVehicle(VehicleObj vData);

Description: This function checks if the vehicle exists in the database already or not. Checking here is done by the primary key - ie, the serial number. Returns true when the serial number already exists, and false when it doesn't exist.

Parameters:

vData: The Vehicle object whose serial number will be checked.

Errors: None.

---

commitVehicle(VehicleObj veh);

Description: Commits the vehicle to the database.
Parameters:
veh: The Vehicle object to commit to the database.
Errors: None.

---

commitType(String type_id);
Description: Commits a new type into the Vehicle_type table.
Parameters:
type_id: The type_id to enter into the Vehicle_type table.
Errors: None.

---

void addOwnerMenu();
Description: adds an owner to the database.
Parameters: None.
Errors: None.

---

void addOwner();
Description: Displays the information for the user to enter appropriate information about adding a new owner to the database.
Parameters: None.
Errors: None.

---

boolean commitPeople(String sin);
Description: This function adds the person to the People database.
Parameters:
Sin: The sin number to add the people to the People table.
Errors: None.

---

void checkPrimaryOwner(String vehicleId);
Description: Checks if the vehicle already has a primary owner.
Parameters:
vehicleId: The vehicleId in which to check.
Errors: throws the following exceptions:
AlreadyPrimaryException: The vehicleId already has a primary owner.

---

void checkAnyPrimary(String vehicleId);
Description: Checks if the vehicle has any primary owners at all.
Parameters:
vehicleId: The vehicleId in which to check.
Errors: throws the following exceptions:
NoPrimaryException: The vehicleId has no primary owner.

Auto Transaction

void autoTransMenu();
Description: Displays the menu for the Auto Transaction program.
Parameters: None.
Errors: None.

---

void addTransaction();
Description: Allows the user to enter in their information for an auto transaction.
Parameters: None.
Errors: None.

---

void finTrans(TransactionObj trans);
Description: Creates the query to push the data into the database.
Parameters:
Trans: The transaction object which holds the data to push to the database.
Errors: None.

Driver Licence Registration

void driverRegMenu();
Description: Displays the menu for driver licence registration.
Parameters: None.
Errors: None.

---

void addLicence();
Description: Displays the information for adding a licence and allows the user to input info.
Parameters: None.
Errors: None.

---

void confirmEntries(DriverObj drive);
Description: Displays the menu for confirmation of the driver's licence registration. Please note that the function has the same title as the one for People, People has its own.
Parameters:
drive: The driver object with information to be committed.
Errors: None.

---

void confirmEntries(PeopleObj peep);
Description: Displays the menu for confirmation of adding a person. Please note that the function has the same title as the one for Driver, Driver has its own.
Parameters:

peep: The People object with information to be committed.

Errors: None.

---

public boolean checkDriver(DriverObj d);

Description: Checks if the primary and unique keys are already in the database for the driver's licence. Returns true when the input is already in the database, false when the input is not already in the database.

Parameters:

d: The DriverObj object to check the unique and primary keys.

Errors: None.

---

void addPeople();

Description: A function used to ask the user for info to add people to the people table.

Parameters: None

Errors: None.

---

boolean checkPerson(PeopleObj p);

Description: Checks if the person already exists. If the person is already in the database, return true. If not, return false.

Parameters:

p: The person to check for existence in the People table.

Errors: None.

---

void commitPerson(PeopleObj per);

Description: Commits the person to the database.

Parameters:

per: The person to be committed to the People table.

Errors: None.

---

void commitDriver(DriverObj dl);

Description: Commits the drier’s licence to the database.

Parameters:

dl: The driver’s licence to be committed to the Drive_licence table.

Errors: None.

---

void drivingConditionsMenu();

Description: A menu asking the user if they would like to add driving conditions.

Parameters: None.

Errors: None.

---

void addDrivingConditions();

Description: Part of the program where the user can add driving conditions.
Parameters: None.
Errors: None.

---

void displayConditions();
Description: Displays the possible driving conditions.
Parameters: None.
Errors: None.

---

void displayInstructions();
Description: Displays the instructions for adding a driving condition/restriction.
Parameters: None.
Errors: None.

---

public void addNewCondition(String condition, String idNum, Restrictions r, ConditionObj c);
Description: Checks adding an entirely new driving condition. Adds this condition automatically to the licence.
Parameters:

condition: The description of the new driving condition.
idNum: The id number for the new driving condition.
r: The restriction object to temporarily add the condition to.
c: The Driving Condition object to temporarily add the condition to.

Errors: throws the following exceptions:

TakenException: The driving condition already exists (by id number).
CantBeNullException: The user has tried to implement a driving condition with null values.

---

void commitConditions(Restrictions r, ConditionObj c);
Description: Commits the user's inputs to the database for driving condition and restrictions.
Parameters:

r: The Restrictions object to commit to the database.
c: The ConditionObj object to commit to the database (if any).

Errors: None.

String checkLicence(String input);
Description: Checks if the licence is valid. Returns if it exists in the database and false otherwise.
Parameters:

input: The user's inputted licence_no.

Errors: throws the following exceptions:

CantBeNullException: The user has entered a null value.
DNEException: The user has entered a value that does not exist in the database.

---

addC(String i, Restrictions r, ConditionObj cond);
Description: Adds a driving condition that was listed in the list already. Only adds the value to the Restriction object, not to the Driving Condition object.
Parameters:
r: The Restriction Object to check if the condition has already been added.
Cond: The Driving Condition object to check if the condition exists.
Errors: throws the following exceptions:
NumberFormatException: Input i is not a number value.
DNEException: The inputted value i does not exist in the driving_condition table.

---

void resetCondRes(Restrictions r, ConditionObj c);
Description: Resets the conditions and restriction object.
Parameters:
r: The Restriction object to clear.
c: The Driving Condition object to clear.
Errors: None.

---

boolean checkConditionNo(String i, Restrictions r, ConditionObj cond, boolean checkCondTable);
Description: Checks if the inputted number exists in the driving conditions table for that licence, or if you have already added it to the temporary object. Returns true if it exists and you can add it safely, and false if it does not.
Parameters:
i: The condition number to check.
r: The Restrictions object to check for the condition’s addition to the licence already.
cond: The Driving Condition object to check for the condition’s existence.
Errors: None.

Violation Record

void violationRecordMenu();
Description: Displays the violation record menu.
Parameters: None.
Errors: None.

---

void addRecord();
Description: Gets info from the user to add a record.
Parameters: None.
Errors: None.

---

void finRecord(ViolationObj vio);
Description: Commits the violation record to the database.
Parameters:
vio: The Violation Record to commit to the database.
Errors: None.

Search Engine

void searchMenu();
Description: Accessed via *mainMenu.menu*, displays a menu of the search options. Pressing numbers 1-4 allows access to the search submenus or return to *mainMenu.menu*.
Parameters: None
Errors: None

DriverSearch

boolean driverSearchMenu();
Description: Accessed via *SearchEngine.searchMenu*, displays a menu of the "Search by" options. Pressing numbers 1-4 allows the user to: specify what they'd like to search by, return to *SearchEngine.searchMenu* or return to *mainMenu.menu*. If a "Search by" option is selected, the user is prompted to enter a valid search term and a search method is called.
Parameters: None
Return Values: default true; false if user selects "Return to main menu" option.
Errors: catches CantBeNullException, TooLongException; the user is notified, the menu is re-displayed and the user is prompted to make a selection.

---

boolean driverSearch(String name, String licenceNo);
Description: Called by *DriverSearch.driverSearchMenu*, generates and executes a query with the provided parameters and stores the results in a ResultSet. If the ResultSet is empty, the search failed and the method returns to caller. If the ResultSet contains results, a parsing method is called, followed by a printing method.
Parameters:
name: name of the driver to search for
licenceNo: licence_no of the driver to search for
Return Values: true if results were found, false if no matches/error.
Errors: catches SQLException; the search failed and the method returns false.

Map<String,DriverObj> parseDriverSearch(ResultSet rs);

<u>Description:</u> Called by *DriverSearch.driverSearch*, parses a result set and stores the results in a one or more DriverObj(s). These DriverObj's are stored in a map, which is then returned to caller.
<u>Parameters:</u>
rs: result set containing driver information.
<u>Return Values:</u> map populated with DriverObj(s), with licence_no as key
<u>Errors:</u> catches SQLException; the result set may not have the proper columns. The user is notified and the method continues.

Void printDriverResults(Map<String,DriverObj> m);
<u>Description:</u> Called by *DriverSearch.driverSearch*, prints the driver records contained in a map.
<u>Parameters:</u>
m: a map containing driverObj(s)
<u>Return Values:</u> none
<u>Errors:</u> none

<u>ViolationSearch</u>

boolean violationSearchMenu();
<u>Description:</u> Accessed via *SearchEngine.searchMenu*, displays a menu of the “Search by” options. Pressing numbers 1-4 allows the user to: specify what they'd like to search by, return to *SearchEngine.searchMenu* or return to *mainMenu.menu*. If a “Search by” option is selected, the user is prompted to enter a valid search term and a search method is called.
<u>Parameters:</u> None
<u>Return Values:</u> default true; false if user selects “Return to main menu” option.
<u>Errors:</u> catches CantBeNullException, TooLongException; the user is notified, the menu is re-displayed and the user is prompted to make a selection.

---

boolean violationSearch(String sin, String licenceNo);
<u>Description:</u> Called by *ViolationSearch.violationSearchMenu*, generates and executes a query with the provided parameters and stores the results in a ResultSet. If the ResultSet is empty, the search failed and the method returns to caller. If the ResultSet contains results, a parsing method is called, followed by a printing method.
<u>Parameters:</u>
sin: sin of the driver to search for
licenceNo: licence_no of the driver to search for
<u>Return Values:</u> true if results were found, false if no matches/error.
<u>Errors:</u> catches SQLException; the search failed and the method returns false.

Map<String,ViolationObj> parseViolationSearch(ResultSet rs);
Description: Called by *ViolationSearch.violationSearch*, parses a result set and stores the results in a one or more ViolationObj(s). These ViolationObj's are stored in a map, which is then returned to caller.
Parameters:
rs: result set containing driver information.
Return Values: map populated with ViolationObj(s), with ticket_no as key
Errors: catches SQLException; the result set may not have the proper columns. The user is notified and the method continues.

Void printViolationResults(Map<String,ViolationObj> m);
Description: Called by *ViolationSearch.violationSearch*, prints the violation records contained in a map.
Parameters:
m: a map containing violationObj(s)
Return Values: none
Errors: none

VehicleSearch

boolean vehicleSearchMenu();
Description: Accessed via *SearchEngine.searchMenu*, displays a menu of the "Search by" options. Pressing numbers 1-3 allows the user to: specify they'd like to search, return to *SearchEngine.searchMenu* or return to *mainMenu.menu*. If search option is selected, the user is prompted to enter a valid search term and a search method is called.
Parameters: None
Return Values: default true; false if user selects "Return to main menu" option.
Errors: catches CantBeNullException, TooLongException; the user is notified, the menu is re-displayed and the user is prompted to make a selection.

---

boolean vehicleSearch(String vin);
Description: Called by *VehicleSearch.vehicleSearchMenu*, generates and executes a query with the provided parameter and stores the results in a ResultSet. If the ResultSet is empty, the search failed and the method returns to caller. If the ResultSet contains results, a parsing method is called, followed by a printing method.
Parameters:
vin: vehicle serial number to search for
Return Values: true if results were found, false if no matches/error.
Errors: catches SQLException; the search failed and the method returns false.

Map<String,ViolationObj> parseVehicleSearch(ResultSet rs);
Description: Called by *VehicleSearch.vehicleSearch*, parses a result set and stores the results in a one or more VehicleHistoryObj(s). These VehicleHistoryObj's are stored in a map, which is then returned to caller.
Parameters:
rs: result set containing vehicle information.
Return Values: map populated with VehicleHistoryObj(s), with vehicle_id as key
Errors: catches SQLException; the result set may not have the proper columns. The user is notified and the method continues.

Void printVehicleResults(Map<String,VehicleHIstoryObj> m);
Description: Called by *VehicleSearch.vehicleSearch*, prints the vehicle history records contained in a map.
Parameters:
m: a map containing VehicleHistoryObj(s)
Return Values: none
Errors: none

Objects:

These are objects and global variables that are utilized in handling the user's requests. Below are the descriptions of their use.

ConditionObj

Description: holds driving condition information such as: c_id and description. Getters and setters are provided.

Owner

Description: holds owner information such as: owner id, vehicle id, and primary owner y/n. Getters and setters are provided, in addition to some basic print methods.

Login

Description: Global variables which holds the connection to the Oracle database, entered upon running the program. stmt for most queries and pstmt for photo queries.

PeopleObj

Description: holds person information such as: sin, name, height, weight, eyecolor, haircolor, address, gender, birthday. Getters and setters are provided, in addition to some basic print methods.

Restrictions

Description: holds restriction information such as: the licence number and driving condition ids. Getters and setters are provided.

VehicleObj

Description: holds vehicle information such as: serial number, maker, model, year, colour, and type id.Getters and setters are provided, in addition to some basic print methods.

DriverObj

Description: holds driver information such as: name, licence number, address, birthday, driving class, driving conditions, expiry date, sin, photo, licence issue date. Getters and setters are provided, in addition to some basic print methods.

ViolationObj

Description: holds violation record information such as: ticket number, violator ID, officer ID, violation type, place, descriptions and violation date. Getters and setters are provided, in addition to some basic print methods.

VehicleHistoryObj

Description: holds vehicle history information such as: vehicle serial number, number of sales, average sale price and number of violations. Getters and setters are provided, in addition to some basic print methods.

IO

Description: A global variable for a universal Scanner object. This object can be used for automated testing purposes. Getters and resetters are provided.

TransactionObj

Description: holds transaction information such as: trans id, seller id, buyer id, vehicle id, date of transaction, and price. Getters and setters are provided, in addition to some basic print methods.

Exceptions:

These are custom exceptions that can be thrown by various functions in the program. To see the specifics of when they are thrown, look for functions that throw them in the Program Classes and the reason why they are thrown.

AlreadyPrimaryException
BuyerIsSellerException
CantBeNullException
DateFormatException
DateIsNullException
FKException
NoPrimaryException
NotOwnerException
TakenException
TooLongException