In this lab we will consider manipulation of sequentially-linked structures.

1. Read the general description of the exercise on page 2.

2. Quickly go over parts 1-4 just to have an idea of what we will be doing.

3. A base project named **Rental** is provided in the Lab 03-04 folder.

4. In IntelliJ Go to `File > Open > select` Project **Rental** location

5. Import **Brock Packages** (We have already done that quite a few times)

N.B: We will try to do parts 1 -2 in lab 3 and parts 3-4 in lab 4. Depending on the time we have in the lab we may cover some parts of part 3 in lab 3. There are two txt files named **Car.txt** and **SetupForm.txt** which we will use for our exercise.

Important Note: By now you should be familiar with how to open a new project / open an existing project/import Brock Packages / Setup up the correct SDKs for your project. If not, you can always go back to the Lab 1 pdf (page 1) to see the detailed description. I have added it here one more time.

Car Rental Agency

A car rental agency owns a number of cars that they rent to customers. The agency maintains a record for each car including its licence plate number (string), its current mileage (integer) and its car category (integer: 0=Economy, 1=Full Size, 2=Van and 3=SUV). When a customer arrives at the agency, a car is selected for the customer. When a car is returned, the charge is computed based on the difference between the mileage at return and the mileage at rental and the mileage rate for the class of car rented. The mileage rates are: Economy - $0.25, Full Sized - $0.50, Van - $1.00 and SUV - $1.25.

The company needs a program to keep track of their fleet of cars. This can be done using two lists: one of cars available for rent and one for cars currently rented but not yet returned. When a car is rented, it is removed from the available list and added to the rented list. Similarly, on return a car is removed from the rented list and added to the available list. At any time, the agency should also be able to display the two lists to track their fleet.

The system can be implemented representing the cars by a `Car` class and using sequentially-linked structures with items of type `Car` to track the fleet. One structure would represent the available list and one the rented list. At any time, each `Car` is on exactly one of the two lists. `Car` objects are deleted from one list and added to the other.

As a proof of concept, implement the system using sequentially-linked structures for the available and rented lists. The fleet information should be read from a text file and used to create the initial available list. The initial rented list will be empty. When a car is to be rented, the first car from the available list is removed and placed at the front of the rented list. When a car is returned, the first car from the rented list is removed, the charge computed and mileage updated, and placed at the front of the available list. When a list of the fleet is required, the two lists are traversed in turn, displaying the licence numbers to the screen. An appropriate GUI should be used to handle the rental and return of cars. It should display the licence plate and, for return should handle the input of mileage and display of the charge for the rental.

Part 1
Cars and Nodes

Estimated time: 30 minutes

The implementation based on a sequentially-linked representation will require a `Car` class to represent the individual cars and a `Node` class to allow them to be linked into a list.

Write a class `Car` that represents all that is relevant to the individual cars. The class should incorporate the relevant information about an individual car and provide a constructor that loads the car information from an `ASCIIDataFile`, appropriate accessor and updater, and operational (mutator) methods. A skeleton `Car` class is provided

The supplied `Node` class encapsulates a `Car` on a sequentially-linked list.

Write a main class that reads an `ASCIIDataFile` of car information (licence (`String`), mileage (`int`) and category (`int`)) and builds a sequentially-linked structure referenced by `avail` using a method `addAvail` to add the nodes to the front of the structure. It should then display the entries on the `avail` structure one per line to an `ASCIIDisplayer` using a method `listAvail` which performs a traversal of the available list displaying the car information to the `ASCIIDisplayer`.

Test the program using the data file `cars.txt`.

Part 2
Rental/Return

Estimated time: 45 minutes

Add the functionality to perform rental and return of the cars.

Introduce a list reference called `rented` that will reference the cars currently rented. Write methods `addRented` that adds the car to the front of the rental list; `removeAvail` and `removeRented` that remove the car from the front of the `avail` and `rented` lists, respectively, and a method `listRented` that lists the items on the `rented` list.

Write the logic to perform the rental and return as described above. Use a `BasicForm` called `form` with 4 buttons:

    0: `Rent`   move front car from the avail list to the front of the rented list.
    1: `Return`   move front car from the rented list to the front of the avail list.
    2: `List`   list the licence plates of the cars in the avail list and then the rented list.
    3: `Quit`

to select the operation to perform. In cases 0 and 1, write a message to the display indicating what happened.

Test the program using the file `cars.txt`.

Part 3
Adding the Interface

Estimated time: 30 minutes

To provide a functional prototype, a user interface will be necessary. The program should repeatedly present a form such as:
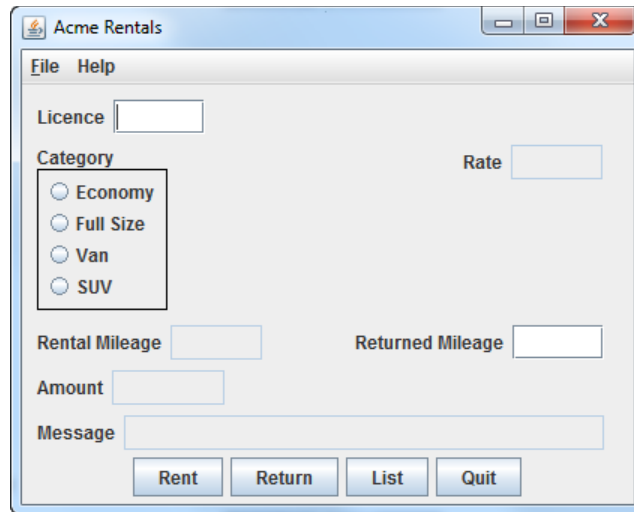

Figure 1 GUI for Acme Rentals

The file `setupForm.txt` contains the method `setupForm` which builds the form as above.

When the user presses `Rent`, the program displays the information on the car to be rented (first on the available list) on the form such as:
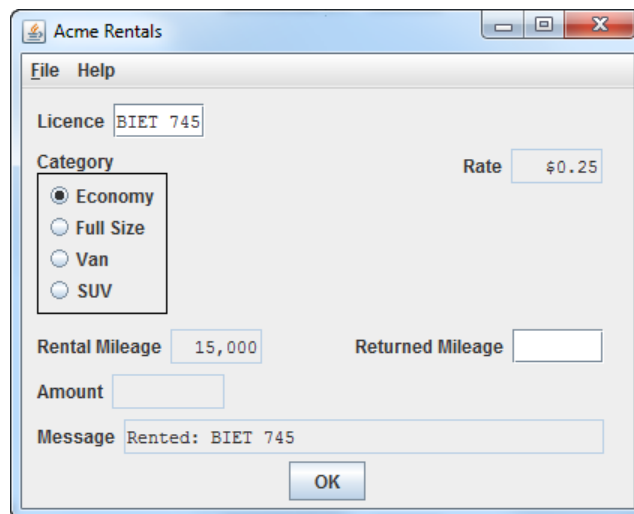

Figure 2 Car Rental

If there are no cars available for rent, a message to that effect should be displayed in the `Message` field. Upon pressing `OK`, the rental form (figure 1) is presented again.

When the user presses `Return`, the program displays the information on the car to be returned (first on rented list) on the form and requests the return mileage. If there are no cars currently rented a message should be displayed in the `Message` field.

Figure 3 Car Return

After the user enters the the mileage and presses OK, the rental cost is computed and displayed on the form and the car returned.

Figure 4 Cost of Rental

Upon pressing OK, the form in Figure 1 will be presented again allowing the user to rent or return another car.

When the user presses List, the program displays a list of the cars in the available list followed by the cars in the rented list to the ASCIIDisplayer (figure 5) and displays the message Listed on the form.
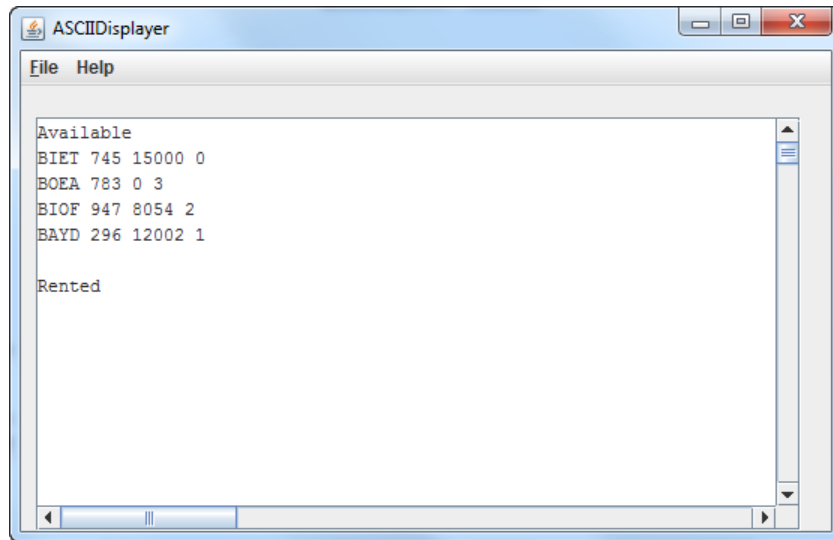
Figure 5 List of Cars in the Fleet

When the user presses OK (on the form), the form in Figure 1 is presented again.

When the user presses Quit, the program terminates.

Test the program using the file cars.txt.

Part 4
Realistic Rental

Estimated time: 45 minutes

The rental agency has accepted the proof of concept so it is time to implement the complete system. When a car is to be rented, the customer specifies a car class and the first car in the available list that matches that class is removed, rented and added to the front of the rented list. When a car is returned, the rented list is searched for the car with the supplied licence plate. This item is removed from the rented list, the charge computed and mileage updated and the car is added to the end of the available list. The GUI and listing of the fleet information is the same as in Part 3.

Modify the main class to support the modified process for car rental/return. When a car is rented, the category field of the form is used to determine the category of car to be rented. The removal from the available list is based on category. The rented car can still be added to the front of the rented list since the order of this list is irrelevant. When a car returned, the licence field of the form is used to determine the car being returned. The removal from the rented list is based on licence plate. The returned car is added to the end of the available list.