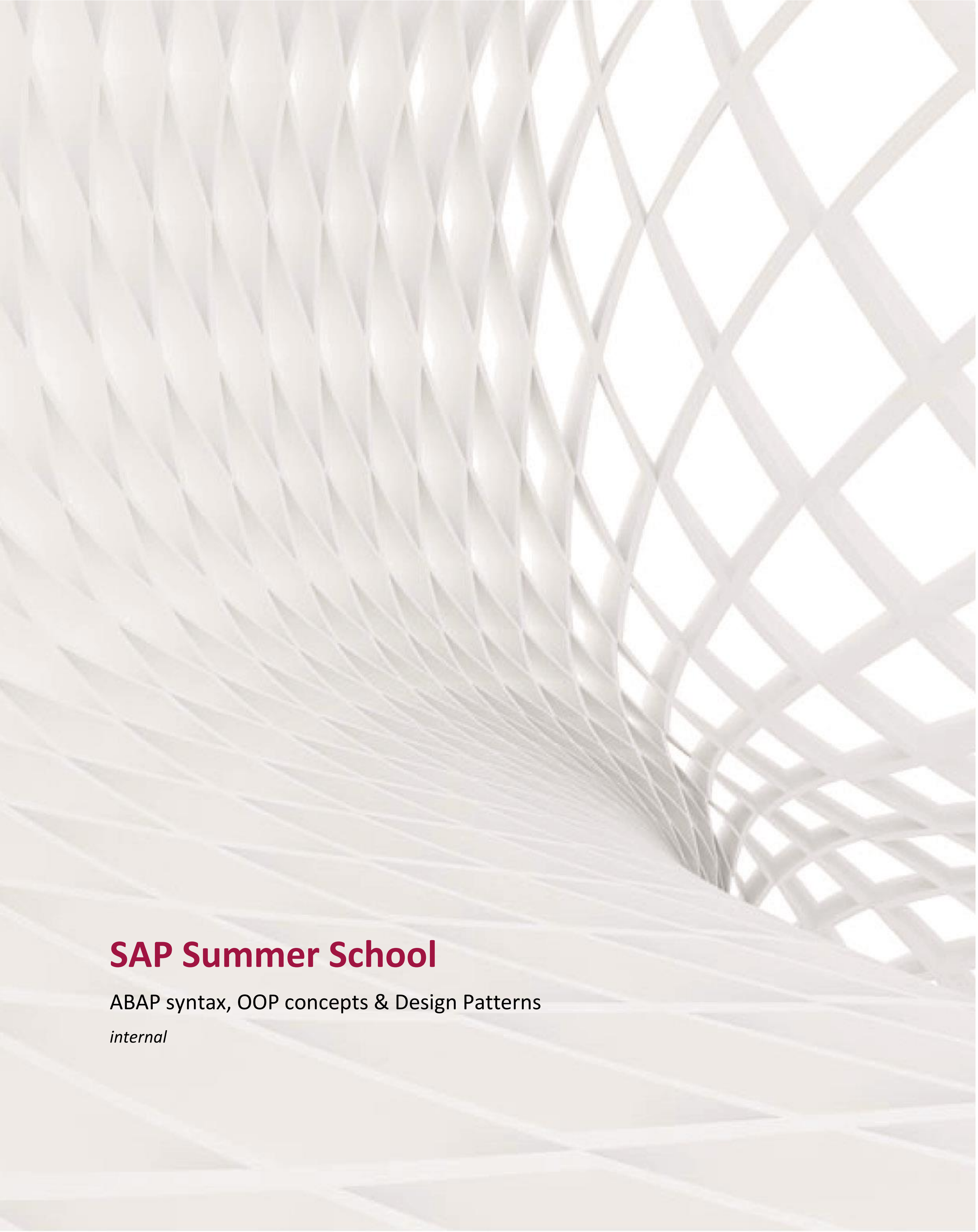


SAP Summer School

ABAP syntax, OOP concepts & Design Patterns

internal

value – inspired by people

1 Table of Contents

1 **Table of Contents** **2**

2 **Introduction** **3**

3 **Requirements** **4**

3.1 Application Structure 4

3.2 Create Vehicle Factory 5

3.3 Test the Rental System Application 6

3.3.1 Store the list of vehicles 6

3.4 Create Rental Interface 6

3.4.1 Check User is Regular Customer 6

3.4.2 Create a new rental 7

3.5 Implement the Rental interface 7

3.5.1 Check User is Regular Customer - Implementation 7

3.5.2 Create a new Rental - Implementation 8

3.5.3 Test your implementation 9

3.5.4 Calculate and display the average consumption for each vehicle in the list 9

2 Introduction

The purpose of this laboratory is to:

- Get used with ABAP syntax, keywords, code structure
- Create and call methods
- Apply OOP concepts
- Use design patterns

3 Requirements

The focus of this laboratory is to extend the vehicles hierarchy you created in ‘SAP Summer School Exercise 4_part 1” with the classes that will handle the operations needed in our Vehicle Rental Company.

Our application needs to provide the following functionalities:

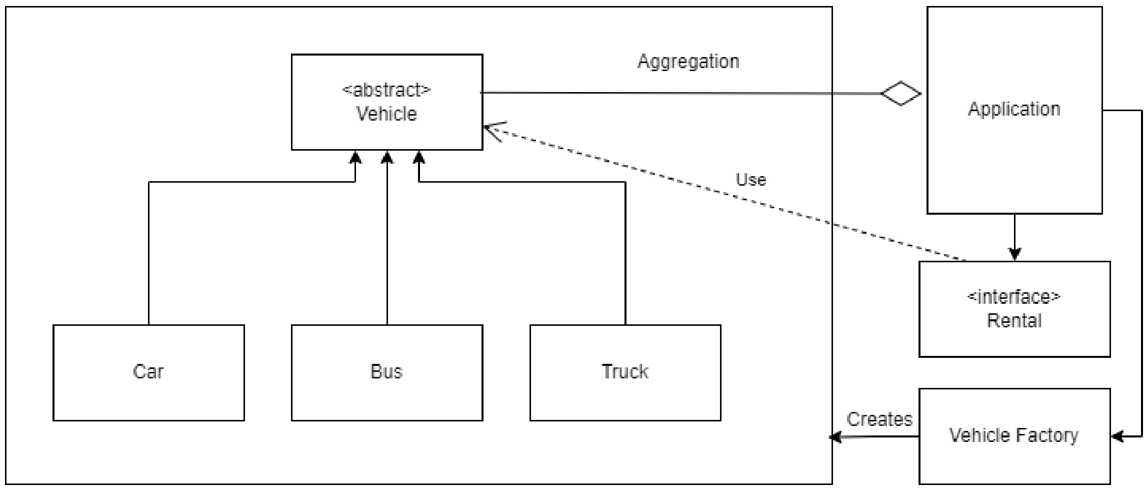
- Create a new vehicle of a given type (car, truck, or bus).
- Needs to hold a list of vehicles (cars, buses, and trucks).
- Add a new vehicle into the retained list.
- Check if the user is a regular customer. To do that, you need to check if the user already did at least one rental in the past.
- Create a new rental, receiving:
  - user id
  - vehicle id
  - period start date
  - period end date

  Total price is calculated as follows: number_of_days * price/day.
  The price per day is different for each vehicle type:
    - Cars: 10 EUR
    - Buses: 50 EUR
    - Trucks: 100 EUR

  If the user is already a customer, and the rental period exceeds 4 days, a 5% discount should be applied to the rental price.
- Calculate and display the average consumption for each vehicle in the list.

3.1 Application Structure

Here is the application structure diagram, that will guide you through this laboratory:



3.2 Create Vehicle Factory

Factory Method is a creational design pattern that provides an interface for creating objects based on the specified type.

We will create a Vehicle Factory class that based on the specified type (car, bus, or truck) will create a new vehicle object.

- Create a new class with the following naming convention:
  ZCL_<initials>_VEHICLE_FACTORY
- Define a local type in this class, that will hold the attributes needed in order to create a vehicle, as follows:

```
TYPES: BEGIN OF mty_vehicle_data,
         manufacturer_name TYPE zsv_manufacturer_name,
         model_name        TYPE zsv_model_name,
         model_year        TYPE zsv_model_year,
         no_passengers     TYPE zsv_no_seats,
         max_cargo         TYPE zsv_max_cargo,
       END OF mty_vehicle_data.
```

- Define a new method for creating a vehicle: CREATE_VEHICLE that receives as input parameters the vehicle type (“CAR”, “BUS”, “TRUCK”) and the vehicle data.
- This vehicle data will have as type the local type defined above in which you can fill only the relevant attributes for the vehicle created (e.g.: for creating a car, the number of passengers and maximum cargo are not needed).
- The method returns a reference to a vehicle object. You specify a returning reference parameter as follows:

```
CLASS-METHODS: create_vehicle
  ...
  RETURNING VALUE(rr_vehicle) TYPE REF TO zsv_cl_vehicle.
```

- Based on the vehicle type received as a string, you can create a new vehicle of the corresponding class as follows:

```
rr_vehicle = NEW zsv_cl_bus( iv_vehicle_id = cl_system_uuid=>create_uuid_c22_static( )
                             iv_manufacturer_name = is_vehicle_data-manufacturer_name
                             ... ).
```

- Create a new interface to store the constants used in the application:
  ZIF_<initials>_GLOBAL_CONSTS.
- For each vehicle type mentioned above, declare a new constant.

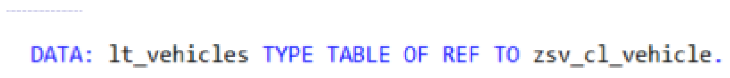
```
CONSTANTS: mc_car_type   TYPE string VALUE 'CAR',
```

3.3 Test the Rental System Application

Create a new class with the following naming convention: ZCL_<initials>_MAIN_PROGRAM, that will implement interface IF_OO_ADT_CLASSRUN.

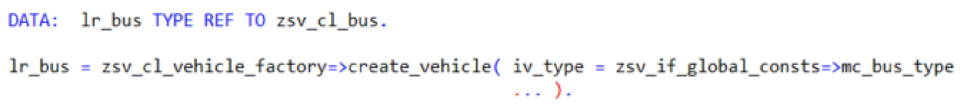
3.3.1 Store the list of vehicles

- In the main method provided by the interface declare a local internal table that will hold the list of vehicles in your application.
- As the vehicles are objects, the table need to store a list of references. You can declare it as follows:



```
DATA: lt_vehicles TYPE TABLE OF REF TO zsv_cl_vehicle.
```

- You should use now your previously created factory in order to create multiple types of vehicles and add them to your list.
- Call the create_vehicle method in order to create at least one vehicle of each type.



```
DATA:  lr_bus TYPE REF TO zsv_cl_bus.

lr_bus = zsv_cl_vehicle_factory=>create_vehicle( iv_type = zsv_if_global_consts=>mc_bus_type
                                                 ... ).
```

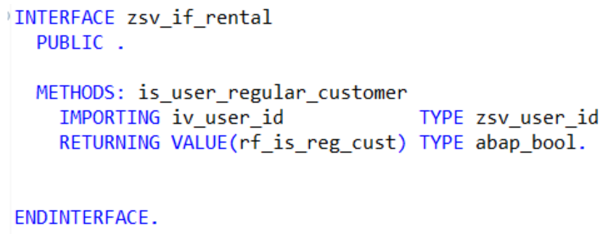
- For the type, use the constants declared into the constants interface. You can access them as follows: ZIF_<initials>_GLOBAL_CONSTS=><constant_name>.
- Add the created objects to the list of vehicles declared above. (Check VALUE and BASE keywords from the previous course - https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-US/abenvalue_constructor_params_itab.htm)

3.4 Create Rental Interface

- Since our application needs to provide multiple common operations, we will use an interface to declare how their signatures should look like.
- Create a new interface for rentals with the following naming convention: ZIF_<initials>_RENTAL. Note that for interfaces we will use convention "IF".

3.4.1 Check User is Regular Customer

- Declare the method for checking if the user is a regular customer as follows:



```
INTERFACE zsv_if_rental
  PUBLIC .

  METHODS: is_user_regular_customer
    IMPORTING iv_user_id          TYPE zsv_user_id
    RETURNING VALUE(rf_is_reg_cust) TYPE abap_bool.


ENDINTERFACE.
```

You can notice here the naming convention used. (I – Importing, R – returning, V – Value, F – Flag) This is useful for helping developers know the type and scope of a parameter inside a method. Beside these conventions, it is always recommended to use meaningful names for the parameters, that suggest their purpose.

The method has one importing parameter, that will be provided by the method caller, and returns one flag which indicates if the user is a regular customer (ABAP_BOOL is used for Boolean type, that can take values "true" or "false")

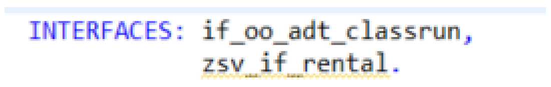
3.4.2 Create a new rental

- Add the method for creating a new rental.
- This method will receive as input parameters the user id, vehicle id, period start date and period end date and returns a flag if the rental was successfully created.

  Note: Interfaces are similar with classes declaration; we only have the area where methods signature is defined.

3.5 Implement the Rental interface

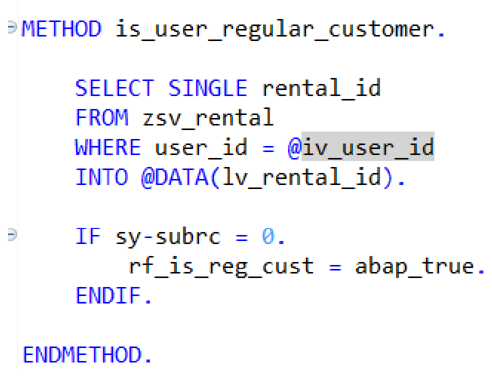
- In the class ZCL_<initials>_MAIN_PROGRAM that you previously created, implement the rental interface.



```
INTERFACES: if_oo_adt_classrun,
            zsv_if_rental.
```

3.5.1 Check User is Regular Customer - Implementation

- A requirement was to be able to check if a given user is a regular customer. In order to do that, you need to implement the method "is_user_regular_customer" from Rental interface.



```
METHOD is_user_regular_customer.

    SELECT SINGLE rental_id
    FROM zsv_rental
    WHERE user_id = @iv_user_id
    INTO @DATA(lv_rental_id).

    IF sy-subrc = 0.
        rf_is_reg_cust = abap_true.
    ENDIF.

ENDMETHOD.
```

- We will select one rental id, if it exists into the database table "z<initials>_rental".
- Here we used in line declaration "INTO @DATA(lv_rental_id)" which only declares a local variable lv_rental_id that will store the result of the select statement. We don't need the id here, but the select statement should store the result into something, otherwise you will get a syntax error.
- After a select statement is executed, the "sy-subrc" system variable is set. This will be set to:
  - 4, if no data was found
  - 0, if the select statement was executed successfully and data was read.

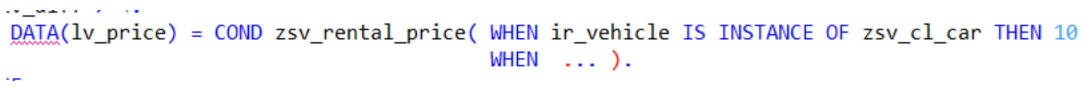
  Based on this, we can set the returning parameter to "true" if the user is a regular customer.

Otherwise, the returning parameter has value “false” by default and stays unchanged.

3.5.2 Create a new Rental - Implementation

- In order to create a new rental, an entry needs to be constructed in a similar way you did in the second laboratory when populating the database table.
    - The price is calculated as mentioned in the requirement:
        - Cars: 10 EUR
        - Buses: 50 EUR
        - Trucks: 100 EUR

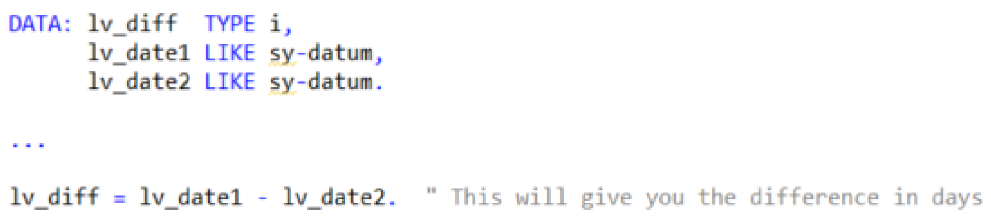
      In order to check which vehicle type you have, you can use “IS INSTANCE OF” keywords, which checks if the reference provided is an instance of the specified class.



```
DATA(lv_price) = COND zsv_rental_price( WHEN ir_vehicle IS INSTANCE OF zsv_cl_car THEN 10
                                        WHEN  ... ).
```

      Here we used COND operator, which has one branch for each condition and its corresponding result. (For syntax see https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-us/abenconditional_expression_cond.htm)

    - For calculating the period between the start and end date, to check if it exceeded 4 days, the following syntax can be used:



```
DATA: lv_diff  TYPE i,
      lv_date1 LIKE sy-datum,
      lv_date2 LIKE sy-datum.

...

lv_diff = lv_date1 - lv_date2.  " This will give you the difference in days
```

    - For checking if the user is a regular customer, you can call the method created above with the user id.
    - If the period exceeds 4 days and the user is a regular customer, you need to apply a 5% discount.

      To apply a 5% discount, you can just multiply by 5 and divide by 100.

      Now you should have the final price.

    - For creating a new rental id you can use the method for generating a new UUID from the previous laboratory: CL_SYSTEM_UUID=>CREATE_UUID_C22_STATIC( ).
    - The currency can be hardcoded to “EUR”.

  After you have all the data, the new rental can be constructed and inserted into the database table:

```
ls_rental = VALUE #( rental_id = cl_system_uuid=>create_uuid_c22_static(  )
                     user_id    = iv_user_id
                     vehicle_id = ir_vehicle->get_id(  )
                     start_date = iv_period_start
                     end_date   = iv_end_period
                     price      = lv_price
                     currency   = iv_currency ).

INSERT zsv_rental FROM @ls_rental.
```

You can notice here a warning caused by the create_uuid_c22_static method call. This method could throw an exception of class CX_UUID_ERROR. In order to avoid that, you can surround the whole statement by a TRY - CATCH – ENDTRY block. (see this course material)

If the insert is successful (see sy-subrc variable from previous exercises), the returning flag should be set to true. Otherwise, it remains false.

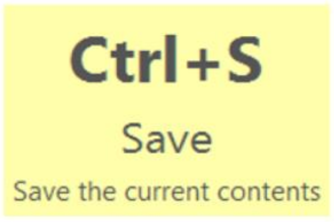
3.5.3 Test your implementation

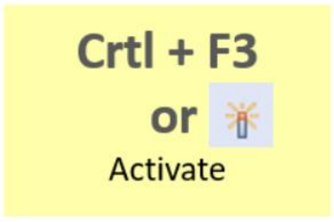
- Create a new report where you select from the database tables created at the previous exercise from the Data Dictionary Chapter one vehicle id and one user id in order to create a new rental.
- For the select statement see method is_user_regular_customer created before.
- Call the method create_rental with the vehicle, user, and a period of your choice.
- Check the returned flag and display an error/message to the user. For writing a message to the user, out->write(<text>) statement can be used.
- Check the rentals database table for the new rental.

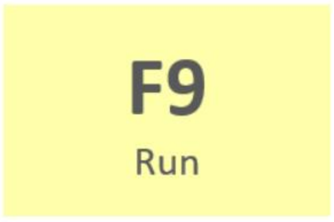
3.5.4 Calculate and display the average consumption for each vehicle in the list

- In the main method of the class, loop over the list of vehicles and write to the console the estimated average consumption for each one of them.

Finally, save, activate and run your class. Check the output in the console/using the debugger.


Ctrl+S
Save
Save the current contents


Crtl + F3
or
Activate


F9
Run