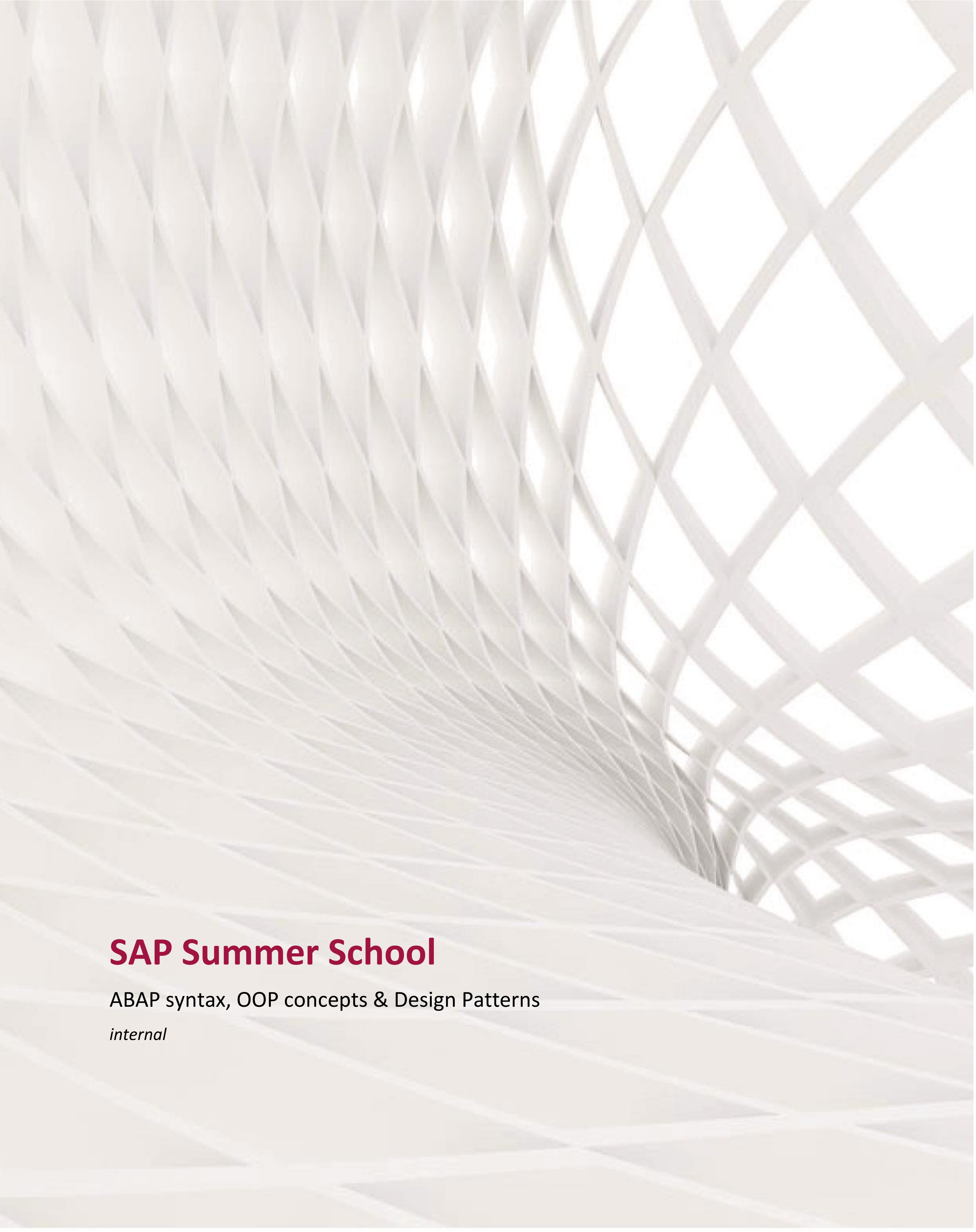


SAP Summer School

ABAP syntax, OOP concepts & Design Patterns

internal

value – inspired by people

1 Table of Contents

2 Introduction

The purpose of this laboratory is to:

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

3 Requirements

The focus of this laboratory is to create the application layer that will handle the operations needed in our Vehicle Rental Company.

In order to do that, the following is needed:

- The application should be able to hold a list of vehicles of 3 types: cars, buses, and trucks.
- For each vehicle we need to retain the following:
  - Id
  - Manufacturer name
  - Model name
  - Model year
- Each bus has in addition the maximum number of passengers.
- Each truck has in addition the maximum cargo.
- Each vehicle can estimate the average consumption. The formula is the following:

  fuel = 100 * weight * factor

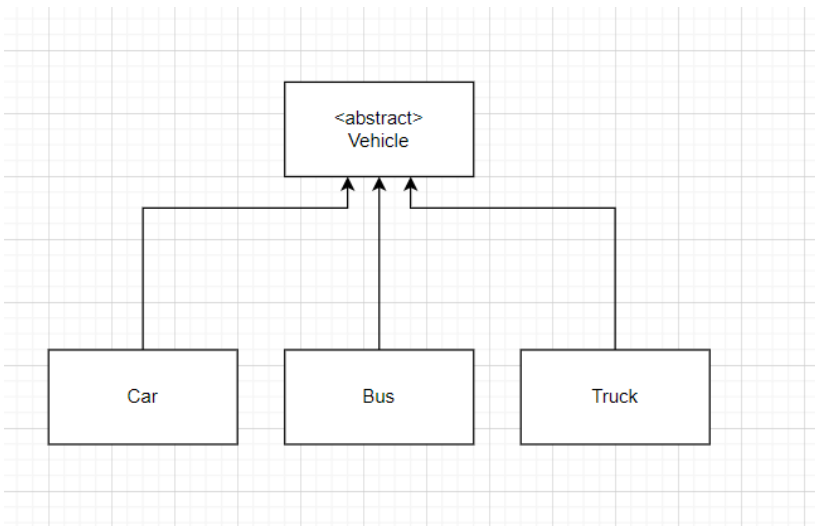
  The weight is calculated for each vehicle as follows:
  - Cars: weight = 4 * average_speed
  - Buses: weight = max_passengers * average_speed
  - Trucks: weight = max_cargo

  For the exercise purpose we will use:
  - average_speed = 50
  - factor = 0.15

In the next sections you will be guided through the creation of the vehicle hierarchy classes needed to implement the above requirements. Some of the methods are given, while some of them you need to do by yourself.

3.1 Application Structure

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

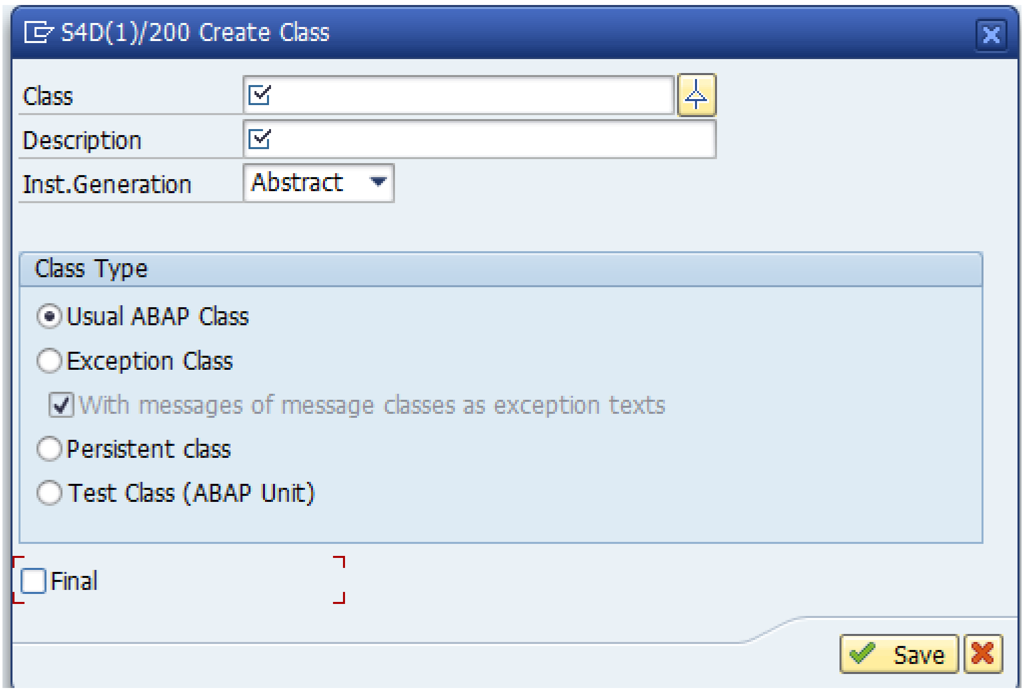


3.2 Create the abstract class Vehicle

Since you will have 3 types of vehicles in the application, each one with its additional attributes and behavior for average consumption estimation, you can use a prototype pattern. (See https://refactoring.guru/design-patterns/prototype - shapes example)

Therefore, an abstract class should be used to represent the vehicles. This will hold the common attributes of vehicles mentioned above.

- Create a new abstract ABAP Class: (in package Z_CAR_RENTAL_<initials>)



S4D(1)/200 Create Class

Class	✓
Description	✓
Inst.Generation	Abstract

Class Type

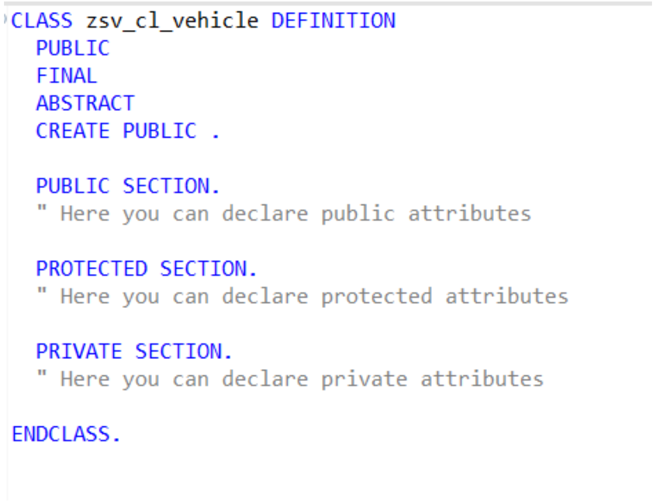
- [✓ Usual ABAP Class]
- Exception Class
- ✓ With messages of message classes as exception texts
- Persistent class
- Test Class (ABAP Unit)

☐ Final

Note: Don’t forget to use your initials in each object you create. (e.g.: ZCL_<initials>_VEHICLE)

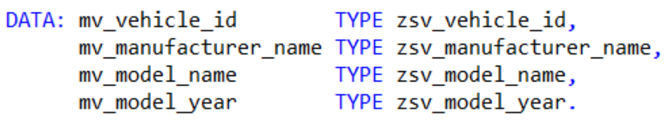
Note: The class is created by default “FINAL” which does not allow it to have subclasses. You need to uncheck the Final checkbox.

- The common attributes of all vehicles need to be declared under the right section:



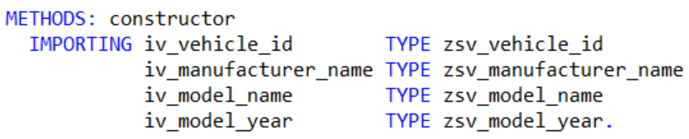
```
CLASS zsv_cl_vehicle DEFINITION
  PUBLIC
  FINAL
  ABSTRACT
  CREATE PUBLIC .

  PUBLIC SECTION.
  " Here you can declare public attributes

  PROTECTED SECTION.
  " Here you can declare protected attributes

  PRIVATE SECTION.
  " Here you can declare private attributes

ENDCLASS.
```

  Attributes can be declared as follows:



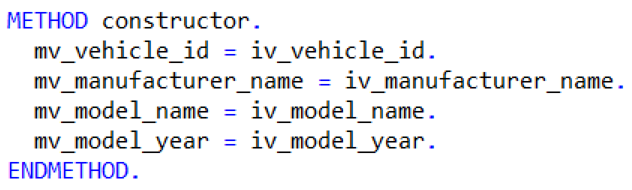
```
DATA: mv_vehicle_id        TYPE zsv_vehicle_id,
      mv_manufacturer_name TYPE zsv_manufacturer_name,
      mv_model_name        TYPE zsv_model_name,
      mv_model_year        TYPE zsv_model_year.
```

  Note:
  - Naming convention “M” is used for member variables.
  - Instance attributes of a class are declared using the “DATA” keyword.
  - Each attribute should be declared with its corresponding type.
- After you declared the attributes, you need to define a proper constructor.
- You have to first declare it under class declaration as follows:



```
METHODS: constructor
  IMPORTING iv_vehicle_id        TYPE zsv_vehicle_id
            iv_manufacturer_name TYPE zsv_manufacturer_name
            iv_model_name        TYPE zsv_model_name
            iv_model_year        TYPE zsv_model_year.
```

- Afterwards, in the implementation you need to pass the values from the import parameters into class attributes:

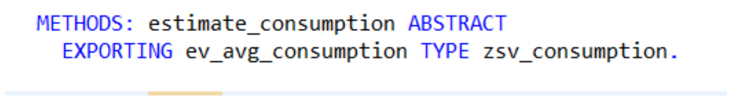


```
METHOD constructor.
  mv_vehicle_id = iv_vehicle_id.
  mv_manufacturer_name = iv_manufacturer_name.
  mv_model_name = iv_model_name.
  mv_model_year = iv_model_year.
ENDMETHOD.
```

- In a similar way, getter methods should be defined for each attribute.

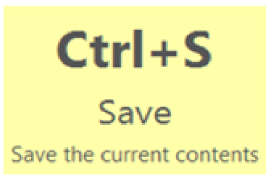
3.2.1 Estimate the average consumption of the vehicle

- It is important to consider that each vehicle type has an own formula for calculating the average consumption, therefore you will make a new abstract method “ESTIMATE_CONSUMPTION” in this class, that will be overridden in all vehicle subtype classes.
- You can create a new data element for the average consumption. (We don’t necessarily care for the decimals here, so INT4 type can be used)
- Afterwards, you can declare the method as follows:



```
METHODS: estimate_consumption ABSTRACT
  EXPORTING ev_avg_consumption TYPE zsv_consumption.
```

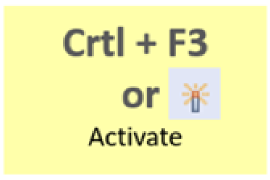
Finally, save and activate the class.



Ctrl+S

Save

Save the current contents



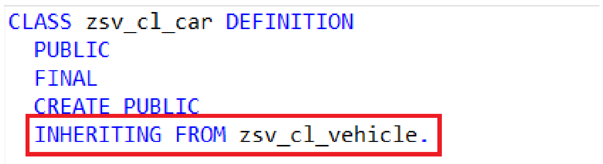
Crtl + F3

or

Activate

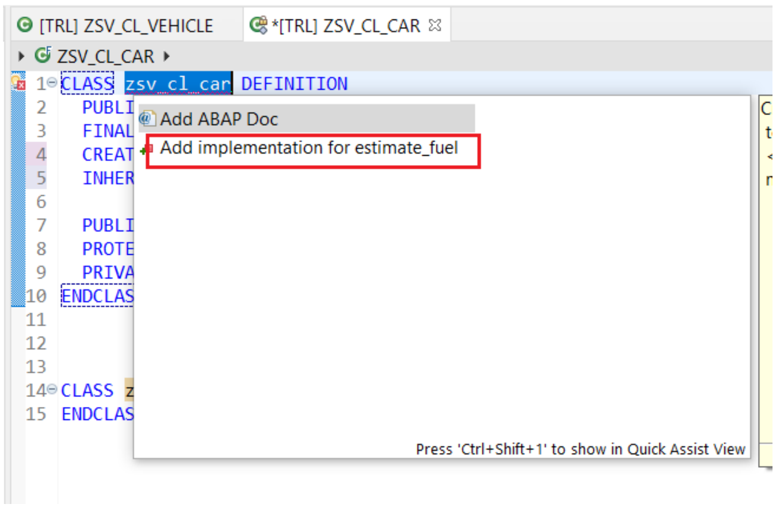
3.3 Create Class Car

- Create a new class for the first vehicle type. The following naming convention should be used: ZCL_<initials>_CAR.
- This class will extend the abstract class Vehicle created above as follows:



```
CLASS zsv_cl_car DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC
  INHERITING FROM zsv_cl_vehicle.
```

- When extending this abstract class, the abstract method needs to be implemented. You can implement it by clicking the error icon displayed here:



[TRL] ZSV_CL_VEHICLE   *[TRL] ZSV_CL_CAR

ZSV_CL_CAR

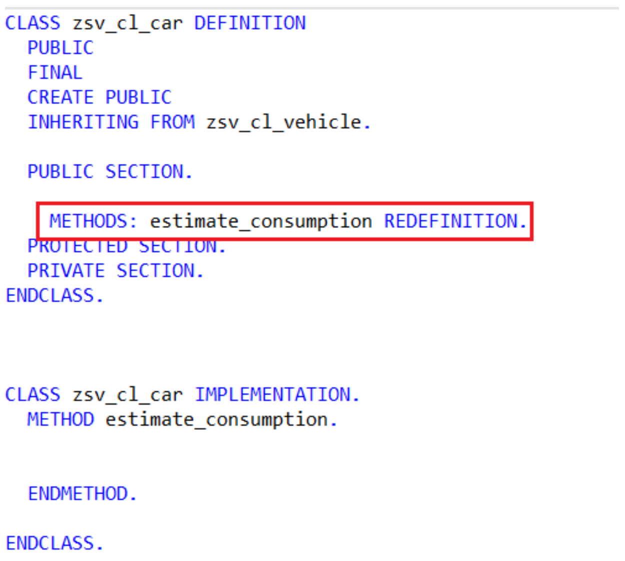
```
CLASS zsv_cl_car DEFINITION
  PUBLI
  FINAL
  CREAT
  INHER

  PUBLI
  PROTE
  PRIVA
ENDCLAS



CLASS z
ENDCLAS
```

Add ABAP Doc

Add implementation for estimate_fuel

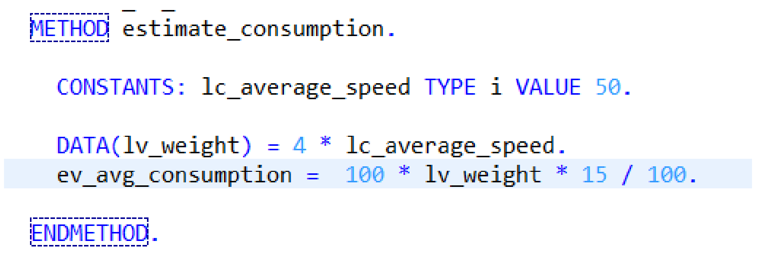
Press 'Ctrl+Shift+1' to show in Quick Assist View

Afterwards, the abstract method is added as redefinition in the class definition area, and you can write your implementation below:



```
CLASS zsv_cl_car DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC
  INHERITING FROM zsv_cl_vehicle.

  PUBLIC SECTION.

    METHODS: estimate_consumption REDEFINITION.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.



CLASS zsv_cl_car IMPLEMENTATION.
  METHOD estimate_consumption.


  ENDMETHOD.

ENDCLASS.
```

- The corresponding formula is applied from the requirement:

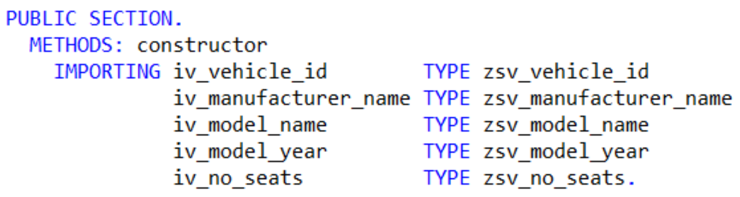


```
METHOD estimate_consumption.

  CONSTANTS: lc_average_speed TYPE i VALUE 50.

  DATA(lv_weight) = 4 * lc_average_speed.
  ev_avg_consumption =  100 * lv_weight * 15 / 100.

ENDMETHOD.
```

Here we used a constant for the "Average Speed", as it has a fixed value.

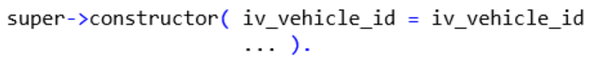
3.4 Create Class Bus

- In a similar way, create the class for "Bus" vehicles.
- Method "estimate_consumption" needs to be redefined with the corresponding formula.
- The additional attribute "Maximum number of passengers" needs to be added.
- Note that when adding a new attribute, you will need an appropriate constructor which will set the Vehicle superclass attributes and the new attribute as well.
- In order to do that, a new constructor should be declared with all the attributes as follows:



```
PUBLIC SECTION.
  METHODS: constructor
    IMPORTING iv_vehicle_id        TYPE zsv_vehicle_id
              iv_manufacturer_name TYPE zsv_manufacturer_name
              iv_model_name        TYPE zsv_model_name
              iv_model_year        TYPE zsv_model_year
              iv_no_seats          TYPE zsv_no_seats.
```

In the implementation, the super class should be called to set the Vehicle class attributes. For this, the "super" reference is used, which points to the super class instance.



```
super->constructor( iv_vehicle_id = iv_vehicle_id
                    ... ).
```

3.5 Create Class Truck

In a similar way, create the class for “Truck” vehicles. This will have an additional attribute for “Maximum Cargo” and an own way of estimating the consumption.