ABAP Course

Chapter 7 – ABAP Objects

Lecturer: Robert Meyer, UCC Technische Universität München
Author: Marcus Homann, André Bögelsack, Valentin Nicolescu

Agenda

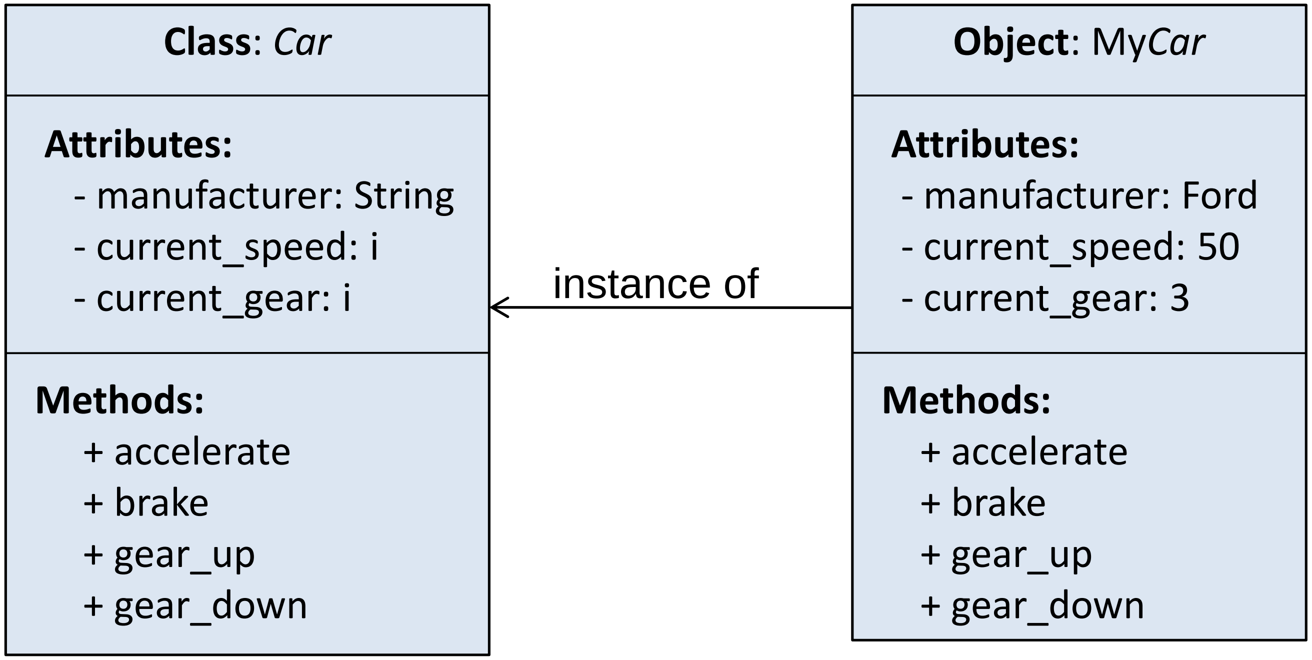
- Introduction
- Principles of object orientation
- Definition of classes
- Implementation of classes

Introduction ABAP Objects

- Object-oriented enhancement to ABAP (since Release 4.6)
- Additive extension to the existing language
  - Advantage: backwards compatibility to existing code
  - Disadvantage: Mixture of object-orientated and procedural programming model possible
- Continuous Enhancement of ABAP Objects:
  - Object Services (since Release 6.1)
    - facilitates Object Persistency (in Database)
  - Shared Objects (since 6.4)
    - Cross-program access to objects stored in application server-wide shared memory area
- Main tools: ABAP Editor (**SE38**) and Class Builder (**SE24**) (both usually accessed through **SE80**)

Principles of Object Orientation I

- **Classes** define attributes and methods of objects and act as templates for objects
- **Objects** are instances of classes
  - Attributes represent the status of an object; have specific values
  - Methods represent the procedures of an object



Source: Author's design

Principles of Object Orientation II

- **Encapsulation**:
  Interaction takes place only by defined interfaces. The implementation of a class is invisible outside the class.
- **Polymorphism**:
  Different classes can have the same interface and thus accessed the same way.
- **Inheritance**:
  A new class can inherit attributes and methods from an existing class and thus be extended by additional attributes and methods.
- **Abstraction**:
  Classes and objects are defined by their interfaces and functionality rather than their implementation details.

Definition of a class I

- **Every Class definition has two parts**:
  1. Declaration part
     - Specifies attributes (name, data type, visibility)
     - Specifies method signatures (name, parameters, visibility)
  2. Implementation part
     - includes the execution statements of the specified methods

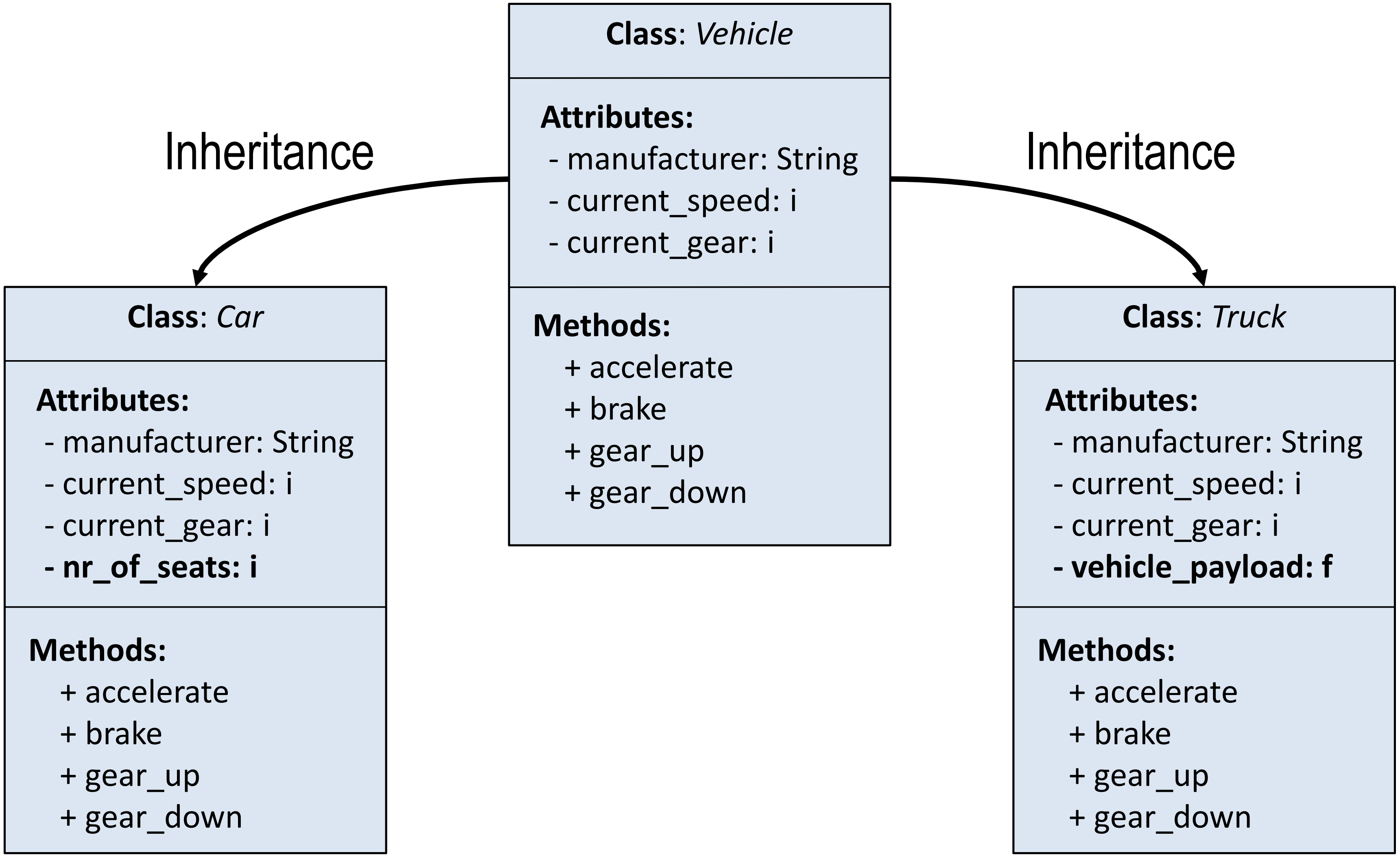
```
Example:

CLASS <class_name> DEFINITION.
 …
ENDCLASS.
```

$\left.\right\}$ Declaration

```
CLASS <class_name> IMPLEMENTATION.
 …
ENDCLASS.
```

$\left.\right\}$ Implementation

Example for Inheritance



Soure: Author's design

Visibility of Methods and Attributes

- Visibility specification keywords: **public / private / protected**
  - **Public** methods/attributes can be accessed without limitations
  - **Private** methods/attributes can be accessed only inside the class
  - **Protected** methods/attributes can be accessed inside the class and from subclasses

```
CLASS <class_name> DEFINITION.

  PUBLIC SECTION.

  PROTECTED SECTION.

  PRIVATE SECTION.

ENDCLASS.
```

$\}$ Declaration / Definition

Visibility of Instantiation / Constructor

- Specifies, who is able to call the constructor to instantiate objects
- **3 instantiation-types**
  - **Public** constructor can be accessed without limitations
  - **Private** constructor can be accessed only inside the class
  - **Protected**: constructor can be accessed inside the class and by subclasses
- Use Case: E.g. Factory-, Singleton Pattern

```
CLASS <class_name> DEFINITION
                   CREATE PUBLIC|PRIVATE|PROTECTED.

ENDCLASS.
```

Declaration of attributes and methods

```
CLASS <class_name> DEFINITION.

   PUBLIC SECTION.
          DATA: <variable_name> TYPE <data_type>.

          METHODS:
            <method_name>
                [IMPORTING  <parameter_name> TYPE <data_type>]
                [EXPORTING <parameter_name> TYPE <data_type>]
                [CHANGING <parameter_name> TYPE <data_type>]
                [RETURNING VALUE(parameter_name) TYPE <data_type>].

   PRIVATE SECTION.
          DATA: …
          METHODS: …

ENDCLASS.
```

- IMPORTING / EXPORTING: **Call by Value**, actual parameters are copied to the formal parameters
- CHANGING: **Call by Reference**, actual parameters are bound to the formal parameters

Class vs. Instance Components

- Instance components
  - Instance components exist for each instance of a class
  - Usual case
- Class components (attributes and methods)
  - Class components exist only once for all instances of a class
  - Use cases: Shared attributes (e.g. instance counter), factory methods, etc.

```
DATA <attr> TYPE <data_type> [READ-ONLY].
METHODS <method> [IMPORTING …]
        … [RETURNING VALUE(r) TYPE data_type]
```

$\Big\}$ Instance Components

```
CLASS-DATA <attr> TYPE <data_type> [READ-ONLY].
CONSTANTS <attr> TYPE <data_type> VALUE <val>.
CLASS-METHODS <method> [IMPORTING …]
        … [RETURNING VALUE(r) TYPE data_type]
```

$\Big\}$ Class Components

Implementation of a class

```
CLASS <class_name>
IMPLEMENTATION.
   …
   METHOD <method_name>.
        …ABAP-Code…
   ENDMETHOD.
   …
ENDCLASS.
```

Calling methods and accessing attributes

- Instance method: (access with “->“)

  ```
  <instance_name>-><method_name>( <import_parameter> = value ).
  ```

  - **Caution:** spaces are important within the bracket to separate parameters and values

- Class method: (access with “=>“)

  ```
  <class_name>=><method_name>( <import_parameter> = value ).
  ```

- Instance attributes: (access with “->“)

  ```
  <instance_name>-><attribute_name>.
  ```

- Class attributes: (access with ”=>“)

  ```
  <class_name>=><attribute_name>.
  ```

Constructor and Object References

- Automatically called by “CREATE OBJECT“
- Explicitly or implicitly defined method “constructor“ used for creating new instances
  - In case a constructor implementation is missing, a new instance is created by means of a default constructor
  - In case of explicit implementation within the PUBLIC SECTION additional steps can be executed when creating new instances (e.g. assigning of default values)
- There is no “destructor” in ABAP
- Object instances are assigned to Reference Variables (“REF TO”)

```
PUBLIC SECTION
   METHODS: constructor IMPORTING
                         a_manufacturer type string
                         a_no_of_seats type i.

DATA r_mycar TYPE REF TO Car.
CREATE OBJECT r_mycar EXPORTING
                         a_manufacturer = ‘Ford‘
                         a_no_of_seats=5.
```

Inheritance

- ABAP Objects supports only **single inheritance** (like Java): a class can only have one parent class
- Inheritance means that a class owns all attributes and methods from its parent class
- New attributes or methods can be defined or existing methods redefined in order to specialize the subclass
- Keyword: “INHERITING FROM”
- Redefinition of methods from parent class with keyword “REDEFINITION”

```
CLASS car DEFINITION INHERITING FROM vehicle
   PUBLIC SECTION
       METHODS: accelerate REDEFINITION.
ENDCLASS.

CLASS car IMPLEMENTATION.
  …
ENDCLASS.
```

Miscellaneuos

- Self-Reference : Variable “me”
  - E.g. me->no_of_seats = 4.
- Garbage Collector
  - Deletes all object instances that have no reference
  - Is called perodically by the ABAP Application Server
- global vs. local classes
  - local classes can only be used in the program where they are defined
  - global classes can be used in all programs of the same ABAP Application server
  - Global classes can only be defined through the tool “Class Builder”
- Start processing event block: “START-OF-SELECTION”
  - Similar to main( ) in Java