X.2 Objects – with Birgers Car

Description

Objects are the basic building blocks being used to create a model of a given system/application in an application domain. An object is an entity that exists inside a computer and we may use objects to represent information about phenomena in the application domain. Such information may be values describing properties of the phenomena.

Objects may also have behavior in the form of actions that may be executed. Such actions may compute new values from values stored in objects and save these values in objects.

Shall the example be vehicles or bank accounts?
A bank example is described in the note *OO-lec-v6-figures-ok* (in Papers in Dropbox). Part of the text from this note may be found here.
Other examples are: train reservation system from Beta Book, section 2.
We must also have an example with an active object.
We actually need several examples, but do we have more than one example here or do we have them in subsequent chapters?

Suppose we want to develop a system for keeping track of motor vehicles — a vehicle registration system. To do this we must create a model of vehicle registrations and other phenomena that a relevant for keeping track of vehicle registrations.

We may represent a vehicle registration by an object – the diagram below illustrates an object that may represent the registration of the car of Birger.

	BirgersCar
licenseNumber:	20116677
kerbWeight:	1500

An objects showing Birger's car

Note that the diagram is not the real object, but a picture of the object. The real object exists within a computer. We return to that later.

We should have have pictures showing Birger's real car and the object representing the car. Or later — see below?

How to generate an object

In order to generate objects in the computer, we must describe the objects in a language that can instruct the computer to generate the objects. As mentioned, such a language is called a programming language.

Below we show the *description* of an object representing the car of Birger.

```
BirgersCar: obj
   licenseNumber: val 20116677
   kerbWeight: val 1500
```

The keyword **`obj`** specifies that we describe an object.
The name of the object is `BirgersCar`.
The object has two attributes characterizing the car:

- Its license number represented by the data-item `licenseNumber` holding the integer value 20116677.
- Its kerb weight represented by the data-item `kerbWeight` holding the integer value 1500. Kerb weight is the weight of the car without any passengers.

The keyword **`val`** is a shorthand for value and specifies that the data-item holds a constant value. We return to this in Chapter XXX.

We must define the notion of object-descriptor (or perhaps just use the term block?) and explain that indentation is used to define a block. We must have a figure with lines that shows the block in the example.

As said above, the code/program above, is a description of an object — it is not the object itself as represented in the computer.

We may give this description to a compiler, which translates it into machine language. We may then execute the resulting machine language. which will generated the object in the computer.

Here we may have pictures showing Birger's real car and the object representing his car. These can be modified versions of slides 34-35 from the Onward presentation. The purpose of these pictures is to emphasize that we create a model.

Method attributes

`BirgersCar` has two data-items but it is also possible to associate so-called method-attributes with an object. A method describes a sequence of actions/statements that may change the state of the object and/or compute new values from the state of the object.

Et alternativ til addPassenger er weightFee, som beregnes basert på kerb weight. Her har du de satser som gælder i Norge fra 2023, og som du ser så er der forskel på elbiler og ikke-elbiler[OM1] ; det kunne f.eks. bruges til at illustrere virtuelle, selvom det jo egentlig bare er en værdi (kr/kg) som skiller de to typer:
Det ble for 2023 innført en egen vektavgift som skal gjelde for alle biler, også elbiler. Denne vektavgiften er på 12,50 kroner per kg på bilens vekt over 500 kg. Vektavgiften som bare gjelder for fossilbiler øker med vekten. For eksempel er vektavgiften 27,92 kroner per kilo på vekten mellom 500 og 1200 kg.

Eksempel taget fra en forhandlers hjemmeside:
Vektavgiften er 12,5 kr pr kg over 500 kg.
Om du for eksempel kjøper en EQC fra Mercedes-EQ (veier 2 495 kg) vil regnestykket seg slik ut:
Bilens totalvekt: 2 495 kg – 500 kg = 1 995 kg.
1 995 x 12,5 (vektavgift pr kilo) = 24 937,5
Der er også den såkaldte årsafgift som beregnes på grundlag af type bil og hvor mange dage den køres, dvs der er en sats pr dag. I Norge indkræves denne af forsikringsselskabene i forbindelse med bilforsikring.

In the example below, we add a data-item `payload` that represents the weight of possible passengers in the car.

In addition, we add a method called `addPassenger` that increase the value of `payload`.

```
BirgersCar: obj
   licenseNumber: val 20116677
   kerbWeight: val 1500
   weightFee: var kerbWeight * 12
   payload: var integer
   addPassenger:
      payload := payload + 80
```

The `payload` data-item is specified as a *variable* using the keyword **`var`** (shorthand for variable). A variable is a data-item that may hold different values during the execution of the program. It has an associated type that specifies the kind of values it may hold. For `payload`, the type is integer, which means that `payload` may (only) hold integer values.

The body of `addPassenger` has the form:

```
   payload := payload + 80
```

which is a so-called assignment statement. The operator `:=` specifies that the variable on the leftside gets a new value as specified by the right-side of `:=`. In this case the expression `payload + 80` computes the value of the `payload` plus `80` (We assume that the the average weight of each passenger is 80 kg.).

We should say something about units here — we assume that the unit is kg but a US person might think it is pound! A text on units may distrurb the flow of the main text. But it might be posssible to place boxes in the right column that plays the role of extended footnotes. And we could say something about the problem with units in most languages and possible solutions. And perhaps we later may have a chapter on units.

The `addPassenger`-method of `BirgersCar` may be executed by a statement of the form:

```
BirgersCar.addPassenger
```

In the next example we have added at method `removePassenger`, that as the name indicates, remove a passenger from the car. In addition, we have added a method `print`, that prints information about the payload of Birgers car on the computer screen.

```
BirgersCar: obj
   licenseNumber: val 20116677
   kerbWeight: val 1500
   payload: var integer
   addPassenger:
      payload := payload + 80
   removePassenger:
      payload := payload - 80
   print:
      print("The payload of Birgers car is " + payload + " kg")
```

The `removePassenger` method naturally subtracts 80 from the payload of the car. The `print` method prints the string `"The payload of Birgers car is "` plus the value of `payload` plus the string `" kg"` on the screen.

A string is a data-type that consists of a sequence of characters and double quotes (“) is used to specify a string literal. The data-type string is further described in Section xx.

The following code fragment shows examples of invoking the methods of `BirgersCar`:

Perhaps the code below should be placed in a object or method. Could be a `main` method as in most mainstream languages.

```
BirgersCar.addPassenger
BirgersCar.addPassenger
BirgersCar.removePassenger
BirgersCar.print
```

Execution of the above code results in the following text being displayed on the screen of the computer:

```
The payload of Birgers car is 80 kg
```

We return to methods in chapter XX.