Programming 2
Exercises 9

Kenny Štorgel

UP FAMNIT

Koper, April 2020

Classes and Objects

A general syntax to define a **class** is the following.

```
class class_name class_parameters =
      object
           instance variables
           more instance variables
           ...

           methods
           more methods
      end;;
```

Exercise: Define a class **basket** without parameters, instance variables and methods.

Instance variables and methods

Instance variables (fields) are declared by using keyword `val` and can be accessed in the class by using their names.

```
val value_name = expression
```

Methods are written by using the keyword `method`.

```
method method_name parameters = expression
```

Exercise: Add to the class **basket** two instance variables **apples** and **pears**, together with the two parameters that will be the initial number of apples and pears in the basket. Then write two methods **get** that will return the values of the two instance variables.

Changing values of instance variables

We can change the value of an instance variable by making it `mutable` (same as with types).

```
val mutable value_name = expression
```

```
value_name <- another_expression
```

Exercise: Correct declaration of instance variables so that we can change their values and write two methods **set** that can change the values of instance variables.

Creating objects and calling methods

We instantiate (create an instance) of the object by using the keyword `new`. Warning, if the type of a variable is unbound, then we get an error.

```
let object_name = new class_name class_parameters;;
```

We can call the methods with the use of (#) sign.

```
object_name#method_name method_parameters;;
```

Exercise: Create an object of type **basket** and try using some of its methods.

Calling methods in a method

By giving the object a name, we can instantiate a pseudo object of the class. The name is usually `self` for self explanatory reasons, but it can have any name.

```
class class_name class_parameters =
      object (self)

      end;;
```

Exercise: Write a method that will count the total number of apples and pears in a basket using the get methods.

Private methods

Private methods are available only to the objects in which they are defined. We define them by using the keyword `private`.

```
method private hidden_method = expression
method call_hidden_method = self#hidden_method
```

Exercise: Define a private method that will construct a string containing the information about the basket. Then define a method that will print that information using the command print_string.

Aggregation

Aggregation means that object is composed of other objects. We can call objects (classes) inside other objects (classes), below is one possible example of declaration.

```
class class_name class_parameters =
      object
           val object_name = (value : second_object_type)
      end;;
```

Exercise: Create a new object **warehouse** with a field **baskets** that is an empty list of object of type basket. Then write a method that returns the list of baskets and a method that takes the number of apples and the number of pears and adds a basket with that many apples and pears to the list baskets.

Initializers

Let-bindings within a class definition are evaluated before the object is created. We can also evaluate an expression right after the object has been created using an anonymous hidden method **initializer** which can access the object itself.

```
class class_name class_parameters =
let name parameters = expression in
      object
           initializer expression
      end;;
```

Exercise: In the class **basket** declare a new instance variable **basketSpace** which is computing before a basket is created and is equal to twice the number of apples and pears in the basket. Then use the initializer method to print "New basket with space X is created.".

Parametrized classes

Unlike before, where the compiler would attempt to find the most general type for our function, we have to declare that the class allows generic types.

Example:

```
class ['a,'b] pair (x0: 'a) (y0: 'b) =
      object
           val x = x0
           val y = y0
           method fst = x
           method snd = y
       end;;
```