Object-Oriented Design

OOP in Scala

2016

This practical introduces a systematic method that can be used to design systems with objects: CRC cards (Class, Responsibilities, Collaborators)[1]. Each card describes a class of objects, abstracting away its implementation. Each such description has three components:

- **Name of class:** This name creates a vocabulary between the conceptors of an application. It must thus be wisely chosen, to correctly describe the class purpose. It must be as indicative as possible, since it will be used in a much wider scope.
- **Responsibilities:** They identify the problems solved by this class. To define them, ask yourself the following questions: *what should my object **know**?* and *what should my object **do** in my application?*

  The first category of responsibilities (know things) encompasses mainly the values it should save while the second category of responsibility (do things) encompass the following:

  - do a given computation ;
  - modify its internal state in some way;
  - create and initialize other objects ;
  - control and coordinate the activity of other objects.
- **Collaborators:** The names of classes with which this class must cooperate to achieve its responsibilities. So they are the classes to which this one will send messages (call methods) or from which it will receive messages.

Here is an example CRC card, form a simple dice.

Dice

Responsabilities	Collaborators
– Save the value (a char) of each of the 6 sides – Save the currently visible side – Allow to retrieve the visible side – Allow to cast another dice (the visible side is updated)	– java.util.Random

★ **Exercice 1:** In this exercise, we design the classes of a simple LOGO interpreter.

The LOGO language was invented in the 60ies at the Massachusetts Institute of Technology (MIT) by Wally Feurzeig and Seymour Papert. This is a renowned tool to teach programming, thanks to the playfun aspects of the graphical turtle. This turtle can do the following actions: move forward by N pixels, turn right by N degrees, turn left by N degrees, move backward by N pixels, hide itself, show itself, move a pen down to leave a trail on the ground, move the pen up, and change the color of the pen.

The class `turtle.Screen` represents the graphical screen of our application. Its public interface is the following:

[1] K. Beck and W. Cunningham. A Laboratory for Teaching Object-Oriented Thinking, in Proceedings of OOPSLA'89. pp.˜1–6, 1989. ACM Press. http://doi.acm.org/10.1145/74877.74879

```
class Screen {
  def Screen(width:Int, height:Int)
  def setForegroundColor(c:Color) :Unit
  def drawLine(xA:Int, yA:Int, xB:Int, yB:Int) :Unit
  def fillRectangle(xA:Int, yA:Int, xB:Int, yB:Int) :Unit
  def setBackgroundColor(c:Color) :Unit
  def clear() :Unit
}
```

▷ **Question 1:** Propose a CRC card for the class `turtle.Screen`.

▷ **Question 2:** Model the turtle itself with a CRC card for the class `turtle.Tortoise`.

▷ **Question 3:** Propose an implementation interface of the `turtle.Tortoise` from its CRC card. Only specify the prototypes of the methods, not their content.

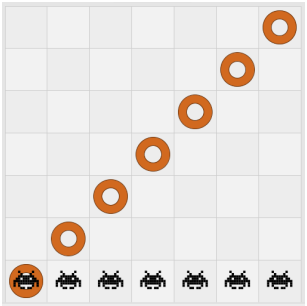
▷ **Question 4:** Propose a CRC card for the main class `turtle.Main`. This application works as follows: on startup, a graphical screen appears, and the user can input its command in a specific text area. When a command is entered, if it is valid, it is executed and the application asks for another command. The accepted commands are the following:

```
FD x    # move the turtle ForwarD by x pixels
BD x    # move the turtle BackwarD by x pixels
LT d    # Left Turn by d degrees
RT d    # Right Turn by d degrees
PENUP   # move the pen up. No trail is left on the screen afterward
PENDOWN # move the pen down. Moving then leaves a trail on the screen
CLEAR   # remove any trail from the screen
BC c    # pick a new color for the pen (0: white, 1:black, 2:blue, etc)
EXIT    # to quit the application
```

▷ **Question 5:** Propose a public interface for the class `turtle.Main`.

▷ **Question 6:** *(bonus)* Actually implement this little interpreter using the Java2D or ScalaFX interface below your `Turtle.Screen`.

★ **Exercice 2:** Specifying the buggles of the PLM



The buggles are little animals that live in a grid world. Just like turtles, they can move on this world (forward or backward by a cell, turn left or right by 90° at a time, move a brush up or down to leave a trail or not, change the color of the brush). Their world can contain baggles, these little biscuits that buggles love, but also walls that prevent the buggles from moving. The buggles must thus know whether they are facing the wall or not. They should also know when they are over a baggle, take it and leave it.

▷ **Question 1:** Propose the CRC cards of all classes needed for this application. The card for `turtle.Screen` created previously should be reused and eventually extended.

▷ **Question 2:** Propose the public interfaces of the CRC cards you proposed.