Languages and translator
Third assignment

2019-2020

1 Context

Web development has become a fundamental part of software development. There is a lot of libraries and most of the are written in the only really web-browser-supported language, Javascript. Many developers consider Javascript as a bad language, mainly because of its flawed typing system, that is weakly and dynamically typed, leading to strange behaviours (for some funny examples, you can look at this video `https://www.destroyallsoftware.com/talks/wat`).

However we see more and more language that compiles to Javascript and compilers that compile existing languages to Javascript (Ruby, Python, Java, Scala, etc.) **For this project, you will produce a DSL for a subset of the Canvas standard library from Javascript**.

Canvas Canvas are a way to display graphical information on a web page. For example, you can draw 2D graphics elements and create small games. See this tutorial `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial` to familiarize yourself with the Javascript API.

Scala.js Scala.js is a compiler for the Scala language that targets Javascript. It provides a full compatibility with existing libraries, while providing all the advantages of Scala, notably its typing system. Scala.js allows using the full Scala standard library, and a (mostly complete) part of the Java standard library, while being in the browser. As Javascript is weakly typed, the compiler cannot infer the type of variables in existing Javascript code; as such, multiple "typing layers" exists. Luckily, as Canvas is part of the "standard library" of Javascript inside browsers, the Scala.JS developers provide such a compatibility layer.

Here is an example of usage of the Canvas library inside Scala.JS:

```
package tutorial

import org.scalajs.dom
import dom.{document, html}

object WebApp {
  def main(args: Array[String]): Unit = {
    val canvas = document.createElement("canvas").asInstanceOf[html.Canvas]
    document.body.appendChild(canvas)

    val w = 300
    canvas.width = w
    canvas.height = w
    scalaJSDemo(canvas)
```

```
  }

  def scalaJSDemo(c: html.Canvas) = {
    val ctx = c.getContext("2d").asInstanceOf[dom.CanvasRenderingContext2D]
    val w = 300
    c.width = w
    c.height = w

    ctx.strokeStyle = "red"
    ctx.lineWidth = 3
    ctx.beginPath()
    ctx.moveTo(w/3, 0)
    ctx.lineTo(w/3, w/3)
    ctx.moveTo(w*2/3, 0)
    ctx.lineTo(w*2/3, w/3)
    ctx.moveTo(w, w/2)
    ctx.arc(w/2, w/2, w/2, 0, 3.14)

    ctx.stroke()
  }
}
```

As you can see, it is still quite verbose and untyped. What happens if we pass `"notacolor"` as an argument for the stroke style?

2 Goal of the project

Your goal is to create a new library that provides ways to interact with the canvas in a more elegant and typed way.

First Part The first part of the project does not require a lot of creativity. It is there to be sure that you can code in Scala.js and that you understand the fundamental concepts. The idea is to be able to add shapes to a canvas and modify them. If you pass the tests on INGInious for this part, you already get 1/3 of the points. Shapes are either normal shapes (rectangle, circle, hexagon, etc.) or a group of shapes (notice the beautiful recursion we have here!). At the end of this part, you must be able to compile this code and it should have the expected behavior.

```
def useMySuperDSL(canvas: html.Canvas): Unit = {
  // After you've done the first part of the project, everything should
  // compile and do the expected behaviour
  val canvasy = new Canvasy(canvas)

  val circles = Array.fill(4)(Circle(50, 0, 0))
  val rectangles = Array.tabulate(5)(i => Rectangle(i*10, i*10, 10, 30))

  canvasy += circles
  canvasy += rectangles

  // First we can modify property of Shapes by modifying their property directly
  circles(0) color "red"
  rectangles(0) strokeWidth 10
  rectangles(1) moveX 10
```

```
  // We should also be able to do the same on a group of shapes
  // (list, array, iterables, ...)
  circles moveX 20

  // We can also change property using the CanvasElementModifier trait
  circles change Color("blue")

  // We can group the shapes easily with the keyword and
  val superGroupOfShapes = circles and rectangles

  // And of course, we have foreach/map/flatmap available
  (rectangles(0) and circles(1)).foreach(_ moveY 30)

  // We should also be able to use common operators to group shapes
  val anotherSuperGroup = rectangles ++ circles

  // Take care that some property change should not compile, like this one
  // (rectangles(0) + circles(0)) change Width(30)
  // because Circles have no width

  // You can have a nice draw function to draw all of this on the canvas
  canvasy.draw()
}
```

Notice how we already use some of the best parts of Scala. But there is still room for improvement in the second part.

Second Part In this part, we ask you to enhance the DSL you made in the previous part such that you can code a small snake game. For a small example of a game coded in Javascript, you can look there `https://jsfiddle.net/codeandcircuit/6uoyvfs3/`. You are not forced to be able to execute the above code any more and you can (**should**) take a different direction for your DSL. Keep in mind that your DSL should **provide a way to code a snake in a good way** and not an API to a snake game. Among other things, it should be easy to register events (on the shapes, between the shapes, on user-input, etc.), update the snake using the usual operators, execute a piece of code every X milliseconds (or seconds, minutes, etc.).

Finally, **use most of the features of Scala**, including

- Strong and static typing
- Type inference
- By-name parameters
- Implicits
- Currying
- Monads
- Closures
- ....

Once you have a basic snake game, keep improving your DSL to allows more possibility to the snake game. You can also demonstrate the use of your DSL on other game that you might think of (e.g. space invaders), be creative!

3 Evaluation and deliverables

There will be automated test on INGInious for the first part of the assignment.

Building a good DSL is partially an art. There is therefore no unique result that is expected for your work. However, there are objective criteria that still can be used to evaluate it. Here is a non-exhaustive list of criteria that will be used to evaluate your project:

- Are you using the concepts presented during the course (see above) in an appropriate way?
- How easy, elegant and flexible is your DSL? Have you added new abstractions that were not initially provided by the library?
- Are you hiding/removing functionalities from the library? Your DSL should not reduce the scope of problems that could be solved using the library but only provide users with an easier way to achieve specific goals
- Is your code clean, well indented and documented? Are the principle of functional programming used when possible (e.g. prefer immutable over mutable state when performances are not a bottleneck)?

Remember: we do not grade the capacities of your library, but how you implemented them and how user access them with your DSL. Talk about your code, and how to use it, and which advanced feature of Scala it uses.

For the deliverables, we ask you **to submit on INGInious**:

- The source code of your project, in a zip file
- A 4-page report, based on this skeleton (you can change it):
  - Functionalities/specificities of your DSL (some documented examples)
  - Strong/weak points of your DSL
  - What could have been improved with more time
- **The deadline for the project is Fiday the 15-th of May 18:00 (S13)**

4 How to start & resources

For a good start, you can

- Clone the skeleton of the project from this repo `https://github.com/AlexandreDubray/lt-project` and verify that it works correctly (**read the instructions in the README**). **WARNING: the skeleton of the project is on Github, but your code can not be openly accessible and you can not share it between students.**
- Install a good scala IDE (IntelliJ is a good choice)
- Follow the tutorial on the Scala.js website `https://www.scala-js.org/doc/tutorial/basic/` and on the Mozilla site (for the canvas) `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial`

If you need information about the DOM, there is documentations here `https://developer.mozilla.org/en-US/docs/Web`