Various evaluation strategies for the lambda calculus have been defined in the course using the concepts of substitution and evaluation context. These definitions are very well suited for reasoning but say nothing on how these strategies can be effectively implemented. The goal of this series is to give more computational definitions for the call-by-value and call-by-name strategies and to complete the file `lambdaCalculus-partial.scala` (accessible from the course page) that implements them.

Call-by-value

The idea is to use the concepts of *environment* and *closure* instead of those of evaluation context and substitution. Informally, an environment is a finite map from variables to values and a closure is a function together with an environment that defines values for the free variables of the function. A value is the result of computing a lambda term, it can be a constant or a closure.

Terms			
s, t, u	$::=$	c	Constant
	$\mid$	x	Variable
	$\mid$	$\lambda x.t$	Abstraction
	$\mid$	$t\ u$	Application
Values			
v, w	$::=$	c	Constant
	$\mid$	$\langle \lambda x.t, E \rangle$	Closure
Environment			
E	$::=$	ϵ	Empty
	$\mid$	$E, x \mapsto v$	Binding

Now we define, using inference rules, the evaluation of a lambda term t to a value v in a environment E, that we write $E \vdash t \Rightarrow v$.

- A constant is already evaluated:

$$\textsc{Const}\ \frac{}{E \vdash c \Rightarrow c}$$

- A variable evaluates to the value it is bound to in the environment:

$$\textsc{Var}\ \frac{}{E \vdash x \Rightarrow E(x)}$$

- An abstraction evaluates to a closure that stores the current environment:

$$\textsc{Abstr}\ \frac{}{E \vdash \lambda x.t \Rightarrow \langle \lambda x.t, E \rangle}$$

- To evaluate an application $(t\ u)$, evaluate first the function part t to a closure, then evaluate the argument part u and finally evaluates the body t' of the function in its environment E' extended with a new binding corresponding to the argument passing:

$$\textsc{App}\ \frac{E \vdash t \Rightarrow \langle \lambda x.t', E' \rangle \qquad E \vdash u \Rightarrow v \qquad E', x \mapsto v \vdash t' \Rightarrow w}{E \vdash t\ u \Rightarrow w}$$

Adapt this formalization to the call-by-name lambda calculus and implement both strategies in Scala.