Paradigmas de la Programación – Examen Final

7 de Julio de 2017

Apellido y Nombre: _____

1. **[10 pt.]** La siguiente expresión está mal tipada:

   ```
   f(g,x) = g(x) + x && g(x)
   ```

   Muestre con el árbol para inferencia de tipos dónde se encuentra el conflicto y en qué consiste. Cómo podría resolver este conflicto un lenguaje de tipado fuerte? y uno de tipado débil?

2. **[20 pt.]** Diagrame los sucesivos estados por los que pasa la pila de ejecución al ejecutar los siguientes tres programas en ML:

```
exception Excpt of int;
fun twice(f,x) = f(f(x)) handle Excpt(x) => x;
fun pred(x) = if x = 0 then raise Excpt(x) else x-1;
fun dumb(x) = raise Excpt(x);
fun smart(x) = 1 + pred(x) handle Excpt(x) => 1;
twice(pred,1);
```

```
exception Excpt of int;
fun twice(f,x) = f(f(x)) handle Excpt(x) => x;
fun pred(x) = if x = 0 then raise Excpt(x) else x-1;
fun dumb(x) = raise Excpt(x);
fun smart(x) = 1 + pred(x) handle Excpt(x) => 1;
twice(dumb,1);
```

```
exception Excpt of int;
fun twice(f,x) = f(f(x)) handle Excpt(x) => x;
fun pred(x) = if x = 0 then raise Excpt(x) else x-1;
fun dumb(x) = raise Excpt(x);
fun smart(x) = 1 + pred(x) handle Excpt(x) => 1;
twice(smart,1);
```

3. **[10 pt.]** Si el resultado del siguiente programa es 19, qué tipo de alcance tiene el lenguaje de programación en el que está escrito, estático o dinámico?

```
val x = 4;
    fun f(y) = x*y;
    fun g(x) = let
        f(3) + x;
    g(7);
```

4. **[10 pt.]** Para solucionar los *name clashes*, algunos lenguajes implementan heurísticas para decidir qué implementación seleccionar en el caso de un *name clash*. Explique qué estrategia utiliza Java para evitar este tipo de problema y describa por lo menos otra estrategia posible.

5. **[10 pt.]** El siguiente es un ejemplo de un operador que busca aumentar la seguridad de un lenguaje. Explique qué aporta este operador en términos de seguridad, y qué otros mecanismos se pueden implementar en lenguajes inseguros para aumentar su seguridad.

   > Kotlin makes a distinction between nullable and non-nullable datatypes. All nullable objects must be declared with a ¿"postfix after the type name. Operations on nullable objects need special care from developers: null-check must be performed before using the value. Kotlin provides null-safe operators to help developers:

   ?. (safe navigation operator) can be used to safely access a method or property of a possibly null object. If the object is null, the method will not be called and the expression evaluates to null.

   ?: (null coalescing operator) often referred to as the Elvis operator:

   ```
   fun sayHello(maybe : String?, neverNull : Int) {
      // use of elvis operator
      val name : String = maybe ?: "stranger"
      println("Hello $name")
   }
   ```

   Un ejemplo de uso:

   ```
   // returns null if (and only if) foo is null, or bar() returns null,
   // or baz() returns null
   foo ?. bar() ?. baz()
   ```

6. **[10 pt.]** En el siguiente código en React, tenemos un ejemplo de programación declarativa? Relacione este código y el framework React en general con el paradigma de actores y con otros frameworks.

```
import React, { PropTypes } from 'react'

const Todo = ({ onClick, completed, text }) => (
  <li
    onClick={onClick}
    style={{
      textDecoration: completed ? 'line-through' : 'none'
    }}
  >
    {text}
  </li>
)

Todo.propTypes = {
  onClick: PropTypes.func.isRequired,
  completed: PropTypes.bool.isRequired,
  text: PropTypes.string.isRequired
}

export default Todo
```

7. **[10 pt.]** Cuando usamos frameworks se da una inversión de control. Explique brevemente en qué consiste la inversión de control y argumente cuál de los dos fragmentos de código que siguen es un ejemplo de inversión de control.

```
class PaymentsController < ApplicationController
  def accept_payment
    if Rails.env.development? || Rails.env.test?
```

```
      @credit_card_validator = BogusCardValidator.new
    else
      @credit_card_validator = RealCardValidator.new
    end
    if Rails.env.production?
      @gateway = RealPaymentGateway.new
    elsif Rails.env.staging?
      @gateway = RealPaymentGateway.new(use_testing_url: true)
    else
      @gateway = BogusPaymentGateway.new
    end
    card = @credit_card_validator.validate(params[:card])
    @gateway.process(card)
  end
end
```

```
class PaymentsController < ApplicationController
  def accept_payment
    card = @credit_card_validator.validate(params[:card])
    @gateway.process(card)
  end
end

# config/dependencies/production.rb:
RailsENV::Dependencies.define do
  prototype :payment_gateway,        RealPaymentGateway
  prototype :credit_card_validator, RealCardValidator
  controller PaymentsController, {
    gateway:                ref(:payment_gateway)
    credit_card_validator: ref(:credit_card_validator)
  }
end

# config/dependencies/staging.rb:
RailsENV::Dependencies.define do
  inherit_environment(:production)
  prototype :payment_gateway, RealPaymentGateway, {use_testing_url: true}
end

# config/dependencies/development.rb:
RailsENV::Dependencies.define do
  inherit_environment(:production)
  singleton :payment_gateway, BogusPaymentGateway
  singleton :credit_card_validator, BogusCardValidator
end
```

8. **[10 pt.]** Elm es un lenguaje de programación que pretende sustituir Javascript. Es un lenguaje de programación declarativo, sin embargo pretende usarse en los mismos contextos que un lenguaje de programación de scripting tipo glue. La propuesta es circunscribir las partes no declarativas de los programas fuera de las componentes puramente declarativas, construyendo programas dentro de la llamada The Elm Architecture (TEA).

   Comente el siguiente texto, comparando la funcionalidad de la TEA con la funcionalidad de las mónadas en otros lenguajes funcionales (como Haskell), explique las propiedades de las componentes decla-

rativas y cómo estas propiedades producen limitaciones en los lenguajes de programación tipo glue, y cómo estas limitaciones se pueden solventar mediante la TEA o las mónadas.

> *The simplest program consists of a model record storing all data that might be updated, a union type Msg that defines ways your program updates that data, a function update which takes the model and a Msg and returns a new model, and a function view which takes a model and returns the HTML your page will display. Anytime a function returns a Msg, The Elm Architecture uses it to update the page.*

9. **[10 pt.]** Diga cuál sería el resultado de la siguiente función con alcance estático y con alcance dinámico, y describa brevemente cómo se obtiene el resultado en cada uno de los casos.

```
var x=1;
function g(z) {return x+z;}
function f(y) {
  x = y+1;
  return g(y*x);
}
f(3)
```

Ejercicios para libres

1. **[-5 pt.]** El siguiente es un ejemplo de "*spaghetti code*". Reescríbalo en pseudocódigo de forma que NO use saltos (GOTO), y en cambio use programación estructurada en bloques.

```
10 i = 0
20 i = i + 1
30 PRINT i; " squared = "; i * i
40 IF i >= 10 THEN GOTO 60
50 GOTO 20
60 PRINT "Program Completed."
70 END
```

2. **[-10 pt.]** El siguiente código está dentro del modelo de actores. Señale en el código los mecanismos propios de actores y sírvase de ellos para explicar cómo los actores implementan concurrencia declarativa.

```
import akka.actor._

case object PingMessage
case object PongMessage
case object StartMessage
case object StopMessage

class Ping(pong: ActorRef) extends Actor {
  var count = 0
  def incrementAndPrint { count += 1; println("ping") }
  def receive = {
    case StartMessage =>
        incrementAndPrint
        pong ! PingMessage
    case PongMessage =>
        incrementAndPrint
        if (count > 99) {
          sender ! StopMessage
          println("ping stopped")
```

```
            context.stop(self)
          } else {
            sender ! PingMessage
          }
    }
}

class Pong extends Actor {
    def receive = {
      case PingMessage =>
          println("  pong")
          sender ! PongMessage
      case StopMessage =>
          println("pong stopped")
          context.stop(self)
    }
}

object PingPongTest extends App {
    val system = ActorSystem("PingPongSystem")
    val pong = system.actorOf(Props[Pong], name = "pong")
    val ping = system.actorOf(Props(new Ping(pong)), name = "ping")
    // start them going
    ping ! StartMessage
}
```

3. **[-10 pt.]** Dada la siguiente base de conocimiento en Prolog, grafique o explique verbalmente cuál sería la secuencia de predicados por los que se realizaría la búsqueda para verificar si es cierto `recomendar(pedro,clara)`.

```
amigo(pedro,maria).       amigo(pedro,juan).
amigo(maria,pedro).       amigo(maria,clara).
amigo(maria,romina).      amigo(juan,pedro).
amigo(juan,clara).        amigo(clara,maria).
amigo(clara,juan).        amigo(romina,maria).

recomendar(X,NuevoAmigo) :-
  amigo(X,Amigo), amigo(Amigo,NuevoAmigo),
  not(amigo(X,NuevoAmigo)), not(X = NuevoAmigo).
```