Universidad Simón Bolívar.
Departamento de Computación.
Lenguajes de programación.
Angel Garces 16-10400

# Examen 3

En este documento se responderán algunas de las preguntas del examen 3 de lenguajes de programación, los códigos se pueden encontrar en el siguiente link de github:

`https://github.com/AngelVzla99/materia-lenguajes-de-programacion.git`

1. Pregunta 1
   Para esta pregunta decidí usar Scala

   *a*) En Scala para crear clases se usa la palabra reservada class seguida del nombre de la clase, para crear métodos simplemente se definen funciones dentro de la clase y para tener campos en una clase estos deben estar en el constructor principal (el cual se define poniendo parámetros al lado del nombre de la clase). Acá podemos ver un ejemplo:

```
class A( var a:Int, var b:Int ){
    // Definicion de un metodo
    def method( c:Int ) : Int ={
        return a + b + c;
    }
    // Sobrecarga del constructor
    def this( x:Int ) = this( x,0 );
}
```

   También debemos apreciar que en Scala a diferencia de otros lenguajes todos los campos de la clase deben ser inicializados y la sobrecarga de constructores es valida siempre cumplan esta propiedad.

   Para crear instancias de una clase se puede hacer de la siguiente manera:
   val obj1 :A = new A(1,2)

   Ya que Scala corre en la JVM el manejo de memoria se realiza de manera implícita ya que el recolector de basura de la maquina virtual se encarga de liberar la memoria de objetos no referenciados .

   La asociación de métodos es dinámica y se puede alterar colocando final delante de un método, de esta manera todos las subclases de la clase con el método final no podrán sobre escribirlo. Aca se puede ver un ejemplo de un codigo que imprime -1

```
class A(){
    def method() : Int ={
        return 1;
    }
}

class B extends A{
    override def method() : Int = {
        return -1;
    }
}

object MakeAClass{
    def main(args: Array[String]){
        val obj2 :A = new B();
        println( obj2.method() );
    }
}
```

Para tener herencia se coloca la palabra reservada **extends** al lado del nombre de la clase seguido de la clase de la que hereda (no existe herencia múltiple). Scala posee polimorfismo paramétrico. El lenguaje ofrece 3 tipos de varianza:

1) class Foo[+A] ( Es una clase covariante )
2) class Bar[-A] ( Es una clase contravariante )
3) class Baz[A] ( Es una clase invariante )

Ejemplo de polimorfismo parametrico:

```
def fpoly[A <: {def apply(i: Int): Any }](x: A, i: Int): Any = {
    x(i)
}
```

En este caso el return de la función es genérico dependiendo de la función que pasemos como parámetro.

*b*) Ambas implementaciones se encuentran en el repositorio del examen, se pueden encontrar en la carpeta **examen3/pregunta1** los nombres son **parteA.Scala y parteB.Scala**. Para correr los codigos primero se usa el comando **scalac parteA.Scala** seguido de **scala ParteA** (análogamente con la parte B).

2. Pregunta 2
Para esta pregunta decidí usar el lenguaje de programación Scala (ya que tiene 5 letras)

*a*) Scala no posee librerías para el manejo de hilos sin embargo de manera nativa permite a los programadores tener programación concurrente. Este lenguaje ofrece dos manera de crear hilos, una es de la forma clásica que hace Java y otra es

usando Future. Veamos la primera.

Tal como en Java, hay 2 maneras de crear hilos, en ambos casos es necesario crear una clase que debe heredar de la clase Threads o de la interface Runnable, y debe definir el método run() el cual sera llamado para crear un hilo de ejecución.

Para crear un hilo se crea una instancia de la clase creada y se inicia el hilo con instancia.start() y para esperar a que este hilo termine se usa instancia.join() (también de manera nativa el lenguaje provee funciones para saber el estado actual del hilo en el SO).

En Scala tenemos una segunda alternativa para crear hilos que es con Future usando la siguiente semántica:

```
val a = Future { Thread.sleep(10*1000); 42 }
a: scala.concurrent.Future[Int] = Future(<not completed>)
```

El objetivo de crear hilos de esta manera es crear tareas en simultaneo suponiendo que en algun momento ese hilo retornara un resultado que podremos usar para seguir calculando cosas (en el caso del código anterior, esperamos que el hilo nos retorne en 42).

Para lograr la sincronización en Scala se hace usando la siguiente estructura

```
synchronized {
// we can write our logic here.
}
```

Como se puede apreciar las secciones criticas se encierran en un bloque que luego el lenguaje entiende que debe ser ejecutado por un solo hilo a la vez. También para hilos podemos usar **java.util.concurrent.BlockingQueue** para enviar mensajes de manera concurrente entre hilos.

*b*) Para esta pregunta guarde los archivos utilizados en la carpeta **examen3/pregunta2** del repositorio

Para ejecutar los archivos es necesario tener intalado el compilador de scala. Para correr el primero programa se crea el ejecutable con **scalac addMatrices.Scala** y se ejecuta el programa con **scala addMatrices**. Para correr el segundo programa se crea el ejecutable con **scalac archivos.Scala** y se ejecuta el programa con **scala Archivos**

3. Pregunta 3
En esta pregunta decidí hacerla en las presentaciones de google, puede acceder a ella con el siguiente enlace:
`https://docs.google.com/presentation/d/1IQoU8lGW-rd1wsMr2p_KprhdAIWkSFQ2LkUM_Rz-gJ8/edit?usp=sharing`

4. Pregunta 4
   Decidí hacer la implementación de esta pregunta en c++, el código se puede encontrar en el repositorio de github **examen3/pregunta4** para ejecutar el programa se crea el ejecutable con el comando **make** y luego **./main**.

   Para correr los test cases se corre el comando **make unit_test** seguido de **./unit_test** (para correr el test unitario se usan los mismos comando agregando _coverage al final). Con estos test se obtuvo un cubrimiento del 100 % del archivo **handler.cpp**

5. Pregunta 5

   *a*) 1) En un lenguaje con evaluación normal el programa no terminaría, la primera expresión que tenemos es:

      misteriosa abc (gen 1)

      = <El primer argumento no se puede simplificar mas, pero el segundo necesita ser calculado>

      (gen 1) por definición es (gen 1) = 1:(gen 2)
      (gen 2) por definición es (gen 2) = 2:(gen 3)
      (gen 3) por definición es (gen 3) = 3:(gen 4)
      ...
      Continúa de esta manera haciendo infinitas operaciones

      2) La corrida paso a paso se encuentra el repositorio ( en **examen3/pregunta5/src/Partea.hs** )

   *b*) Mi implementación de esta pregunta es la siguiente (también se puede encontrar en el repositorio **examen3/pregunta5/src/Parteb**)

```
foldA :: (a -> b -> b -> b) -> b -> Arbol a -> b
foldA _ b Hoja = b
foldA f b (Rama a nodeL nodeR) =
    f a (foldA f b nodeL) (foldA f b nodeR)
```

   *c*) La corrida paso a paso se encuentra el repositorio ( en **examen3/pregunta5/src/Partec.hs** )

6. Pregunta 6
   Bueno, la 6 no me dio tiempo de hacerla jeje ... pero aprovecharle este espacio en blanco para darle las gracias por la materia, quedo muy chevere :D