UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación avanzada II
Examen final Primera convocatoria ordinaria
Curso 2024/25

PARTE 1: Programación funcional

4 puntos **Ejercicio 1.1**

Supongamos el siguiente trait con las operaciones básicas de grafos dirigidos.

```
trait Graph[N] {
  def addNode(node: N): Graph [N] // añade un nodo al grafo
  def addEdge(from: N, to: N): Graph [N] // añade una arista del nodo from al nodo to, lanza una
        // IllegalArgumentException si alguno de los nodos no existe en el grafo.
  def neighbors(node: N): Set[N] // devuelve los nodos vecinos de un nodo dado (consideramos vecinos a
        // los extremos de aristas con el nodo node como origen)
  def hasNode(node: N): Boolean // comprueba si el grafo contiene un nodo
}
```

Proporciona una clase `GraphOverMap[T]` que implemente `Graph[T]`. La clase debe usar una `Map[N, Set[N]]` para almacenar los nodos y aristas de un grafo. Si un grafo tiene una arista `a -> b`, el map correspondiente tendrá `a` como clave y `b` estará en el conjunto de elementos asociados a `a`. Además de los métodos del trait, la clase `GraphOverMap[T]` proporcionará:

- Un constructor principal privado que tome un argumento de tipo `Map[N, Set[N]]`.
- Un constructor sin argumentos (`this()`).
- Redefiniciones de los métodos `toString` (mostrando el conjunto con el formato `"Graph(1 -> 2, 2 -> 3, 3 -> 1, 3 -> 2)"`), `hashCode` y `equals` para asegurar que dos maps con los mismos nodos y aristas se consideren iguales.

2 puntos **Ejercicio 1.2**

Utilizando match y funciones de orden superior, define una función `combinations` que reciba un número k y una lista, y devuelva todas las combinaciones de tamaño k de los elementos de la lista. Por ejemplo, combinations(2, List(1, 2, 3, 4)) es List(List(1, 2), List(1, 3), List(1, 4), List(2, 3), List(2, 4), List(3, 4))
combinations(3, List(1, 2, 3, 4)) es List(List(1, 2, 3), List(1, 2, 4), List(1, 3, 4), List(2, 3, 4))

2 puntos **Ejercicio 1.3**

Dada la clase

```
class Person(name: String, age: Int) {
  def getAge = age
  def getName = name
  override def toString: String = s"Person($name, $age)"
}
```

y un valor

```
val people = List(Person("Alice", 30), ..., Person("Oscar", 36))
```

- Crea un map en el que las claves son las iniciales de los nombres y los valores son conjuntos de los nombres de las personas con esa inicial.
- Crea un map en el que a cada persona se le asocia su edad en meses.
- Calcula la edad media de las personas.

2 puntos **Ejercicio 1.4**

Sin utilizar la función `partition`, implementa una función `partitionBy` que particione una lista en dos listas según un predicado, es decir, coloque los elementos que cumplen el predicado en una lista y los que no lo cumplen en otra. Por ejemplo:

```
partitionBy(List(1, 2, 3, 4, 5, 6))(_ % 2 == 0) // Output: (List(2, 4, 6), List(1, 3, 5))
partitionBy(List("apple","banana","cherry","date"))(_.length > 5) // Output: (List(banana,cherry), List(apple,date))
```

Funciones del API List	Explicación
list.splitAt(n:Int)	Divide list en dos por la posición n y devuelve las dos sublistas (List(0..n-1),List(n..list.size-1)
list.filter(p)	Devuelve la sublista de list con los elementos que satisfacen p
list.map(f)	Devuelve la lista obtenida aplicando f a cada elemento de list
list.foldLeft(z)(f) list.foldRight(z)(f)	Aplica la operación binaria f a los elementos de list de izquierda a derecha (o de derecha a izquierda) empezando por z
list.sum	Suma los elementos de list
list1.zip(list2)	Devuelve la lista de pares construida con elementos de list1 y list2
list.groupBy(f)	Construye un mapa a partir de los elementos de list utilizando la función f

Nota: Si utilizas REPL, recuerda que para cargar las funciones de un fichero en REPL se utiliza el comando ***:load <path>*** donde ***<path>*** es la ruta al fichero.

Programación concurrente

El Juego de las Sillas

Para jugar al juego de las sillas colocamos varias sillas formando un círculo, con el asiento hacia el exterior. Si tenemos J jugadores, tendremos J - 1 sillas. Por ejemplo, si hay 5 jugadores debe haber 4 sillas. Inicialmente, los jugadores forman un círculo alrededor de las sillas, y a continuación se pone la música. En ese momento los participantes empiezan a dar vueltas alrededor de las sillas. Cuando se apaga la música los jugadores deben sentarse en una de las sillas. Al faltar una silla, uno de los jugadores quedará en pie, quedando eliminado. A continuación, se quita una silla y se repite el juego, que continúa ronda tras ronda del mismo modo hasta que queda solo una silla y dos jugadores. El jugador que consigue sentarse en esa última ronda gana el juego. Por ejemplo, si inicialmente hay 4 sillas y 5 jugadores, las partidas tienen las siguientes configuraciones:

Partida 1: 4 sillas-5 estudiantes
Partida 2: 3 sillas-4 estudiantes
Partida 3: 2 sillas-3 estudiantes
Partida 4: 1 silla-2 estudiantes. El estudiante que se sienta gana.

PARTE 2: Sincronización con semáforos

7 puntos

Ejercicio 2.1. Implementa el juego de las sillas utilizando **semáforos binarios**. Para hacerlo se debe utilizar el esqueleto proporcionado en el CV que contiene las siguientes clases y funciones:

- La clase `SillasMusicales` tiene como parámetro el número de sillas iniciales del juego (las de la primera partida) y proporciona las siguientes funciones:
  - `def meSiento(id:Int, p:Int): Boolean`. El jugador `id` la utiliza para intentar sentarse en una de las sillas en la partida `p` del juego. La función devuelve `true` si el jugador consigue sentarse y `false`, en otro caso.
  - `def esperoFelicitaciones(): Unit` y `def felicito(id:Int)` se piden en el ejercicio 2.2.
  - La función `main` que crea el objeto `sillasMusicales` y las hebras `jugador` **cuyo comportamiento ya está implementado**. Cada hebra llama a la función `meSiento` en las sucesivas partidas hasta que o bien pierde (la función devuelve `false`) o bien gana el juego (`meSiento` devuelve `true` en la última partida).

En la función `meSiento` deben implementarse las siguientes condiciones de sincronización:

CS-Jugador-1: Un jugador que logra sentarse en la partida `p` debe esperar sentado hasta que todos los demás jugadores intenten sentarse también en la misma partida (uno de ellos no lo logra).

CS-Jugador-2: Un jugador no debe intentar sentarse en la partida `p+1` si aún no se han levantado de las sillas los jugadores de la partida anterior `p`.

Notas

1. El jugador que pierde la partida `p` es el que avisa al resto de que se ha terminado.
2. En la partida `p+1` hay un jugador y una silla menos que en la partida `p`.

3 puntos **Ejercicio 2.2 (3 puntos):** Extiende el sistema de la primera parte para que, al final del juego, los jugadores que han perdido feliciten al ganador. Utiliza solo **semáforos binarios** para implementar el comportamiento pedido.

- Implementa la función `def felicito(id:Int)` de la clase `SillasMusicales`. Un jugador que ha perdido llama a esta función para felicitar al ganador.
- Implementa la función `def esperoFelicitaciones(): Unit` de la clase `SillasMusicales`. El jugador que ha ganado llama repetidamente a esta función para aceptar las felicitaciones de cada uno los otros jugadores y dar las gracias.

Las condiciones de sincronización que hay que implementar son:

CS-Jugador-3: Un jugador que ha perdido espera a que el ganador esté disponible para felicitarle.

CS-Ganador-1: El ganador espera a que lo felicite un perdedor para darle las gracias, el ganador tiene que repetir este proceso hasta que todos los perdedores lo hayan felicitado.

Descomenta en la función `main` los códigos en el que los jugadores que han perdido llaman a la función `felicito` y el ganador llama sucesivamente a `esperoFelicitaciones` para aceptarlas.

Una posible ejecución del juego de las sillas con 6 jugadores y 5 sillas se muestra a continuación. Las últimas líneas con las felicitaciones corresponden a la segunda parte del ejercicio.

```
Thread-4:Iteración 0: Jugador 4 se ha sentado!! :-))
Thread-2:Iteración 0: Jugador 2 se ha sentado!! :-))
Thread-5:Iteración 0: Jugador 5 se ha sentado!! :-))
Thread-3:Iteración 0: Jugador 3 se ha sentado!! :-))
Thread-1:Iteración 0: Jugador 1 se ha sentado!! :-))
Thread-0:Iteración 0: Jugador 0 ha perdido :-((. Sale del juego
Thread-0:Jugador 0 sale de la sala. 5
Thread-1:Jugador 1 sale de la sala. 4
Thread-3:Jugador 3 sale de la sala. 3
Thread-5:Jugador 5 sale de la sala. 2
Thread-2:Jugador 2 sale de la sala. 1
Thread-4:Jugador 4 sale de la sala. 0
Thread-4:----------Fin iteración 0----------
Thread-2:Iteración 1: Jugador 2 se ha sentado!! :-))
Thread-4:Iteración 1: Jugador 4 se ha sentado!! :-))
Thread-5:Iteración 1: Jugador 5 se ha sentado!! :-))
Thread-1:Iteración 1: Jugador 1 se ha sentado!! :-))
Thread-3:Iteración 1: Jugador 3 ha perdido :-((. Sale del juego
Thread-3:Jugador 3 sale de la sala. 4
Thread-1:Jugador 1 sale de la sala. 3
Thread-5:Jugador 5 sale de la sala. 2
Thread-4:Jugador 4 sale de la sala. 1
Thread-2:Jugador 2 sale de la sala. 0
Thread-2:----------Fin iteración 1----------
Thread-2:Iteración 2: Jugador 2 se ha sentado!! :-))
Thread-1:Iteración 2: Jugador 1 se ha sentado!! :-))
Thread-5:Iteración 2: Jugador 5 se ha sentado!! :-))
Thread-4:Iteración 2: Jugador 4 ha perdido :-((. Sale del juego
Thread-4:Jugador 4 sale de la sala. 3
Thread-5:Jugador 5 sale de la sala. 2
Thread-1:Jugador 1 sale de la sala. 1
Thread-2:Jugador 2 sale de la sala. 0
Thread-2:----------Fin iteración 2----------
Thread-1:Iteración 3: Jugador 1 se ha sentado!! :-))
Thread-2:Iteración 3: Jugador 2 se ha sentado!! :-))
Thread-5:Iteración 3: Jugador 5 ha perdido :-((. Sale del juego
Thread-5:Jugador 5 sale de la sala. 2
Thread-2:Jugador 2 sale de la sala. 1
Thread-1:Jugador 1 sale de la sala. 0
Thread-1:----------Fin iteración 3----------
Thread-2:Iteración 4: Jugador 2 se ha sentado!! :-))
Thread-1:Iteración 4: Jugador 1 ha perdido :-((. Sale del juego
Thread-1:Jugador 1 sale de la sala. 1
Thread-2:Jugador 2 sale de la sala. 0
Thread-2:----------Fin iteración 4----------
Thread-2:Jugador 2 es el ganador!!!! Espero las felicitaciones de mis compañeros
Thread-1:                    Felicidades!!!!
Thread-2:                Gracias.....
```

```
Thread-5:          Felicidades!!!!
Thread-2:       Gracias.....
Thread-4:          Felicidades!!!!
Thread-2:       Gracias.....
Thread-3:          Felicidades!!!!
Thread-2:       Gracias.....
Thread-0:          Felicidades!!!!
Thread-2:       Gracias.....
```

NOTA IMPORTANTE: Debes recordar que una salida por pantalla similar a la anterior no implica que el código implementado lo sea. Para aumentar la variabilidad de la ejecución puedes ejecutar varias veces tu solución, con distinto número de jugadores o modificando o quitando los sleeps de las hebras.

PARTE 3: Sincronización con monitores

Ejercicio 3: Implementa las dos partes del juego de las sillas descritas en la parte 2 utilizando métodos sincronizados o locks. La valoración de cada parte es la misma que la de los ejercicios 2.1 y 2.2.