UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación avanzada II
Primer Control Grupo IS-A
Curso 2024/25

En este examen crearemos un proyecto en **C:\Usuario\Alumno\Documentos** con nombre PAII_GIS_A. Se desarrollarán todas las clases y funciones solicitadas en una misma clase Scala (control1.scala), que entregaremos usando la tarea creada para tal fin en el Campus Virtual.

En algunos ejercicios tienen sugerencias sobre la forma en que deben ser implementados. Sólo las soluciones que sigan estas indicaciones tendrán la valoración máxima.

Ejercicio 1 (3 puntos)

Supongamos el siguiente `trait` con las operaciones básicas sobre vectores

```
trait ImmutableVector{
  def toList:List[Double] //devuelve una lista con los elementos del vector
def dim:Int        // devuelve la dimensión del vector   def
+(v:ImmutableVector):ImmutableVector //suma el vector this y v   def
*(v:ImmutableVector):Double //calcula el producto escalar de this y v
}
```

Proporciona una clase `MiVector` que implemente `ImmutableVector`. La clase debe usar una `List[Double]` para almacenar los elementos del vector. Además de los métodos del `trait`, la clase `MiVector` proporcionará:

La clase debe usar una `List[Double]` para almacenar los elementos del vector. La clase tiene un constructor principal **privado** que toma como argumento la lista de tipo `List[Double] con los` elementos del vector**.**
Además debes implementar las siguientes funciones:

- Un constructor **privado** que tome un argumento de tipo List[Double]
- Un constructor **público** que toma una secuencia de valores Double (`this(Double*)`)
- Un método `+(v: MiVector): MiVector` que devuelve la suma del vector actual y el vector que recibe como argumento. La suma solo puede realizarse si ambos vectores tienen la misma dimensión. Por ejemplo (1.0,3.0,4.0)+(2.0,1,0,0.0) = (3.0,4.0,4.0)

  Sugerencia: utilizar alguna función de la API de la clase `List` (como `filter`, `map`, `zip`, `unzip`, etc.) sin recursión ni pattern matching

- Un método `*(v: MiVector): MiVector` que devuelve el productor escalar del vector actual y el vector que recibe como argumento. El producto solo puede realizarse si ambos vectores tienen la misma dimensión. Por ejemplo (1.0,3.0,4.0)*(2.0,1,0,0.0) = 1.0*2.0 + 3.0*1.0+4.0*0.0 = 6.0

  Sugerencia: utilizar las funciones `foldRight` o `foldLeft` de la clase List, además de alguna otra función del API List (`filter`, `map`, `zip`, etc)

- Redefiniciones de los métodos `toString` (mostrando el vector con el formato “(v_1, v_2, v_3)", `equals`, para asegurar que dos vectores con los mismos elementos se consideren iguales, y `hashCode`.

Ejercicio 2 (2 puntos)

Escribe una función **recursiva de cola** con la siguiente definición: **`def`**

`propercuts[A](list:List[A]):List[(List[A],List[A])]`

que dada una lista list construye una lista con los pares de listas no vacías (list1,list2) tales que `list1++list2=list`. Por ejemplo,

```
propercuts(List(1,2,3,4)) =
List((List(1, 2, 3),List(4)), (List(1, 2),List(3, 4)), (List(1),List(2, 3, 4)))
```

Nota: utiliza alguna función del API Lista para partir la lista en dos sublistas

Ejercicio 3 (2 puntos)

a) Escribe la función `merge` con la siguiente definición:

```
def merge[A](lq:(A,A)=>Boolean)(l1:List[A],l2:List[A]):List[A]
```

que dada una función comparación `lq` mezcla de forma ordenada las dos listas utilizando `lq`. Por ejemplo, `merge[Int](_<=_)(List(1,2,3),List(1,3,4)) = List(1,1,2,3,3,4)`

b) Define e implementa una función polimórfica `mergesort` que, haciendo uso de una función mezcla del tipo de `merge(lq)`, ordene una lista siguiendo el algoritmo de ordenación por mezcla. Por ejemplo,

```
mergeSort[Int](genMerge(_<=_))(List(3,5,2,1,3)) = List(1, 2, 3, 3, 5)
```

Ejercicio 4 (3 puntos)

a) Implementa la función powerset con la siguiente definición :

```
def powerset[A](list:List[A]):List[List[A]]
```

que, asumiendo que la lista de entrada `list` no tiene elementos repetidos, construye la lista de todas las sublistas posibles de `list`. Por ejemplo,

```
powerset(List(1,2,3))=List(List(1, 2, 3), List(1, 2), List(1, 3), List(1),
List(2, 3), List(2), List(3), List())
```

Nota: el orden de las sublistas en la lista devuelta no importa.

b) Usando la función `powerset`, implementa la función

```
def knapsack(n:Int,list:List[Int]):Option[List[A]]
```

que devuelve un valor `option` con una sublista de `list` cuyos elementos sumen `n`, si existe o `None` en otro caso. Por ejemplo,

`knapsack(5,List(1,2,3,4,5))` podría devolver `Some(List(1, 4))`
`knapsack(18,List(1,2,3,4,5)) = None`

Sugerencia: utilizar funciones funciones del API List (`filter`, `map`, `zip`, etc)

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