UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación avanzada II
GII A - 28/4/2025

En este examen crearemos un proyecto en **/Usuario/Alumno/Documentos** con nombre **GIIPAIIA**. Se desarrollarán todas las clases y funciones solicitadas en un mismo worksheet de Scala (**control1.sc**), que entregaremos usando la tarea creada para tal fin en el Campus Virtual.

Algunos ejercicios indican la forma en que deben ser implementados. Las soluciones que no cumplan con las indicaciones serán consideradas como incorrectas. La eficiencia de las soluciones será tenida en cuenta.

Ejercicio 1

Supongamos el siguiente trait con las operaciones básicas de conjuntos.

```
trait ImmutableSet[T] {
  def add(elem: T): ImmutableSet[T] // añade el elemento elem al conjunto si no está presente
  def remove(elem: T): ImmutableSet[T] // elimina elem del conjunto; no modifica el conjunto si no está
  def contains(elem: T): Boolean // comprueba si un elemento está en el conjunto
  def size: Int // devuelve el número de elementos en el conjunto
  def isEmpty: Boolean // comprueba si el conjunto está vacío
}
```

Proporciona una clase `SimpleSet[T]` que implemente `ImmutableSet[T]`. La clase debe usar una `List[T]` para almacenar los elementos **sin repeticiones**. Además de los métodos del trait, la clase `SimpleSet[T]` proporcionará:

- Un constructor principal privado que tome un argumento de tipo `List[T]`.
- Un constructor que tome una secuencia de valores del tipo correspondiente (`this(T*)`).
- Un método `toList: List[T]` que devuelva los elementos del conjunto como una lista.
- Un método `union(other: SimpleSet[T]): SimpleSet[T]` que devuelva la unión del conjunto actual y el conjunto que recibe como argumento. La implementación debe usar match y recursión de cola.
- Un método `intersection(other: SimpleSet[T]): SimpleSet[T]` que devuelva la intersección del conjunto actual y el conjunto que recibe como argumento. La implementación debe utilizar funciones de orden superior.
- Un método `difference(other: SimpleSet[T]): SimpleSet[T] que` devuelva la diferencia entre el conjunto actual y otro conjunto (elementos que están en el primero pero no en el segundo). La implementación debe utilizar la función `foldLeft`.
- Redefiniciones de los métodos `toString` (mostrando el conjunto con el formato `"Set(1, 2, 3)"`), `hashCode` (utilizando `foldRight`, calculando el valor hash simplemente como la suma de los valores hash de cada uno de los elementos, sin preocuparnos de posibles desbordamientos) y `equals` para asegurar que dos conjuntos con los mismos elementos se consideren iguales.

No se utilizarán conjuntos predefinidos para ninguna de las operaciones anteriores.

Ejercicio 2

Dada una lista de frases, queremos contar la frecuencia de las palabras significativas de las cadenas que empiecen por `"FINAL"` (sin distinguir mayúsculas y minúsculas). Por ejemplo, dada la lista

```
val sentences = List(
  "FINAL: Scala is a functional language",
  "DRAFT: The power of functional programming is great",
  "DRAFT: Programming is elegant",
  "FINAL: Functional programming is elegant",
  "FINAL: Object-oriented programming is great")
```

y el conjunto de palabras no significativas

```
val stopWords = Set("a", "the", "is", "of")
```

queremos como salida un map

```
HashMap(programming -> 2, language -> 1, object-oriented -> 1, scala -> 1, elegant -> 1,
functional -> 2, great -> 1)
```