UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación avanzada II
Lab 2.2. Colas mutables e inmutables

En esta práctica, implementaremos diferentes versiones de una cola (*queue*) en Scala con el objetivo de comprender distintas estrategias para su implementación y optimización:

1. **Una cola mutable sobre un array**: Utilizaremos un `ArrayBuffer` para gestionar la estructura de la cola de manera eficiente.
2. **Una cola inmutable simple**: Implementaremos una cola basada en una estructura propia, donde los elementos se extraen por un extremo y se agregan recursivamente por el otro.
3. **Una cola inmutable eficiente**: Implementaremos una versión optimizada con dos listas, permitiendo extracción directa desde el frente y una inserción eficiente en el otro extremo.

Ejercicio 1: Implementación de colas mutables sobre arrays

- Implementa un `trait MutableQueue[T]` con las operaciones básicas:
  - `enqueue(elem: T): Unit` que añade un elemento al final de la cola.
  - `dequeue(): Option[T]` que extrae un elemento del frente de la cola, si existe.
  - `isEmpty: Boolean` que indica si la cola está vacía.
- Implementa una clase `ArrayQueue[T]` que extienda `MutableQueue[T]`, usando un `ArrayBuffer[T]` como estructura subyacente.
- Además de los métodos del trait, la clase `ArrayQueue[T]` proporcionará un constructor sin argumentos y otro que acepte múltiples valores (`T*`) y métodos `toString`, `equals` y `hashCode`.

Ejercicio 2: Implementación de colas inmutables simples

- Implementa un `trait ImmutableQueue[T]` con las operaciones básicas:
  - `enqueue(elem: T): ImmutableQueue[T]` que devuelve una nueva cola con el elemento añadido.
  - `dequeue(): (Option[T], ImmutableQueue[T])` que devuelve una tupla con el elemento extraído y la nueva cola.
  - `isEmpty: Boolean` que indica si la cola está vacía.
- Implementa una clase `SimpleQueue[T]` que extienda `ImmutableQueue[T]` y proporcione un constructor sin argumentos y otro con `T*` y métodos `toString`, `equals` y `hashCode`.

Ejercicio 3. Implementación de colas inmutables eficientes

- Implementa una clase `EfficientQueue[T]` que extienda `ImmutableQueue[T]` y que utilice dos listas (`front` y `rear`) para almacenar los elementos de la cola. Si visualizamos la cola como una secuencia de elementos y la dividimos en algún punto, en front tendríamos los elementos al frente de la cola, en rear los elementos en la cola. Para hacerlo más eficiente mantendremos los elementos en rear invertidos, de forma que:
  - Para extraer un elemento lo extraemos de la cabeza de front. Cuando `front` esté vacío, antes de sacarlo, invertimos `rear` y lo transferirmos a `front`.
  - Para insertar un elemento lo insertamos en la cabeza de rear
- La clase proporcionará un constructor sin argumentos y otro con `T*`, y redefiniciones de los métodos `toString`, `equals` y `hashCode`.