ESTRUCTURA DE DATOS

Práctica Laboratorio 2

Grado en Ingeniería

Jordi Lazo Florensa

Alejandro Clavera Poza

Universitat de Lleida
Escola Politècnica Superior

Tarea 1

Este problema consiste en dividir una lista de enteros en 2 listas diferentes a partir de un elemento pivote, donde una contendrá los valores más pequeños que el pivote y la otra los valores más grandes.

Para resolver este problema nos hemos valido de dos clases **PairOfList** la cual almacena dos listas mediante un par **ArrayList** de enteros. Y la clase **Partitioner** la cual contiene un método que realiza la partición de la lista de enteros, donde guardará en una lista los elementos más pequeños o iguales que el pivote y en la otra lista los elementos más grandes que este.

Tarea 2

Al igual que la tarea 1 este problema consiste en dividir una lista en dos diferentes a partir de un elemento pivote con la diferencia que ahora la lista puede ser de cualquier tipo de elemento y sus subtipos, es decir, genérica.
Esta tarea la hemos realizado de dos maneras diferentes. Mediante el uso de elementos que implementa el interfaz **Comparable** y un **Comparator**.

Solución Comparable:
Hemos utilizado dos clases **PairOfList** la cual almacena dos listas mediante un par **ArrayList** de elementos genéricos. Y la clase **Partitioner** que contiene el método **partition** que realiza la partición de la lista de elementos genéricos. Este método para poder comparar el elemento de la lista con el elemento pivote utilizamos la función **compareTo** la cual la tienen todos los elementos de la lista ya que la función solo admite elementos que implementen la interfaz.

Solución Comparator:
Para realizar esta solución hemos añadido una nueva clase llamada **Comparate** que implementa la interfaz **comparator**. Esta clase será la encargada de comparar dos elementos (para facilitar la comparación de cualquier elemento obliga que los elementos a comparar tengan implementado la interfaz **Comparable**, y así, poder aprovechar el método **compareTo**).
Además, utilizamos las clases **Partitioner** y **PairOfList** que ya hemos explicado anteriormente con la única diferencia que ahora se usará otro método de la clase **Partitioner** el cual utilizará un elemento comparador de la clase **Comparate** que se encargará de comparar cada elemento de la lista con el elemento pivote. Finalmente los añadirá a la lista correspondiente.

Tarea 3

La tarea tres consiste en copiar directamente los valores de una lista en dos listas diferentes seleccionadas anteriormente en función de un elemento pivote. De esta manera ya no haremos uso de la clase **PairOfList**.
Para resolver el problema lo hemos realizado de dos maneras diferentes. Mediante el uso de elementos que implementa el interfaz **Comparable** y un **Comparator**.

Solución Comparable:
Hemos usado la clase **Partitioner** la cual contiene el método **copyPartition** que recibe dos listas donde guardará cada parte de la partición. Este método compara cada elemento de la lista con el elemento pivote utilizando la función **compareTo** que implementan todos los elementos de la lista ya que la función solo admite elementos que implementen la interfaz **Comparable**.

Solución Comparator:
Para realizar esta solución hemos añadido una nueva clase (junto a la clase anteriormente usada **Partitioner**) llamada **Employeeomparator** que implementa la interfaz **comparator**. Esta clase será la encargada de comparar dos elementos Empoye.
En la clase **Partitioner** hemos añadido un nuevo método **copyPartition** que recibirá además de las listas un elemento comparador de la clase **Employeeomparator** el cual compara el pivote con los otros elementos de la lista para poder determinar en qué lista se debe colocar.

ANEXO

Para la realización de los test hemos definido la siguiente jerarquía de clases: