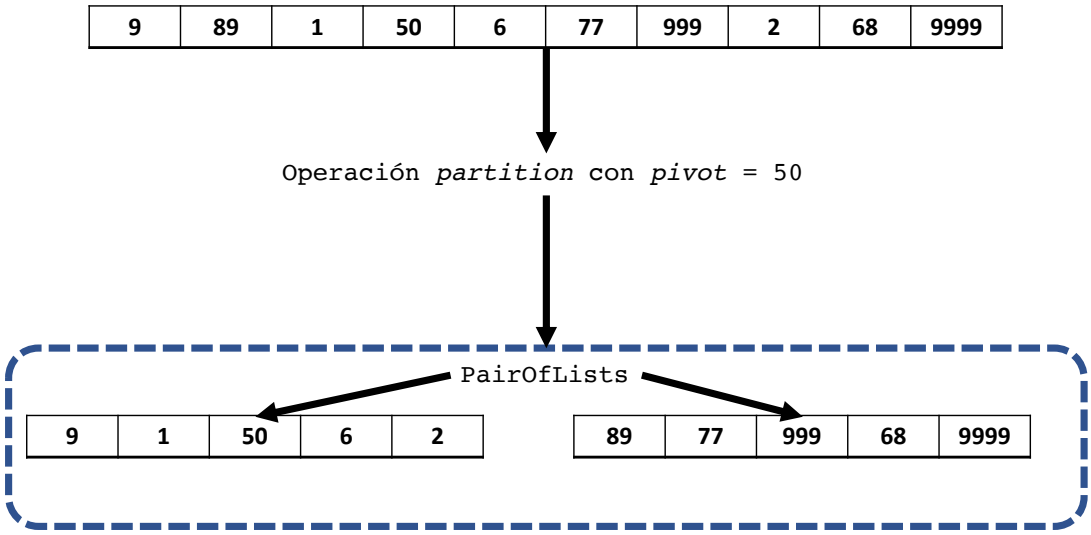
Laboratorio 2 – Genéricos, Iteradores y Comparadores

Este laboratorio tiene tres objetivos fundamentales:

1. Utilizar por primera vez Genéricos.
2. Comenzar a trabajar con iteradores.
3. Uso de las interfaces Comparable y Comparator.

Tarea 1

Implementar un método (*PairOfLists partition*) que reciba una lista de enteros junto a un entero, denominado *pivot*. Como resultado, el método devolverá un par de listas de enteros donde una contendrá los enteros menores o iguales que *pivot* de la lista recibida y la otra, los mayores. En concreto, la operación deberá realizar lo siguiente:



La cabecera del método sería la siguiente:

```
public class Partitioner{
    //. . .
    public static PairOfLists
                              partition(List<Integer> src,
                                        int pivot){
    //. . .
    }
}
```

Para recorrer la lista de enteros, será necesario utilizar un iterador para esa colección (un ejemplo de uso de un iterador se encuentra en el Tema 2). Recordad que un iterador es un objeto que nos permite recorrer una colección de manera secuencial. Independientemente del tipo de colección y como ésta se haya implementado, siempre se sabe cómo se comporta el iterador:

```
public interface Iterator<E>{
    //. . .

    boolean hasNext(); //Devuelve positivo si el iterador tiene
                       //más elementos

    E next() //Devuelve el siguiente elemento en la iteración

    void remove() //Elimina el último elemento devuelto por next()
}
```

Dado que habrá que crear una lista, podéis crear un *ArrayList* como implementación concreta de *List*. Las cabeceras del constructor y de los métodos necesarios son las siguientes:

```
public class ArrayList<E>{
    //. . .

    ArrayList() //Construye una lista vacía

    Iterator<E> iterator() //Genera un iterador sobre la lista

    boolean add(E e) //Añade un elemento al final de la lista

    //. . .
}
```

Una vez ha sido implementado el método, haciendo uso de JUnit 5, comprobar el correcto funcionamiento del método.

Tarea 2

Esta segunda tarea consiste en crear la versión genérica del método implementado en la tarea anterior. Por lo tanto, ahora habrá que implementar el método PairOf*Lists*<*E*> *partition,* el cuál vuelva a recibir una lista, en este caso genérica (*List*<*E*>) así como un elemento de tipo E que define el pivote. Como resultado, el método devolverá una pareja de listas *PairOfLists*<*E*>: una con los elementos menores o iguales que el pivote (ahora de tipo E); y otra con los elementos mayores. En este sentido, será necesario crear la versión genérica de la clase *PairOfLists*.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

La implementación se debe realizar de dos maneras:

1. Trabajando con elementos comparables (*Comparable<E>*).
2. Añadiendo y utilizando un comparador (*Comparator<E>*). Este comparador será pasado como parámetro de la función.

En base a lo anterior, las cabeceras de dichas funciones serán:

```
public static ¿? PairOfLists<E> partition(List<E> scr, E pivot)
public static ¿? PairOFLists<E> partition(List<E> src,
                                          E pivot,
                                          Comparator<E> comp)
```

Una vez ha sido implementado el método, haciendo uso de JUnit 5, comprobar el correcto funcionamiento del método.

Tarea 3

La última tarea consiste en implementar un método *copyPartition*, que copie de una lista *List<E> src*, los elementos menores o iguales que un pivote de tipo *E* en una lista *List<E> trg1* y los mayores, en otra lista *List<E> trg2*. Se pasarán como parámetros tanto la lista origen como el pivote y las listas *trg1* y *trg2*. De nuevo, al igual que en la Tarea 2, la implementación se debe realizar de dos maneras:

1. Trabajando con elementos comparables (*Comparable<E>*).
2. Añadiendo y utilizando un comparador (*Comparator<E>*). Este comparador será pasado como parámetro de la función.

Finalmente, se debe intentar generalizar al máximo la signatura de ambos métodos usando comodines.

Una vez ha sido implementado el método, haciendo uso de JUnit 5, comprobar el correcto funcionamiento del método.

PISTA 1: Necesitaréis definir una jerarquía de clases y subclases de varios niveles para poder probar los casos generales.

PISTA 2: Cuando vayáis a realizar los test, podría ser de vuestro interés hacer uso del método estático *List.of* incluido en Java desde la versión 9 y que crea una lista inmutable con el comando *List.of(E…elements)*. Si lo que se necesita es una lista mutable, se puede pasar como parámetro del constructor de ArrayList: *new ArrayList<>(List.of(E…elements))*. Para más información: https://docs.oracle.com/javase/9/docs/api/java/util/List.html#of--

Otras consideraciones

No olvidéis compilar usando la opción –Xlint:unchecked para que el compilador os avise de errores en el uso de genéricos.

Sin hacer uso de este flag, el compilador da por bueno cosas como:

```
Iterator it = l.iterator();
```

o

```
List l = new ArrayList();
```

Es decir, daría por bueno código que en ningún momento usa genéricos pero la práctica va esencialmente de genéricos.

Las clases genéricas han de instanciar los genéricos y, si no ponéis ese flag de compilación, el compilador no os ayudará (simplemente muestra un *warning* en *messages* que o no leéis, o al que no le dais importancia).

En resumen: estaréis solos ante el peligro de equivocaros. Por lo tanto, no olvidéis que el error es vuestro; sois vosotros los que habéis escrito el código y vosotros los que debéis entender cómo se debe trabajar con genéricos.

Para añadir el flag:

Preferences > Build, Execution, Deployment > Compiler > Java Compiler y lo añadís al apartado "Additional Command Line parameters".

Otra opción es:

File > Other settings > Preferences for new projects
En este caso, se añadirá cuando creéis proyectos nuevos (si no, la tenéis que añadir proyecto a proyecto aunque no está de más comprobar que está bien puesta).

Entrega

El resultado obtenido debe ser entregado por medio de un proyecto IntelliJ por cada grupo, comprimido y con el nombre “Lab2_NombreIntegrantes”. En el proyecto deberán estar presentes los test realizados con JUnit 5.

Además, se debe entregar un documento de texto, máximo de cuatro páginas, en el cual se explique la solución a cada una de las tareas.

Universitat de Lleida
Departament d’Informàtica
i Enginyeria Industrial

Consideraciones de Evaluación

A la hora de evaluar el laboratorio se tendrán en cuenta los siguientes aspectos:

- El código de cada proyecto ofrece una solución a cada una de las tareas planteadas.
- Realización de test con JUnit 5.0.
- Explicación adecuada del trabajo realizado, es decir, esfuerzo aplicado al documento de texto solicitado.
- Calidad y limpieza del código.