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 (*List<Integer> withinRange*) que reciba una lista de enteros junto a dos enteros que definen un rango (uno será un valor mínimo y otro un valor máximo). Como resultado, el método devolverá una lista de enteros que serán aquellos enteros de la lista recibida como parámetro cuyo valor se encuentre dentro del rango definido por el valor mínimo y máximo recibido. En concreto, por cada elemento *elem* de la lista se realiza la siguiente comprobación:

$$valor\ mínimo \leq elem\ < valor\ máximo$$

En cuanto a la cabecera del método, sería la siguiente:

```
public class Task{
    . . .

    public static List<Integer> withinRange(List<Integer> l,
                                            , int max
                                            , int min)
    . . .
}
```

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 cómo se ha 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 4, 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 *List<E> withinRange,* el cuál vuelva a recibir una lista, en este caso genérica (*List<E>*) así como dos elementos de tipo E que definen un rango (uno será un valor mínimo y otro un valor máximo). Como resultado, el método devolverá una *list<E>* con los elementos que estén en la lista recibida como parámetro y se encuentren dentro del rango definido por el valor mínimo y máximo, también recibidos como parámetros.

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.

Es decir, las cabeceras de dichas funciones serán:

```
public static ¿? List<E> withinRange(List<E>, E max, E min)
public static ¿? List<E> withinTange(Comparator<E> comp,
                                                    , List<E>,
                                                    , E max,
                                                    , E min)
```

Una vez ha sido implementado el método, haciendo uso de JUnit 4, comprobar el correcto funcionamiento del método.

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

Tarea 3

La última tarea consiste en implementar un método *copyWithRange*, que copie de una lista en otra, ambas pasadas como parámetros (*trg* y *src*), los elementos que se encuentren dentro del rango definido por dos valores (valor máximo y mínimo), también pasados como parámetros. De nuevo, al igual que 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 4, comprobar el correcto funcionamiento del método.

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

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 4.

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.

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 4.0.
- Explicación adecuada del trabajo realizado, es decir, esfuerzo aplicado al documento de texto solicitado.
- Calidad y limpieza del código.