Laboratorio 2 – Programación Orientada a Objetos, *Comparable*, Iteradores, Genéricos y Comodines

El objetivo de este laboratorio es practicar conceptos de Programación Orientada a Objetos y del *Java Collections Framework* vistos en teoría. Se considera adecuado trabajar ciertos conceptos que serán de gran utilidad en lo que queda de curso.

La práctica está compuesta por tres ejercicios descritos a continuación.

Ejercicio 1: Implementación de un método *print* genérico de listas en el cual se haga uso de *Iterator<T>*

A lo largo de este ejercicio se debe implementar un método *print* genérico de manera que, mediante un iterador, recorra todos los elementos de una lista de personas (profesor o lector) teniendo en cuenta las siguientes especificaciones:

- Una persona tendrá un nombre y DNI.
- Un profesor es una persona. Además de nombre y DNI debe tener un despacho.
- Un lector es un profesor. Junto a nombre, DNI y despacho tiene un grupo de investigación asociado.

La lista de profesores será un *ArrayList<T>* y la de lectores un *LinkedList<T>*. Estas listas llegarán al método *print*, el cual debe escribir por consola (*System.out.println()*) todos los elementos de ambas listas. Para recorrer las listas, concretamente, sólo se hará uso de *Iterator<T>*.

En este punto se debe recordar que tal y como se ha visto en la parte de teoría, un *Iterator<T>* permite recorrer contenedores, como son las listas. En concreto, en JCF, las listas tienen un método que devuelve un *Iterator<T>* sobre la lista, en este caso, sobre *ArrayList<T>* o *LinkedList<T>*.

También es necesario indicar que *ArrayList<T>* y *LinkedList<T>* son clases implementadas en JCF y que se verán posteriormente en la asignatura. Ambas son de tipo *List<T>*.

Para poder escribir información de los elementos de ambas colecciones (de profesores y lectores) es necesario implementar el método *toString()* de las

clases *Persona*, *Profesor* y *Lector*. Dado que *toString()* es un método definido en la clase *Object*, superclase de toda clase en Java, el objetivo es redefinir esta operación.

En concreto y, para resumir, el método *main* debe ser capaz de hacer lo siguiente:

```
import java.util.*;
public class Laboratorio2 {

  public static void main(String[] args) {

    // Creación de ArrayList de profesores
    ArrayList<Profesor> profesores = new ArrayList<Profesor>();
    Profesor profe1 = new Profesor("11111111A", "Juan", "3.01");
    Profesor profe2 = new Profesor("22222222A", "Antonio", "3.02");
    profesores.add(profe1);
    profesores.add(profe2);

    LinkedList<Lector> lectores = new LinkedList<Lector>();
    Lector lec1 = new Lector("11111111A", "Juan", "3.01", "GI_1");
    Lector lec2 = new Lector("22222222A", "Kike", "3.02", "GI_3");
    lectores.add(lec1);
    lectores.add(lec2);

    //Escribo profesores
    System.out.println("Lista de profesores:");
    imprimirLista(profesores);

    //Escribo lectores
    System.out.println("Lista de lectores:");
    imprimirLista(lectores);
  }

//Aquí irá la función print (imprimirLista en ejemplo)
}
```

A continuación, se lista lo que se debe hacer:

- Diseñar e implementar las clases *Persona*, *Profesor* y *Lector*.
  - Establecer relación de herencia.
  - Redefinir método *toString()*.
- Implementar método *print* genérico que permita escribir todos y cada uno de los elementos de una lista (de inicio a final), en este caso *ArrayList<T>* y *LinkedList<T>*, por medio de un *Iterator*.
- ¿Más información?
  - https://docs.oracle.com/javase/8/docs/api/java/util/List.html

La entrega de este ejercicio consiste en un proyecto Netbeans con los ficheros:

- Persona.java
- Profesor.java
- Lector.java
- Ejercicio1.java

Ejercicio 2: Implementación de un método *sort* genérico con comodines para listas

Se debe implementar un método *sort* genérico que permite ordenar un *ArrayList<T>* de *Personas* en base a su DNI y un *LinkedList<T>* de *Coches* en base a su matrícula. Este método debe ser válido para ambas, es decir, genérico, no siendo válido realizar uno para ordenar *Personas* y otro para ordenar *Coches*. Queda libre la elección de si el orden debe ser de mayor a menor o viceversa, así como el algoritmo de ordenación (*quicksort*, *bubblesortm*, etc).

El método *sort* debe poder trabajar con objetos de tipo *ArrayList<T>* y *LinkedList<T>* por lo que es necesario pensar de qué tipo es el parámetro que recibe. Si es de tipo *ArrayList<Personas>* funcionará para la lista de *Personas* pero no para la de *Coches*. ¿Comodines? ¿Subtipos? Cabe recordar que cuando hablamos de que una clase es del tipo de otra, suele referirse a que hereda (*extends* una clase abstracta, clase no abstracta o una interfaz) o implementa (*implements* una interfaz).

Ahora el método *main* debe ser capaz de hacer lo siguiente:

```
import java.util.*;
public class Laboratorio2 {

  public static void main(String [] args) {

    // Creación de ArrayList de Personas
    ArrayList<Persona> personas = new ArrayList<Persona>();
    Persona per1 = new Persona("11111111A", "Juan");
    Persona per2 = new Persona("22222222A", "Antonio");
    personas.add(per1);
    personas.add(per2);

    LinkedList<Coche> coches = new LinkedList<Coche>();
    Coche c1 = new Coche("Juan", "L-1111");
    Coche c2 = new Coche("Kike", "C-4444");
    coches.add(c1);
    coches.add(c2);

    //Ordeno personas
    sort(personas);

    //Ordeno coches
    sort(coches);

    //Escribo personas ordenadas
    System.out.println("Lista de personas ordenadas:");
    imprimirLista(personas);

    //Escribo lectores
    System.out.println("Lista de coches ordenados:");
    imprimirLista(coches);
  }
```

```
//Aquí irá función sort
}
```

Como se puede comprobar, también se pide diseñar e implementar las clases *Persona* y *Coche*. Deben ser de tipo *Comparable*<*T*>, interfaz definida en JCF.

Concretando se debe hacer lo siguiente:

- Implementar la clase *Persona* que sea del tipo *Comparable*<*T*>.
- Implementar la clase *Coche* que sea del tipo *Comparable*<*T*>.
- Implementar un método *sort* genérico para listas en la clase principal. Será *static* y no devolverá nada (*void*).

La entrega de este ejercicio consiste en un proyecto Netbeans con los ficheros:

- Persona.java
- Coche.java
- Ejercicio2.java

Ejercicio 3: Diseñar un método *copy* genérico con listas y comodines

En teoría se han visto los comodines con *super* (<*? super T*>) y con *extends* (<*? extends T*>). En base a esto, en este ejercicio se debe diseñar un método genérico *copy* que tenga dos parámetros: (1) una lista llamada *destino*; y (2) una lista *origen*. El objetivo es copiar la lista origen en el destino.

```
public static <T> copy (List <? X T> destino, List <? Y T> origen) {
    for(int i=0; o<origen.size(); i++)
        destino.set(i, origen.get(i));//sustituye destino(i) por
                                      //origen(i)
}
```

En este ejercicio, la tarea consiste en explicar si X(Y) debe ser *super* o *extends* y por qué. El razonamiento de la respuesta no debe ser más de 3 líneas.

La entrega de este ejercicio consiste en un fichero Ejercicio3.txt con la respuesta.

Entrega

El resultado obtenido debe ser entregado por medio de un único fichero comprimido por cada grupo con el nombre "Lab2_NombreIntegrantes", en el

cual existan tres carpetas, una para cada ejercicio. En la descripción de cada uno de los ejercicios se indica qué debe contener su carpeta.

Consideraciones de Evaluación

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

- Solución a los tres ejercicios planteados.
- Calidad y limpieza del código.
- Diseño orientado a objetos.
- Comentarios en código.
- Realización de tareas opcionales.