EJERCICIO 1

a) ¿Cuáles son las implementaciones de propósito general más comunes para cada una de las interfaces de la API de colecciones de Java? (*List*, *Set* , *Queue*, *Map*)

b) Si se necesitase algún tipo de orden entre los elementos de las colecciones que se quieran usar, ¿qué interfaces son las más adecuadas para proporcionar ese orden? ¿De qué dos formas se puede establecer el orden en una colección?

c) ¿Qué clase de la jerarquía de colecciones Java proporciona una serie de métodos estáticos de propósito general para realizar distintas funciones sobre las colecciones?

EJERCICIO 2

Crear una clase Java que haga lo siguiente:

- Crear una clase propia que contenga un objeto de tipo *HashSet*.

- Añadir los siguientes 7 elementos, debidamente inicializados, como valores para el *HashSet* creado: 2 String, 2 objetos Persona, 3 enteros (int o Integer).

  · La clase *Persona* deberá ser creada de forma adicional (por ejemplo, como clase interna auxiliar), y contendrá como campos: nombre (*String*), letra inicial del nombre(*char*) y edad (*int*), con sus métodos de acceso y modificación.

  Se puede crear como una clase estática interna a la clase principal del ejercicio. Es decir, dentro de la clase principal, como: *static class Persona { ... }*

- Crear un iterador para recorrer los elementos del conjunto, y listarlos por pantalla.

- [*OPCIONAL*] Repetir el proceso para un *LinkedList* y un *ArrayList*.

EJERCICIO 3

Crear un método *findDup* que permita detectar los elementos duplicados en una colección de String. El método recibirá un array estático de String, no devolverá ningún valor, pero imprimirá por pantalla dos tipos de acciones.

Por un lado, cada vez que detecte un elemento duplicado lo imprimirá por pantalla. Por otro lado, listará al final del todo el *conjunto* de elementos únicos, es decir, la colección original de elementos menos los duplicados, así como el conjunto de elementos sin duplicado, es decir, aquellos para los que no se ha detectado duplicado.

La signatura del método será: `void finDup(String[] elements)`

Por ejemplo, si tenemos el conjunto de elementos:

```
String[] rios = {"Tajo", "Júcar", "Segura", "Tajo", "Duero", "Ebro", "Ebro"}
```

Y llamamos a la función de la manera siguiente: `findDup(rios);`

Su salida por pantalla debería ser:

```
Elemento duplicado: Tajo

Elemento duplicado: Ebro
```

```
Elementos únicos:
[Júcar, Tajo, Duero, Ebro, Segura]

Elementos sin duplicados:
[Júcar, Duero, Segura]
```

Nota: un método de utilidad pordría ser *removeAll* de la interfaz *Set* (consultar API)

SOLUCIÓN EJERCICIO 1

a) List (*ArrayList*) | Set (*HashSet*) | Queue (*LinkedList*) | Map (*HashMap*)

b) Las interfaces de colección ordenadas en Java son *SortedSet* y *SortedMap*, cuyas implementaciones más comunes son *TreeSet* y *TreeMap*, respectivamente.

El orden en dichas colecciones se puede establecer según el orden natural de los elementos de la colección (su orden interno), o bien según el orden total que defina un comparador externo a las colecciones.

c) La clase *Collections*, en la cual todos sus métodos son estáticos.

SOLUCIÓN EJERCICIO 2

El código fuente de una posible solución al ejercicio 2 sería:

```
public class MiClase {
   // conjunto interno de la clase
   Set<Object> miConjunto = new HashSet();
   // constructor donde se realizarán todas las operaciones
   public MiClase() {
      // añadimos los elementos pedidos
      miConjunto.add("Cadena 1."); miConjunto.add("Cadena 2.");
      miConjunto.add(new Persona("Juan", 'J', 23));
      miConjunto.add(new Persona("Ana", 'A', 27));
      miConjunto.add(5); miConjunto.add(78); miConjunto.add(0);
      // recorremos con el iterador
      Iterator<Object> it = miConjunto.iterator();
      while(it.hasNext())
         System.out.println(it.next()); }

   static class Persona {
      private String nombre;
      private char letraInicial;
      private int edad;

      public Persona(String nombre, char letraIncial, int edad) {
         this.nombre = nombre;
         this.letraInicial = letraIncial;
         this.edad = edad; }
      // métodos de acceso y modificación ... } }
```

SOLUCIÓN EJERCICIO 3

El código fuente de una posible solución al ejercicio 3 sería:

```
void findDup(String[] elements) {
    // lista auxiliar que copia los parámetros de entrada
    List<String> auxList = Arrays.asList(elements);
    // lista para almacenar los duplicados que encontremos
    List<String> duplicados = new ArrayList();
    // conjunto para almacenar los elementos sin repetir ninguno
    Set<String> auxSet = new HashSet();

    // recorremos los elementos, mostramos mensaje por pantalla
    // si encontramos un duplicado, y lo añadimos a los duplicados
    for(String cad : auxList) {
      if(!auxSet.add(cad)) {
        System.out.println("Elemento duplicado: " + cad);
        duplicados.add(cad); } }

    // listamos los elementos únicos
    System.out.println("\nElementos únicos:\n" + auxSet);
    // eliminamos del conjunto, aquellos con un duplicado originalmente
    auxSet.removeAll(duplicados);
    // listamos el conjunto de los elementos sin duplicados
    System.out.println("\nElementos sin duplicados:\n" + auxSet); }
```