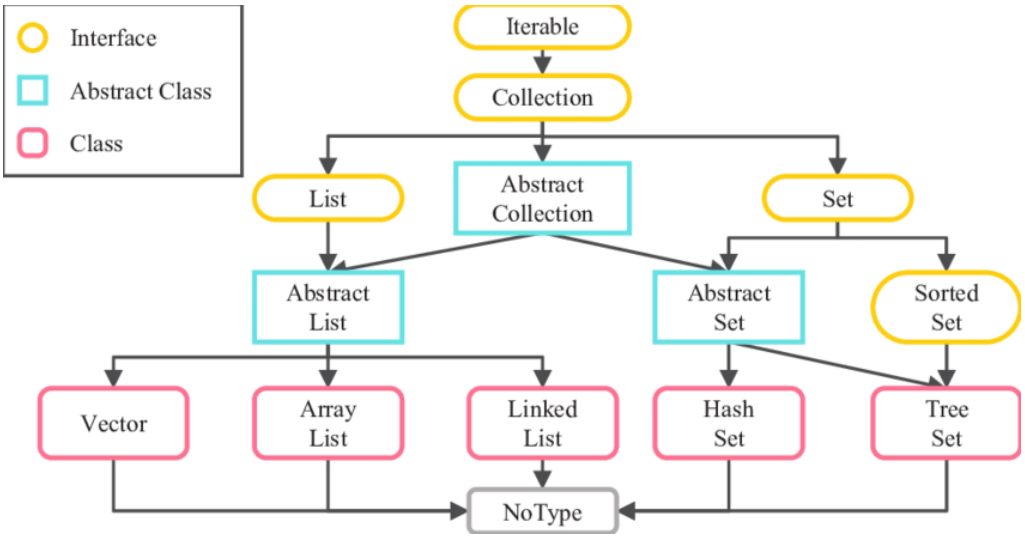
Ejercicios de la UD07



1. Ejercicios

1. (`package Varios`) Diseñar la clase `Varios` con los siguientes métodos **estáticos** que se harán apoyándose en alguna clase de las vistas al estudiar las colecciones de Java:
    - `int[] quitarDuplicados (int[] v)`, que dado un array de enteros devuelva otro array con los mismos valores que el original pero sin duplicados.
    - `int[] union1(int[] v1, int[] v2)`, que dados dos arrays v1 y v2 devuelva otro array con los elementos que están en v1 o que están en v2, sin que ningún elemento se repita.
    - `int[] unión2(int v1[], int v2[])`, que dados dos arrays v1 y v2 devuelva otro array con los elementos que están en v1 o que están en v2. En este caso, si hay elementos duplicados se mantendrán.
    - `int[] interseccion(int v1[], int v2[])`, que dados dos arrays v1 y v2 devuelva otro array con los elementos que aparecen en los dos arrays. Cada elemento común aparecerá una sola vez en el resultado.
    - `int[] diferencia1 (int v1[], int v2[])`, que dados dos arrays v1 y v2 devuelva otro array con los elementos de v1 que no están en v2. En caso de haber elementos duplicados en v1 estos se mantendrán en el resultado.
    - `int[] diferencia2 (int v1[], int v2[])`, que dados dos arrays v1 y v2 devuelva otro array con los elementos de v1 que no están en v2. El array resultante no tendrá elementos duplicados.
2. (`package Biblioteca`) Se quiere hacer una aplicación en la que los usuarios van a hacer búsquedas de libros para saber si se encuentran en los fondos de la biblioteca. El funcionamiento básico sería algo así: Al iniciarse la aplicación todo el catálogo de libros se cargaría en memoria y a partir de ese momento los usuarios pueden realizar búsquedas por título, que interesa que sean lo más rápidas posibles. Nunca se insertan nuevos libros durante la ejecución de la aplicación.
    - Diseña la clase `Libro` con los métodos que consideres oportunos y los siguientes atributos:
        - `Titulo` (String): Es el dato que identifica al libro.
        - `Autor` (String): Autor del libro.
        - `Estantería` (String): Estantería de la biblioteca en la que se encuentra el libro.
    - Diseña la clase `CatalogoLibros` como una colección de libros. Utiliza el tipo de colección que crees que más se ajusta a los requisitos de la aplicación justificando la elección. Implementa los siguients métodos:
        - `public CatalogoLibros(Libro v[])`: Constructor. Para simplificar, inicializa el catálogo y lo rellena con los libros del `array v`, en lugar de obtenerlos de un fichero.

- `public String buscar(Libro l)`: Dado un libro, lo busca en el Catálogo y devuelve la estantería en la que se encuentra el libro o null si el libro no está en el Catálogo.

3. (**`package Academia`**) Se quiere diseñar una clase `Academia`. De una Academia se conoce su nombre, dirección y las Aulas que tiene (Necesitas también generar la clase `Aula`).

   Definir la clase `Academia` utilizando una **Collection** para almacenar las aulas. El tipo de colección a utilizar se decidirá teniendo en cuenta que éstas se quieren mantener ordenadas según el criterio del método `compareTo` de la clase `Aula`. Implementar los atributos, el constructor, y los siguientes métodos:

   - `void ampliar (Aula a)`, que añade un aula a la academia.
   - `void quitar (Aula a)`, que elimina un aula de la academia.
   - `int getNumAulas()`, que devuelva el número de aulas que tiene.
   - `toString()`, que muestre todas las aulas de la academia.

4. (**`paquete ListaEspera`**) Deseamos mantener la lista de espera de pacientes de un hospital. Se quiere poder:

   - Añadir pacientes a la lista.
   - Mantener los pacientes por orden de inserción en la lista de espera.
   - Obtener el paciente más prioritario (al que hay que atender de la lista de espera), es decir, el que más tiempo lleva esperando, el que antes entró en el hospital.

   1. Diseña la clase `Paciente` con los atributos nombre y gravedad. La gravedad es un valor aleatorio (entre 1: más grave y 5: menos grave) generado al crear el paciente.
   2. Realiza la implementación de la clase `ListaEspera` con la estructura de datos elegida. Añade los métodos para insertar un paciente, obtener de la lista de espera al más prioritario y eliminar de la lista de espera al más prioritario. ¿Qué tipo de estructura de datos utilizaríamos para almacenar los pacientes, un `List`, `Set` o `Map`? Justifica la respuesta (puntos a favor de la que elijes y en contra de las que descartas).
   3. Implementa una `ListaEsperaPorGravedad`, en la que se atienda primero a los pacientes más graves, independientemente de si llegaron antes o después.

5. (**`paquete DiccionarioIngEsp`**)

   - Diseñar la clase `DiccionarioBilingüe` para almacenar pares formados por:

     1. Palabra en castellano.
     2. Colección de traducciones a ingles.

     La clase dispondrá de los siguientes métodos:

     - Constructor: crea el diccionario vacio.
     - `anyadirTraduccion(String cast, String ingl)`: Añade la pareja (cast, ingl) al diccionario de forma que:
       - Si la palabra `cast` no estaba en el diccionario la añade, junto con su traducción.

        - Si la palabra `cast` estaba ya en el diccionario pero no aparecía como traducción la palabra `ingl`, añade `ingl` a su colección de traducciones.
        - Si la palabra `cast` estaba y la traducción `ingl` también, no se realizarán cambios.
        - El método devuelve `true` si se han realizado cambios en el diccionario y `false` en caso contrario.
    - `QuitarTraduccion(String cast, String ingl)`: Quita la traducción `ingl` a la palabra `cast`. Si la palabra en castellano se queda sin traducciones, se elimina del dicicionario. Si se han producido cambios se devuelve `true` y en caso contrario `false`.
    - `traduccionesDe(String cast)`: Devuelve una colección con las traducciones de la palabra indicada o null si la palabra no está en el diccionario.
    - `toString()`: Devuelve un String con las palabras del diccionario y sus traducciones.
  - (clase `RepeticionPalabras`) Escribe un programa que abra el fichero de texto que indique el usuario y muestre cuántas veces se repite cada palabra que contiene. Ayudarse de un `Map`. ¿Se podría resolver con un `Set`? ¿Y con un `List`?
  - (clase `TraductorSimple`) Escribir un programa que solicite al usuario una frase y la muestre traducida a inglés, palabra a palabra. Para ello, se dispone de un fichero `palabras.txt` que contiene parejas (palabra en español, palabra en inglés) separadas por un tabulador. Cada pareja se encuentra en una línea del fichero. El proceso será el siguiente:
    - Leer el fichero y cargar sus datos en una estructura de datos adecuada. Tener en cuenta que nos interesará buscar una palabra en castellano y obtener su correspondencia en inglés.
    - Solicitar al usuario una frase. Traducir, usando la estructura de datos anterior, cada palabra de la frase y formar con ellas la frase traducida.
    - Mostrar la frase traducida al usuario.
6. (**`paquete ListaAdmitidos`**) Una serie de personas han solicitado realizar un curso de inglés. De las que han sido admitidas se quiere almacenar su nif, su nombre y su nivel en un `HashSet`. En el `HashSet` se almacenarán objetos de la clase Inscripción
   - Implementa la clase `Incripción` para representar el nif, nombre y nivel de un solicitante. Además de los atributos implementa aquellos métodos que consideres necesarios.
   - Escribe un programa (clase `ComprobarAdmision`) que
     - Defina un `HashSet` de Inscripciones, llamado `admitidas`.
     - Añada varias inscripciones (invéntate los datos).

    - Permita al usuario introducir un dni para comprobar si la persona indicada ha sido admitida. Indicarle si aparece o no en la lista y, en caso afirmativo, mostrar el nombre y el nivel en que ha sido admitido.

7. (`clase PalabrasOrdenadas`) Escribe un programa que, dado un fichero de texto cuya ubicación indica el usuario, muestre sus palabras ordenadas ascendentemente y, después, descendentemente. Cada palabra se mostrará una sola vez, aunque en el texto aparezca varias.

2. Actividades

Actividad 1. Realizar las siguientes actividades relacionadas con `ArrayList`.

- Crear un `ArrayList` de enteros llamado misNumeros.
- Añadir los valores 1, 6, 3, 2, 0, 4, 5.
- Mostrar los datos del `ArrayList`.
- Mostrar el valor de la posición 5.
- Añadir el valor 8 en la posición 4.
- Cambiar el valor de la posición 1 por 9.
- Eliminar el valor 5. (`misNumeros.remove(new Integer(5))`)
- Eliminar el valor de la posición 3.
- Recorrer el array con un bucle for.
- Recorrer el array con un bucle `Iterator`.
- Comprobar si existe el elemento 0.
- Comprobar si existe el elemento 7.
- Clonar el `ArrayList` `misNumeros` en otro llamado `copiaArrayList`.
- Añadir el elemento 9.
- Mostrar la posición de la primera ocurrencia del elemento 9.
- Mostrar la posición de la última ocurrencia del elemento 9.
- Borrar todos los elementos del `ArrayList` `copiaArrayList`.
- Comprobar si el `ArrayList` `copiaArrayList` está vacio.
- Convertir el `ArrayList` `misNumeros` en un `Array` y recorrerlo con un bucle mejorado.

Actividad 2. Un cine precisa una aplicación para controlar las personas de la cola para los estrenos de películas. Debemos crear una lista con la edad de las personas de la cola y tendremos que calcular la entrada según la edad de la persona (mínimo 5 años). Para la edad de la persona se generan aleatoriamente números entre 5 y 60 años. Al final, deberemos mostrar la cantidad total recaudada. El número de personas de la cola se elige al azar entre 0 y 50.

La lista de precios se basa en la siguiente tabla.

EDAD	PRECIO
Entre 5 y 10 años	5 €
Entre 11 y 17 años	7.5 €
Mayor de 18 años	9.5 €

Como comprobación imprime el número de personas, el precio total y la lista de edades. Por ejemplo:

```
Hay un total de 6 personas en la cola.
El precio total es de 57,00 euros
[18, 36, 50, 35, 28, 55]
```

Actividad 3. Un supermercado nos pide que hagamos una aplicación que almacene los productos comprados. La aplicación debe almacenar `Productos` (clase) y cada producto al crearse contiene una cantidad, un precio (generados aleatoriamente). El nombre del producto será básico (producto1, producto2, producto3, etc.). Calcular el precio total de una lista de entre 1 y 10 productos (aleatorio). Mostrar un ticket con todo lo vendido y el precio final.

Actividad 4. Desarrollar un sistema de gestión de pacientes. Tendremos un archivador dónde iremos guardando todas las fichas de los pacientes. Las fichas contienen la siguiente información: nombre, apellidos y edad.

Todas las fichas que vayamos creando, se podrán guardar o eliminar del archivador. Al archivador también le podremos pedir un listado. Este listado consistirá en visualizar por pantalla el número de fichas guardadas, así como el contenido de las fichas.

La clase `GestionPacientes` tiene un método `main` en el que se crea un archivador, dos o tres fichas que se guardarán en el archivador, se listará el contenido, se eliminará alguna ficha y se volverá a listar su contenido. Todas las clases se guardarán en el paquete `gestionpacientes`.

Actividad 5. Crear una estructura `Map` llamada divisas, que almacene pares de moneda y valor al cambio en euros. Por ejemplo Dólar: 0,81€

- Añadir los siguientes pares moneda/valor al Map divisas:

Moneda	Valor en €
Dólar Americano	0.81
Franco Suizo	0.85
Libra Esterlina	1.14
Corona Danesa	0.13
Peso Mexicano	0.04
Dólar Singapur	0.62
Real Brasil	0.24

- Mostrar el valor de la Libra Esterlina.
- Mostrar todas las divisas con las que se opera y su valor.
- Indicar el número de divisas del Map.
- Eliminar la divisa Real Brasil y mostrar los datos del Map.
- Mostrar si existe la divisa Peso Mexicano.
- Mostrar si existe la divisa Euro.
- Mostrar si existe el valor al cambio 0.85 €.

- Mostrar si existe el valor al cambio 0.33 €.
- Indicar si el Map divisas está vacío.
- Borra todos los componentes del Map divisas.
- Volver a indicar si el Map divisas está vacío.

3. Ejercicios Genericidad

1. Crear una clase Genericos (proyecto Genéricos, paquete Genericos) que incorpore los métodos genéricos que se indican a continuación. Los métodos creados serán *public static.* En el proyecto se creará además la clase o clases necesarias para probar los métodos desarrollados.

    1. `Object minimo (Object o1, Object o2)`, que devuelva el mínimo de dos objetos cualesquiera (que se suponen del mismo tipo). Una vez desarrollado, prueba el método para obtener el mínimo de dos objetos Integer. Pruébalo también para obtener el mínimo entre un Objeto Integer y un objeto String. En éste último caso, el programa ¿da error de ejecución? Si es así, explica por qué.
    2. `Object maximo (Object o1, Object o2)`, que devuelva el maximo de dos objetos cualesquiera (que se suponen del mismo tipo).
    3. `Object minimo (Object v[])`, que devuelva el mínimo de un array de objetos cualesquiera (que se suponen del mismo tipo). Al respecto de éste último comentario, ¿Se puede poner en un array de Object objetos de distinto tipo, como por ejemplo Strings, Integer, ...? En caso afirmativo, ¿funcionaría el método desarrollado con un array construido así?
    4. `Object maximo (Object v[])`, que devuelva el maximo de un array de objetos cualesquiera (que se suponen del mismo tipo).
    5. `int numVeces(Object v[], Object x)` que devuelva el el numero de apariciones del objeto x en el array v.
    6. `int numVecesOrdenado(Object v[], Object x)` que devuelva el el numero de apariciones del objeto x en el array v **ordenado ascendentemente**.
    7. `int mayores(Object v[], Object x)` que, dado un array de Object v y un Object x devuelva el número de elementos de v que son mayores que x.
    8. `int mayoresOrdenado(Object v[], Object x)` que, dado un array de Object v **ordenado ascendentemente** y un Object x devuelva el número de elementos de v que son mayores que x.
    9. `int menores(Object v[], Object x)` que, dado un array de Object v y un Object x devuelva el número de elementos de v que son menores que x.
    10. `int menoresOrdenado(Object v[], Object x)` que, dado un array de Object v **ordenado ascendentemente** y un Object x devuelva el número de elementos de v que son menores que x.
    11. `boolean estaEn(Object v[], Object x)` que devuelva true si el Objeto x está en el array v.
    12. `boolean estaEnOrdenado(Object v[], Object x)` que devuelva true si el Objeto x está en el array v, **ordenado ascendentemente**.
    13. `int posiciónDe(Object v[], Object x)`, que devuelva la posición que ocupa x dentro del array v, o -1 si x no está en v.

14. `int posicionDeOrdenado(Object v[], Object x)`, que devuelva la posición que ocupa x dentro del array v **ordenado ascendentemente**, o -1 si x no está en v.
15. `boolean estaOrdenado(Object v[])`, que devuelva true si el array está ordenado ascendentemente.

2. (**`paquete Nevera`**) Se quiere crear una aplicación que controla una nevera inteligente de última generación. Los alimentos que contiene la nevera se van a representar como objetos de la clase `Alimento` y la clase `NeveraInteligente`, tiene un array de Alimentos entre sus atributos privados.

   Se pide implementar la clase **Alimento** teniendo en cuenta que uno de los métodos de NeveraInteligente necesitará ordenar **por calorías** los Alimentos que contiene la nevera utilizando un método de Ordenación genérico. El diseño de la clase Alimento ha de incluir, por tanto, determinados elementos que lo permitan. La clase Alimento tendrá únicamente dos atributos (privados): nombre y calorias.

3. (**`paquete Academia`**) Se quiere diseñar una clase *Academia* . De una Academia se conoce su nombre, dirección y las *Aulas* que tiene (*Aula* es una clase implementada en un ejercicio de herencia que ya hicimos).
   1. Definir la clase `Academia` utilizando una colección (que permita ordenación) para almacenar las aulas.: Implementar los atributos, el constructor, y los siguientes métodos:
      - `void ampliar (Aula a)`, que añade un aula a la academia.
      - `void quitar (Aula a)`, que elimina un aula de la academia.
      - `int getNumAulas()`, que devuelva el número de aulas que tiene.
      - método `toString()`
   2. Realiza en la clase `Aula` los cambios necesarios para que se pueda ordenar las aulas de la Academia usando un método genérico de ordenación. El orden sería creciente por capacidad del aula., y a igual capacidad primero las aulas de mayor superficie
   3. Añade a la clase `Academia` un método `ordenar` que ordene las aulas con el criterio especificado. Para realizar la ordenación se llamará a un método de ordenación.

4. (**`paquete Conjuntos`**)
   1. Diseñar un **interface *Conjunto*** para modelizar conjuntos de elementos. Diseñar (solo la cabecera) de los siguientes métodos de la clase conjunto (prestar atención a si los métodos deben ser *static* o no):
      - `Añadir`, que añade un elemento al conjunto, provocando la excepción `ElementoDuplicado` si el elemento ya estaba en el conjunto.
      - `Quitar`, que elimina el elemento indicado al conjunto. Provoca `ElementoNoEncontrado` si el elemento indicado no estaba en el conjunto.
      - `Intersección`¸ que dados dos conjuntos que recibe como parámetro devuelve un tercer conjunto que es la intersección de los dos dados.

- `Pertenece`, que dado un elemento devuelve si este pertenece o no al conjunto.

2. Diseñar una clase `ConjuntoArray` que implemente el interface `Conjunto`. Esta clase implementará los métodos del interface `Conjunto`. Para ello utilizará un array `Object elementos[]` y un `int numElementos`, de manera que los elementos del conjunto se mantendrán almacenados en el array. Además de los métodos del interface habrá que crear un constructor para la clase y también vendrá bien tener un método `toString` para poder probarla.

   NOTA: También lo puedes implementar con una Colección de las vistas en el tema anterior.

4. Fuentes de información

- Wikipedia
- Programación (Grado Superior) - Juan Carlos Moreno Pérez (Ed. Ra-ma)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- Apuntes José Luis Comesaña
- Apuntes IOC Programació bàsica (Joan Arnedo Moreno)
- Apuntes IOC Programació Orientada a Objectes (Joan Arnedo Moreno)
- Apuntes Lionel