1. Escriba las sentencias necesarias en un método *main* de una clase test que vaya dando respuesta a los siguientes apartados:
   i. Crear una lista de números enteros llamada ***numeros*** cuyo contenido inicial sea [6,5,4,5,6,7,8,9,8,7] (en ese mismo orden).
   ii. Mostrar en la consola el número de elementos de la lista anterior.
   iii. Mostrar en la consola en líneas diferentes el mayor y el menor elemento de la lista anterior.
   iv. Mostrar en la consola un mensaje que indique si el valor 0 está contenido en la lista anterior.
   v. Mostrar en la consola en líneas diferentes las ***posiciones naturales*** de la primera y última aparición del valor 7.
   vi. Eliminar el primer elemento de la lista anterior y mostrar su contenido en la consola.
   vii. Eliminar el último elemento de la lista anterior y mostrar su contenido en la consola.
   viii. Ordenar la lista anterior y mostrarla ya ordenada en la consola.
   ix. Eliminar de la lista anterior la única aparición del valor 7 y mostrarla en la consola tras la operación.
   x. Eliminar todas las apariciones de los valores 5 y 8 de la lista anterior y mostrarla en la consola tras la operación.

2. Escriba las sentencias necesarias en un método *main* de una clase test que vaya dando respuesta a los siguientes apartados:
   i. Crear cuatro conjuntos de números naturales:
      i. ***multiplosDe2***, con todos los enteros positivos múltiplos de 2 menores o igual que 30;
      ii. ***multiplosDe3***, con todos los enteros positivos múltiplos de 3 menores o igual que 30;
      iii. ***multiplosDe5***, con todos los enteros positivos múltiplos de 5 menores o igual que 30;
      iv. ***menoresQue30***, con todos los enteros positivos mayor que 1 y menores que 30
   ii. A partir de los conjuntos anteriores, mediante operaciones entre conjuntos, obtener los enteros positivos múltiplos de 2, 3 y 5.
   iii. A partir de los conjuntos del apartado a) y mediante operaciones entre conjuntos, obtener los enteros positivos múltiplos de 2 que no lo son ni de 3 ni de 5.
   iv. A partir de los conjuntos del apartado a) y mediante operaciones entre conjuntos, obtener los enteros positivos múltiplos de 2 o de 3 o de 5.
   v. A partir de los conjuntos de los apartados anteriores y mediante operaciones entre conjuntos, obtener los primos mayores que 5 y menores que 30.

3. Escriba las sentencias necesarias en un método *main* de una clase test que vaya dando respuesta a los siguientes apartados:
   i. Obtener un array de char de nombre ***letras*** cuyos elementos sean todas las letras en mayúsculas de un objeto String que contiene la cadena de texto ***"fundamentos de programación"*** y mostrarlo en la consola.
   ii. Crear un array de String de nombre ***palabras*** con cinco elementos ("aprendiendo", "a", "programar", "en", "Java") y mostrarlo en la consola. Revise el método ***split*** del tipo String.
   iii. Obtener un conjunto ordenado de String de nombre ***setPalabras*** a partir del array palabras anterior y mostrarlo en la consola.

4. Dado el siguiente trozo de código fuente:

```
public static void main(String[] args) {
        List<Integer> pares = new ArrayList<>(5);
        List<Integer> impares = new ArrayList<>(5);
        Collections.addAll(pares, 2,8,4,6,10);
        Collections.addAll(impares, 3,1,5,5,5);
        SortedSet<Integer> numeros = new TreeSet<>(pares);
        numeros.addAll(impares);
```

```
        System.out.println("El conjunto de números es: " + numeros);
        Integer[] enteros = numeros.toArray(new Integer[10]);
        System.out.println("El array de enteros es: " + Arrays.toString(enteros));
}
```

i. ¿Cuál sería la salida en la consola?
ii. ¿Cuántos elementos tiene cada colección?
iii. ¿Qué instrucciones añadiría para que el objeto ***enteros*** tuviese a los enteros del 1 al 10 ordenados ascendentemente y mostrarlos en la consola?
iv. ¿Qué instrucciones añadiría para ampliar el objeto ***enteros*** a un array con los enteros del 1 al 11 ordenados ascendentemente?

5. A la vista del siguiente trozo de código, indique qué salida se obtiene en la consola tras su ejecución. Para ello, repase la semántica de las operaciones addAll, retainAll y removeAll para el tipo List.

```
List<Integer> numeros = new ArrayList<Integer>();
List<Integer> numeros2 = new ArrayList<Integer>();

numeros.add(1); numeros.add(2); numeros.add(3); numeros.add(4);
numeros.add(5); numeros.add(6); numeros.add(7); numeros.add(8);
numeros2.add(1); numeros2.add(2); numeros2.add(3); numeros2.add(4);

numeros2.addAll(numeros);
System.out.println(numeros2);

numeros2.retainAll(numeros);
System.out.println(numeros2);

numeros2.removeAll(numeros);
System.out.println(numeros2);
```

6. Suponiendo que en el código del ejercicio anterior cambiamos las líneas 1 y 2 por las dos líneas siguientes, indique qué salida se obtiene en la consola tras su ejecución. Para ello, repase la semántica de las operaciones addAll, retainAll y removeAll para el tipo Set. ¿Qué observa que cambia con respecto al ejercicio anterior?

```
Set<Integer> numeros = new HashSet<Integer>();
Set<Integer> numeros2 = new HashSet<Integer>();
```

7. A la vista del siguiente trozo de código, indique qué salida se obtendrá en la consola tras su ejecución. Para ello, repase la semántica de las operaciones específicas del tipo List.

```
List<Integer> numeros = new ArrayList<Integer>();
List<Integer> numeros2 = new ArrayList<Integer>();

numeros.add(1); numeros.add(0,2); numeros.add(0,3);
Collections.addAll(numeros2, 1,2,3);
numeros.addAll(0,numeros2);
System.out.println(numeros);
System.out.println("Primer 1 en posición " + numeros.indexOf(1));
System.out.println("Último 1 en posición " + numeros.lastIndexOf(1));

numeros.remove(numeros.indexOf(3));
numeros.set(2, numeros.get(2)+1);
System.out.println(numeros);

numeros.add(2,3);
numeros.add(4,3);
System.out.println(numeros);
```

8. El siguiente método estático pretende eliminar todos los números de la lista de entrada desde la primera aparición de numero1 hasta la primera aparición de numero2, ambos inclusive. Indique cómo debe completar el código para conseguir el efecto deseado. Recuerde que el método subList del tipo List devuelve una vista sobre la lista original.

```
public static void eliminaElementos(List<Integer> numeros,Integer numero1,Integer numero2){
        List<Integer> vistaNumeros = numeros.subList(__________________,__________________);
        ____________________________________;
}
```

9. Para facilitar la planificación de reuniones entre dos participantes, se desea disponer de los siguientes métodos estáticos que reciben dos conjuntos de LocalDate que representan las fechas en las que cada participante está disponible.
   i. calculaFechasDisponiblesEnAlguno: devuelve las fechas disponibles en alguno de los dos conjuntos de entrada.
   ii. calculaFechasCompatibles: devuelve las fechas disponibles al mismo tiempo en ambos conjuntos de entrada.
   iii. calculaFechasDisponiblesSoloEnUno: devuelve las fechas disponibles en un conjunto pero no en el otro.

   Implemente los tres métodos y pruébelos en un método main. Para ello, cree los dos siguientes conjuntos de fechas disponibles, y muestre por consola el resultado de invocar sobre ellos cada uno de los métodos:

```
Set<LocalDate> agenda1 = new HashSet<LocalDate>();
Set<LocalDate> agenda2 = new HashSet<LocalDate>();
Collections.addAll(agenda1, LocalDate.of(2017, 2, 1), LocalDate.of(2017, 2, 2),
        LocalDate.of(2017, 2, 3), LocalDate.of(2017, 2, 5), LocalDate.of(2017, 2, 6),
        LocalDate.of(2017, 2, 7), LocalDate.of(2017, 2, 10));
Collections.addAll(agenda2, LocalDate.of(2017, 2, 1), LocalDate.of(2017, 2, 3),
        LocalDate.of(2017, 2, 4), LocalDate.of(2017, 2, 9), LocalDate.of(2017, 2, 10),
        LocalDate.of(2017, 2, 11), LocalDate.of(2017, 2, 12));
```

   ¿Qué cambiaría en la implementación de los métodos anteriores para que las fechas mostradas en consola aparezcan ordenadas cronológicamente? Reflexione sobre **el mínimo** de cambios necesarios para conseguirlo.

10. Se desea implementar los siguientes métodos estáticos, que reciben una lista (List<T>) y dos enteros indicando una posición inicial y una posición final.
    i. ordenaTrozo: devuelve una nueva lista igual a la lista recibida pero en la que el trozo indicado (desde posición inicial hasta posición final, ambas incluidas) haya sido ordenado de mayor a menor según la relación de orden del tipo de los objetos de la lista.
    ii. minimoEnTrozo: devuelve el menor elemento de la lista, de entre los ubicados entre posición inicial y posición final, ambas inclusive.
    iii. maximoEnTrozo: devuelve el mayor elemento de la lista, de entre los ubicados entre posición inicial y posición final, ambas inclusive.

    Implemente los tres métodos y pruébelos en un método main. NOTA: incluya en la cabecera de los métodos, justo detrás de la palabra reservada static, la siguiente definición: <T extends Comparable<? super T>>[1].

[1] Esta definición indica que el tipo de los elementos de la lista de entrada debe ser Comparable. En caso contrario, no se podría llevar a cabo la funcionalidad pedida en el ejercicio.

11. Los métodos implementados en el ejericio anterior pueden lanzar excepciones de tipo IndexOutOfBoundsException, si las posiciones recibidas exceden el rango de posiciones válidas de la lista de entrada. Modifique los métodos para que se capture si se produce posible excepción, y en dicho caso devuelvan null.

12. Implemente un método estático que reciba un array de String y lo muestre por pantalla con los elementos ordenados alfabéticamente. Tenga en cuenta que el método no debe modificar el array que recibe como parámetro.

13. Se pide al alumno que realice los siguientes pasos en tres clases denominadas TestTipoX donde X tomará los valores Array, List o Set, en función del tipo que se esté testeando. Para cada uno de ellos, realice los siguientes pasos:
    i. Defina una variable del tipo X para contener los elementos “Hola”, “Mundo” y “Cruel”.
    ii. Muestre por pantalla la variable en cuestión. Recuerde que si se trata de un tipo Array, debe apoyarse en el método Arrays.toString para poder visualizarlo correctamente.
    iii. Añada una nueva cadena “Hola” al objeto en la variable.
    iv. Vuelva a mostrar por pantalla la variable.
    v. Finalmente, muestre por pantalla el primer elemento dentro del objeto.

    En base al resultado obtenido, conteste si las siguientes cuestiones resultan ser ciertas o falsas:

    i. Los arrays son tipos de objetos que guardan información de manera dinámica.
    ii. Los arrays son objetos que implementan la interfaz Collection.
    iii. Existe posibilidad de acceder a cada posición de un array.
    iv. Los objetos de tipo List son indexados (permiten acceso a cada posición) y pueden tener elementos repetidos.
    v. Los objetos de tipo Set no permiten elementos repetidos.
    vi. Los objetos de tipo Set no permiten el acceso a cada una de las posiciones aunque su organización interna respeta siempre el orden de entrada de los elementos.

14. En este ejercicio implementaremos métodos estáticos relacionados con distintas utilidades matemáticas. Todos ellos han de ser resueltos utilizando bucles, e implementándose dos versiones, una mediante un bucle tipo for y otra mediante un bucle tipo while. Para ello, cree una clase de utilidad denominada Matematicas y añada los siguientes métodos:
    a. Un método que reciba un número entero y decida si es primo. Tenga en cuenta que un número será primo si no existe ningún número entre 2 y su raíz cuadrada que lo divida.
    b. Haciendo uso del método anterior, implemente un método que reciba un número entero y devuelva una lista con aquellos números primos que sean menores o igual al parámetro.
    c. Un método denominado *potencia* que reciba un parámetro de tipo real y otro de tipo entero. Devolverá el resultado de elevar el primer parámetro (base) al segundo (exponente), multiplicando la base tantas veces como diga el exponente.
    d. Un método que reciba un número entero y devuelva el valor de su factorial (de tipo Long).
    e. Un método que reciba un número entero y decida si es perfecto. Un número se dice perfecto si es igual a la suma de sus divisores exceptuando él mismo. Por ejemplo, el número 6 es perfecto ya que 6 = 1 + 2 + 3.

15. Queremos realizar un programa que permita al usuario acertar un número elegido al azar. Para ello implementaremos varias estrategias. Implemente los siguientes métodos haciendo uso de bucles tipo for y/o while. En primer lugar, crearemos una clase de utilidad denominada Acertijo e incluiremos en ella las siguientes definiciones de constantes:

```
private static final Integer MINIMO_ALEAT = 100;
private static final Integer MAXIMO_ALEAT = 150;
private static final Integer MAX_INTENTOS = 5;
```

a) Implemente un método que genere un número entero aleatorio en el intervalo [*MINIMO_ALEAT, MAXIMO_ALEAT*] e interaccione con el usuario repetidamente hasta que éste lo averigüe. El proceso comenzará leyendo un número entero desde teclado y se le informará al usuario si el valor a averiguar es mayor o menor hasta que lo acierte.

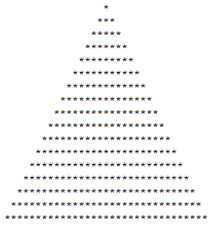
b) Implemente una nueva versión del método anterior que añada un máximo número de intentos, basado en la constante *MAX_INTENTOS*. El método procederá igual que en el caso anterior, salvo que en esta ocasión habrá dos alternativas de finalización: o bien porque el usuario adivine el número en menos de *MAX_INTENTOS*, o bien porque se han alcanzado todos los intentos permitidos.

Nota: Para generar un número aleatorio utilice las siguientes instrucciones, y adecúelas sabiendo que el método nextInt genera un número entero aleatorio en el intervalo [0,MAX).

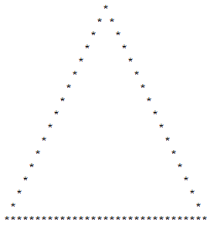
```
Random r = new Random();
int aleat = r.nextInt(MAX);
```

16. Queremos implementar distintas formas de imprimir una pirámide por consola. Para ello, cree una clase de utilidad denominada Piramides y añada los siguientes métodos:

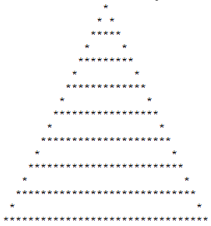
    a. Un método *imprimePiramide*, que reciba un entero con el valor de una altura y muestre por consola el siguiente resultado:



    b. Un método *imprimePiramideHueca*, que reciba un entero con el valor de una altura y muestre por consola el siguiente resultado:



    c. Un método *imprimePiramideARayas*, que reciba un entero con el valor de una altura y muestre por consola el siguiente resultado:



    Se recomienda seguir estas indicaciones:

    - La siguiente expresión nos permite obtener la base de la pirámide, que coincide con la anchura total de cada una de las líneas de la misma:

- anchura = (altura - 1) * 2 + 1

- La expresión anterior también proporciona el número de puntos o asteriscos que se imprimirán en cada nivel (en el caso de la pirámide del apartado a):
  - numPuntos = (nivel - 1) * 2 + 1
- El número de espacios que precederán a el primer asterisco de cada nivel se obtendrán con la expresión:
  - espacios = (anchura - numPuntos) / 2
- Para mayor claridad, se recomienda hacer uso de un método privado imprimeLineaPiramide, que reciba la anchura (o base de la pirámide) y el número de puntos a imprimir.

17. Escriba un método principal (***main***) que lea por teclado una cantidad cualquiera de números enteros separados por coma e invoque a un método estático para obtener la media aritmética de todos ellos y mostrar el resultado en la consola. El tipo String proporciona un método llamado ***split*** que permite obtener un array con subcadenas o *"trozos"* de la cadena que lo invoca. Asi por ejemplo, mediante las dos instrucciones siguientes obtendríamos un array de siete objetos String, con subcadenas o *"partes"* de la cadena que invoca a ***split***:

```
String palabrasReservadas ="public,class,interface,extends,implements,super,this";
String[] arrayDePalabras = palabrasReservadas.split(",");
//arrayDePalabras {"public", "class", "interface", "extends", "implements","super", "this"}
```

Recuerde que el tipo Scanner proporciona el método ***nextLine*** para leer una cadena de texto por teclado y que puede obtener un objeto Integer a partir de un objeto String si este último representa un literal válido invocando a uno de sus dos constructores mediante el operador new. Por ejemplo:

```
String[] array = {"2016", "2017"};
Integer a = new Integer(array[0]); // crearía el objeto Integer a con valor 2016
Integer b = new Integer(array[1]); // crearía el objeto Integer b con valor 2017
```

18. Escriba un método principal que pida al usuario un texto cualquiera y muestre en la consola, en líneas separadas:
    a. El número total de apariciones de letras mayúsculas en dicho texto (con repeticiones).
    b. El número total de apariciones de letras minúsculas en dicho texto (con repeticiones).
    c. El número total de apariciones de caracteres que no son letras (con repeticiones).
    d. El número total de apariciones de letras vocales en dicho texto (con repeticiones).
    e. El número total de apariciones de letras consonantes en dicho texto (con repeticiones).

    Utilice un método distinto a invocar desde el principal que dé respuesta a cada uno de los apartados anteriores tras leer una cadena introducida por teclado.

19. Escriba un método principal que pida al usuario un texto cualquiera y muestre en la consola en líneas separadas el número de apariciones de cada uno de los caracteres del mismo. Utilice tres métodos, dos de ellos a invocar desde el principal:
    a. Uno para obtener un conjunto ordenado con todos los caracteres de una cadena.
    b. Otro para obtener la frecuencia de un carácter en una cadena.
    c. Otro para mostrar los resultados en la consola.

20. Modifique el ejercicio anterior para que solamente tenga en cuenta los caracteres que son letras y considere como uno mismo sus apariciones tanto en mayúscula como en minúscula.

21. Modifique el ejercicio 18 anterior de forma que el método principal pida al usuario un texto cualquiera y muestre en la consola, en líneas separadas:

a. El número de mayúsculas distintas en dicho texto (sin repeticiones).
b. El número de minúsculas distintas en dicho texto (sin repeticiones).
c. El número de caracteres distintos que no son letras en dicho texto (sin repeticiones).
d. El número de vocales distintas en dicho texto (sin repeticiones).
e. El número de consonantes distintas en dicho texto (sin repeticiones).

Utilice un método distinto a invocar desde el principal que dé respuesta a cada uno de los apartados anteriores así como otro para leer el texto por teclado. Apóyese en el tipo Set para obtener la solución.

22. Escriba un programa que pida al usuario dos enteros positivos a y b tal que a<=b y calcule los años bisiestos entre ambos.

23. Escriba la clase TestFibonacci que pida al usuario un valor máximo y haga uso de un método de utilidades que genere una lista con todos los números de la secuencia de Fibonacci que sean menores que dicho valor máximo. Finalmente, muestre por pantalla la lista.

24. Escriba el algoritmo de Euclides para calcular el máximo común divisor (m.c.d.) en una función de utilidad. El algoritmo de Euclides es un procedimiento para calcular el m.c.d. de dos números. Los pasos son:
    a) Se divide el número mayor entre el menor.
    b) Si la división es exacta, el divisor es el m.c.d.
    c) Si la división no es exacta, volvemos a dividimos el divisor entre el resto obtenido y se continúa de esta forma hasta obtener una división exacta, siendo el último divisor el m.c.d.

    Tenga en cuenta que m.c.d(a, 0) = a para todo a. Compruebe en la clase TestEuclides que su solución es correcta con los siguientes casos de prueba.

Dividendo	Divisor	Resultado	Excepción
2	3	1	-
-2	100	2	-
100	-2	2	-
0	4	4	-
6	0	6	-

25. Realice las siguientes pruebas en una clase denominada TestForExtendido:
    a) Defina una variable del tipo Collection para contener los elementos “Hola”, “Mundo” y “Cruel”. Muestre por pantalla la variable en cuestión mediante un **for extendido**. Ejecute el código.
    b) Repita el método anterior pero cambie el tipo de la variable y defínala como un array de String.
    c) Cambie de nuevo el tipo de la variable y en su lugar, declárela como un String con el valor “Hola Mundo Cruel”. En el caso de que haya un error de compilación en el bucle, ¿cómo se podría resolver esta situación? Eche un ojo al API de Java y busque el método split de String...
    d) Repita la prueba del primer apartado pero en lugar de mostrarlos por pantalla, intente eliminarlos de la colección ¿qué ocurre?
    e) Repita el ejercicio anterior pero en lugar de trabajar con String, juegue con elementos del tipo Punto. Dentro del for extendido, antes de mostrarlos por pantalla, cambie la coordenada X al valor 1.0.

    En base al resultado obtenido, diga si las siguientes cuestiones resultan ser ciertas o falsas:

    a) El for extendido sólo se puede utilizar sobre objetos de tipo Collection.

b) El for extendido sólo se puede utilizar sobre arrays.
c) El for extendido se puede utilizar sobre objetos de tipo array y Collection.
d) Tanto la inicialización de arrays como la modificación de las colecciones de datos se pueden implementar mediante for extendido.
e) Los objetos contenidos en la estructura que se recorre mediante un for extendido nunca pueden ser modificados.

26. Vamos a implementar un juego en el que el usuario, a partir de una lista de palabras, intente adivinar cuál de ellas es la clave elegida. Para ello, crearemos una clase TestPasswordFinder e incluiremos en ella las siguientes definiciones de constantes:

```
private static final String[] PALABRAS = {"hola", "mundo", "cruel",
       "alumno", "profesor", "robot", "parlamentarismo", "jugador"};
private static final Integer MAX_INTENTOS = 10;
```

a) Para dar una pista al usuario vamos a definir una medida de similitud entre lo que introdujo y la clave. Para ello, necesitamos un método que transforme un String en una lista de caracteres. Impleméntelo usando un **for extendido**.

```
List<Character> convierteALista(String palabra)
```

b) Apoyándonos en este método, seremos capaces de dar una medida de similitud. Para ello, calcularemos el número de caracteres de una palabra x que está contenida en una clave. Para ello, implementaremos un nuevo método mediante un **for extendido** que devuelva el número de letras de la lista asociada a *palabra* que contiene la lista asociada a *clave*.

```
Integer calculaSimilitud(String palabra, String clave)
```

c) Finalmente, sólo nos queda implementar en el main el programa del juego. Para ello, seleccionaremos una palabra aleatoria como clave. Para ello use un objeto de tipo Random que devuelva un índice al azar (consulte qué método puede interesarnos en el API de Java 8). Después, realizaremos un bucle **for clásico** en el que en cada iteración hasta MAX_INTENTOS:

i. Se muestra la lista de palabras posibles
ii. Se le pide al usuario que escriba un intento
iii. Si la palabra no es válida (no está en la lista), se le lanza un mensaje de aviso de trampa y se pasa al punto i.
iv. Si la palabra es válida, se calcula la similitud entre la palabra y la clave y viceversa, se suman y se dividen entre la suma de los tamaños de ambas palabras. Dicho valor se muestra como un porcentaje.
v. Seguidamente, si la palabra es la clave, se aborta el bucle (use la palabra reservada **break**) y si no, se pasa al punto i.
vi. Finalmente, tanto si hubo éxito como si se superó el número de intentos, se muestra un mensaje por pantalla como final del juego.

NOTA PARA ALUMNOS AVANZADOS: piense en cómo definir un método de utilidad que lea las palabras del diccionario desde un fichero en el que cada palabra estará en una línea distinta. Se le sugiere usar un objeto de tipo Scanner sobre el fichero y utilizar los métodos hasNext y next para gestionar las lecturas mediante un **bucle while**. El prototipo de este método podría ser:

```
List<String> creaDiccionarioPalabras(String path)
```

27. Se desean realizar algunas modificaciones en los tipos del paquete música. En concreto, vamos a añadir

un nuevo tipo denominado **Album**, así como realizar algunos cambios en los tipos **Artista** y **Cancion**. A continuación se describen las propiedades y características de todos ellos (los cambios para los tipos que son modificados aparecen en negrita):

Album:

- Propiedades:
  - Id, de tipo String, consultable
  - Nombre, de tipo String, consultable.
  - Popularidad, de tipo Integer, consultable. La popularidad de un álbum es independiente de la de los artistas o las canciones del mismo.
  - Año de publicación, de tipo Integer, consultable.
  - Tipo del álbum, consultable. Podrá tener uno de los siguientes valores: ALBUM, SINGLE, COMPILATION.
  - Canciones, de tipo List<Cancion>, consultable. Es una lista de las canciones que forman el álbum.
  - Artistas, de tipo List<Artista>, consultable. Es una lista de los artistas que participan en el álbum.
  - Géneros, de tipo List<String>, consultable. Es una lista de los géneros musicales del álbum.
  - URL de imágenes, de tipo List<String>, consultable. Es una lista que contendrá las URLs de las imágenes relacionadas con el álbum.
- Constructores:
  - C1: recibe un parámetro por cada una de las propiedades básicas del tipo.
- Restricciones:
  - R1: El id debe contener 22 caracteres.
  - R2: La popularidad debe estar comprendida entre 0 y 100, ambos inclusive.
  - R3: La url de todas las imágenes del álbum deben comenzar por “http”.
- Representación como cadena: el nombre del álbum y su identificador entre paréntesis. Ejemplo: “Chumi Chuma (4159oFYGeKWFk4DeeTNhN1)”
- Criterio de igualdad: Dos álbumes serán considerados iguales si tienen el mismo identificador.

Artista:

- Propiedades:
  - Id, de tipo String, consultable
  - Nombre, de tipo String, consultable.
  - Popularidad, de tipo Integer, consultable y modificable.
  - **Géneros, de tipo List<String>**, consultable. Es una lista de los géneros musicales del artista.
  - **URL de imágenes, de tipo List<String>**, consultable y modificable. Es una lista que contendrá las URLs de las imágenes relacionadas con el artista.
- Constructores:
  - C1: recibe un parámetro por cada una de las propiedades básicas del tipo.
- Restricciónes:
  - R1: El id debe contener 22 caracteres.
  - R2: La popularidad debe estar comprendida entre 0 y 100, ambos inclusive.
  - R3: La url de todas las imágenes del artista deben comenzar por “http”.

Se pide:

a) Modifique la interfaz Artista y la clase ArtistaImpl para que se ajusten a las nuevas características de las propiedades y sus restricciones.
b) Modifique la clase TestArtista de forma conveniente.

Cancion:

- Propiedades:
  - Id, de tipo String, consultable
  - Nombre, de tipo String, consultable.
  - Duración, de tipo Duration, consultable.
  - Número de pista, de tipo Integer, consultable.
  - Popularidad, de tipo Integer, consultable y modificable.
  - **Artistas, de tipo List<Artista>**, consultable. Es una lista con los artistas que participan en la canción.
  - **URL de preescucha, de tipo String**, consultable. Contiene la URL a un MP3 con 30 segundos de la canción.
- Constructores:
  - C1: recibe un parámetro por cada una de las propiedades básicas del tipo.
- Restricciones:
  - R5: La URL de preescucha debe comenzar por "http".
- Representación como cadena: el nombre de la canción y todos los artistas que participan en ella entre corchetes. Ejemplo: "Loca [Ernesto Snajer, Silvia Pérez Cruz]"

Se pide:

a) Modifique las interfaces Cancion y Artista y las clases CancionImpl y ArtistaImpl para que se ajusten a las nuevas características descritas.
b) Modifique las clases TestCancion y TestArtista de forma conveniente.
c) Implemente la interfaz Album y la clase AlbumImpl.
d) Implemente una clase TestAlbum que incluya un método main en el que se cree un álbum y se muestre por la consola tanto la representación como cadena del mismo, como cada una de las propiedades por separado.

28. Vamos a implementar un nuevo tipo en el paquete fp.musica para trabajar con listas de reproducción, según la siguiente definición:

ListaReproduccion:

- Propiedades:
  - Nombre, de tipo String, consultable y modificable.
  - Canciones, de tipo List<Cancion>, consultable.
  - Número de canciones, de tipo Integer, consultable. Se calcula a partir de la propiedad anterior.
- Otras operaciones:
  - void aleatoriza(): cambia aleatoriamente el orden de las canciones en la lista. Utilice el método estático *shuffle* de la clase Collections.
  - void incorpora(Cancion c): añade la canción al final de la lista de reproducción.
  - void incorpora(Album a): añade todas las canciones del álbum al final de la lista de reproducción.
  - void incorpora(List<Cancion> canciones): añade todas las canciones de la lista al final de la lista de reproducción.
  - void eliminaPrimera(Cancion c): elimina la primera aparición de la canción en la lista de reproducción. Si la canción no pertenece a la lista, se lanza IllegalArgumentException.
  - void eliminaUltima(Cancion c): elimina la última aparición de la canción en la lista de reproducción. Si la canción no pertenece a la lista, se lanza IllegalArgumentException.
  - void eliminaTrozo(int primeraPosicion, int ultimaPosicion): elimina todas las canciones de la lista que van desde la posición *primeraPosicion* hasta la posición *ultimaPosicion*, ambas inclusive. Debe cumplirse que primeraPosicion sea mayor o igual que cero, ultimaPosicion sea menor que el número de canciones en la lista, y primeraPosicion sea menor o igual que ultimaPosicion; en caso contrario se lanza IllegalArgumentException.

- Constructores:
  - C1: recibe un parámetro para indicar el nombre de la lista. Al crearse, la lista de reproducción no contendrá canciones.
- Representación como cadena: el nombre de la lista, seguido del número de canciones entre paréntesis. Por ejemplo: "Música tranquila (142 canciones)".
- Criterio de igualdad: dos listas son iguales si sus listas de canciones son iguales (mismas canciones y en el mismo orden)[2].

Se pide:

a) Implemente la interfaz ListaReproduccion y la clase ListaReproduccionImpl.
b) Añada una clase UtilesMusica al paquete fp.musica. Implemente en dicha clase un método funcional con el siguiente prototipo:

```
public static ListaReproduccion generaListaReproduccion (String nombreArtista, int maxCanciones, ClienteSpotify cliente)
```

El método devuelve una lista de reproducción con, como máximo, maxCanciones del artista especificado. Utilice el método getArtistas de ClienteSpotify[3] para obtener una lista de artistas para el nombre especificado. Después utilice el método getAlbumesArtista de ClienteSpotify para obtener los álbumes del artista que ocupa la primera posición de la lista anterior. Este último método recibe el id del artista para el que se quieren obtener los álbumes. Añada todas las canciones de todos los álbumes obtenidos a una lista de reproducción, aleatorice el orden de las canciones y finalmente recorte el trozo necesario para cumplir con la especificación del método.

c) Añada una clase TestListaReproduccion al paquete fp.musica.test. En el método main de dicha clase, cree dos listas de reproducción usando el método generaListaReproduccion, para los dos artistas que desee, cada una con 10 canciones. Cree una nueva lista de reproducción y añádale las canciones de las dos listas anteriores. Finalmente, aleatorice esta última lista y muestre las canciones que la forman por pantalla.

29. Aplique los siguientes cambios a los tipos MiembroStaff y Pelicula definidos en boletines anteriores:

MiembroStaff:
- Nuevas propiedades:
  - ***Alias***, de tipo List<String>, consultable y modificable. Sustituye a la propiedad anterior "alias", de forma que ahora un objeto de este tipo puede tener más de un alias asociado.
- Constructores: realice los cambios adecuados en aquellos constructores en los que sea necesario según las nuevas propiedades en lugar de las antiguas.

Pelicula:
- Nuevas propiedades:
  - ***Generos***, de tipo List<String>, consultable y modificable. Sustituye a la propiedad anterior "genero", de forma que ahora un objeto de este tipo puede tener varios géneros asociados en vez de uno.
  - ***Productoras***, de tipo List<String>, consultable y modificable. Sustituye a la propiedad anterior "productora", de forma que ahora un objeto de este tipo puede tener varias productoras asociadas en vez de una.

[2] Esto coincide con el criterio de igualdad del tipo List
[3] Para poder utilizar el tipo ClienteSpotify, descargue e incorpore a su proyecto la librería SpotifyFP desde la carpeta Laboratorio > Librerías en Enseñanza Virtual.

- ***Paises***, de tipo List<String>, consultable y modificable. Sustituye a la propiedad anterior "pais", de forma que ahora un objeto de este tipo puede tener varios paises asociados en vez de uno.
- ***Directores***, de tipo List<MiembroStaff>, consultable y modificable. Sustituye a la propiedad anterior "director", de forma que ahora un objeto de este tipo puede tener varios directores asociados en vez de uno.
- ***Actores***, de tipo List<MiembroStaff>, consultable y modificable. Propiedad nueva.
- ***Guionistas***, de tipo List<MiembroStaff>, consultable y modificable. Propiedad nueva.

■ Constructores: realice los cambios adecuados en aquellos constructores en los que sea necesario según las nuevas propiedades en lugar de las antiguas.

Se pide:

a) Modifique las interfaces MiembroStaff y Pelicula y las clases MiembroStaffImpl y PeliculaImpl, partiendo de las versiones anteriores y realizando los cambios solicitados.
b) Implemente una clase TestPelicula que incluya un método main en el que se cree una película, se muestre por la consola la representación como cadena de la misma y a continuación cada una de sus propiedades en diferentes líneas, así como la representación como cadena y cada una de las propiedades de los directores, actores y guionistas de la misma.

30. Se desea efectuar algunos cambios al tipo Vuelo. A continuación se detallan sólo los cambios con respecto a las definiciones de estos tipos contenidas en los boletines de ejercicios anteriores.

Vuelo:

■ Nuevas propiedades:
- **Pasajeros**, de tipo Set<Persona>, consultable. Contiene el conjunto de las personas que irán en un vuelo.
- **NumeroPasajeros**, de tipo Integer, consultable. Su valor dependerá del conjunto de pasajeros.

■ Constructores: realice los cambios adecuados a los constructores para que tengan en cuenta, en su caso, los valores para las nuevas propiedades básicas en lugar de las antiguas. El constructor que antes recibía el número de pasajeros, ahora recibe el conjunto de pasajeros. El constructor que antes inicializaba el número de pasajeros a 0, ahora inicializa el conjunto de pasajeros con un conjunto vacío.

■ Otras operaciones (se implementarán como nuevas propiedades del tipo, en ningún caso son métodos funcionales):
- **void nuevoPasajero(Persona p)**, añade una persona al vuelo o lanzaIllegalArgumentException si el vuelo ya está completo. Si la persona tiene el valor null la operación no tiene efectos.
- **void eliminaPasajero(Persona p)**, elimina a la persona que se pasa como parámetro como pasajero del vuelo, si estaba en él. En otro caso, no hace nada.

Se pide:

a) Modifique la interfaz Vuelo y la clase VueloImpl, partiendo de las versiones anteriores y realizando los cambios solicitados.
b) Implemente una clase TestVuelo que incluya un método main en el que se cree un vuelo y se muestre por la consola tanto la representación como cadena del mismo, como cada una de las propiedades por separado.

31. A continuación se da la definición de un nuevo tipo de nombre Aeropuerto para el cual se pide su implementación (interfaz y clase):

Aeropuerto

■ Propiedades:

- ***Nombre***, de tipo String, consultable. Esta propiedad no puede ser nula.
- ***Ciudad***, de tipo String, consultable. Esta propiedad no puede ser nula.
- ***Vuelos***, de tipo SortedSet<Vuelo>, consultable.

- Constructores: un constructor para inicializar todas las propiedades básicas y otro que cree un objeto Aeropuerto sin vuelos.
- Criterio de igualdad: según el nombre.
- Orden natural: alfabéticamente por nombre.
- Representación como cadena: el nombre seguido de la ciudad entre paréntesis.
- Operaciones (se implementarán como nuevas propiedades del tipo, en ningún caso son métodos funcionales):
  - ***void nuevoVuelo (Vuelo v)***, añade un vuelo al conjunto de vuelos del aeropuerto. Si el vuelo tiene el valor *null* la operación no tiene efectos.
  - ***void nuevosVuelos(Collection<Vuelo> vuelos)***, añade una colección de vuelos no nulos al conjunto de vuelos del aeropuerto. Si colección tiene el valor *null* la operación no tiene efectos.
  - ***boolean contieneVuelo (Vuelo v)***, devuelve un valor lógico que indica si un vuelo pertenece al aeropuerto.
  - ***void eliminaVuelo (Vuelo v)***, elimina un vuelo del conjunto de vuelos del aeropuerto, siempre que el vuelo no sea null.
- Restricciones:
  - **R1**: Todos los vuelos de un aeropuerto deben tener su ciudad como origen o destino.

Se pide:

a) Implemente la interfaz Aeropuerto y la clase AeropuertoImpl.
b) Implemente una clase TestAeropuerto que incluya un método main en el que se cree un aeropuerto, se le añadan vuelos, y se muestre por la consola tanto la representación como cadena del mismo, como cada una de las propiedades por separado.

32. Para trabajar con los tipos relacionados con bibliotecas vamos a definir un nuevo tipo Biblioteca que permita relacionar otros tipos que ya hemos realizado en otros boletines (Libro, Persona, Prestamo). Dicho tipo viene definido como sigue:

Biblioteca

- Propiedades:
  - Nombre, de tipo String, consultable.
  - Direccion, de tipo String, consultable y modificable.
  - Codigo Postal, de tipo String, consultable y modificable.
  - Localidad, de tipo String, consultable.
  - Telefono, de tipo String, consultable y modificable.
  - Email, de tipo String, consultable y modificable.
  - Libros, de tipo List<Libro>, consultable.
  - Usuarios, de tipo Set<Persona>, consultable.
  - Prestamos, de tipo Set<Prestamo>, consultable.

- Constructores:
  - C1: Constructor con valores para nombre y localidad que inicializa a colecciones vacías las propiedades libros, usuarios y préstamos y el resto las deja a null.
- Representación como cadena:
  - Nombre de la biblioteca y localidad donde está entre paréntesis. Ejemplo: “Infanta Elena (Sevilla)”

- Criterio de igualdad: Dos bibliotecas serán iguales si tienen el mismo nombre y la misma localidad.
- Restricciones:
  - R1: El nombre y la localidad no pueden ser null.
  - R2: El email o es null o debe contener los caracteres '@' y '.'
- Otras operaciones:
  - void nuevoLibro(Libro l). Añade un nuevo libro, que se pasa como parámetro a la lista de libros de la biblioteca. Se admiten libros duplicados en la biblioteca. Si el libro tiene el valor null la operación no tiene efectos.
  - void eliminaLibro(Libro l). Se elimina el libro que se pasa como parámetro de la lista de libros. Si el libro no existe, se lanza IllegalArgumentException (IAE).
  - void nuevoUsuario(Persona p). Añade una nueva persona como usuaria de la biblioteca. Si la persona ya existe o tiene el valor null la operación no tiene efectos.
  - void eliminaUsuario(Persona p). Elimina la persona que se pasa como parámetro del conjunto de usuarios.
  - void nuevoPrestamo(Libro l, Persona p). Añade un nuevo préstamo al conjunto de préstamos. Se añade el prestamo con la fecha actual. El libro y la persona deben pertenecer a la biblioteca. Si no, se lanza IAE.

Se pide:

a) Definir la interfaz Biblioteca y la clase BibliotecaImpl.
b) Implemente una clase TestBiblioteca que incluya un método main en el que se cree una biblioteca y se muestre por la consola tanto la representación como cadena de la misma, como cada una de las propiedades por separado. Haga unos de las operaciones que añaden información al tipo (operaciones nuevo[…] yy elimina[...]), tratando de tener en cuenta todos los posibles casos.

33. Se trata de actualizar algunos de los tipos relacionados con las películas y series de televisión realizadas en el boletín 2. Para ello, cree los tipos nuevos Episodio y Temporada y realice las modificaciones necesarias al tipo SerieTV que se describen a continuación:

Episodio

- Propiedades:
  - Id, de tipo Integer, consultable. Identificador del episodio.
  - FechaEmision, de tipo LocalDate.
  - NumeroEpisodio, de tipo Integer, consultable.
  - Titulo, de tipo String, consultable.
  - Sinopsis, de tipo String, consultable.
  - actores, de tipo List<MiembroStaff>, consultable.
  - artistasInvitados, de tipo List<MiembroStaff>, consultable.
- Constructores:
  - C1: recibe un parámetro por cada propiedad básica del tipo.
- Restricciones:
  - R1: No se permiten valores nulos para las propiedades.
- Representación como cadena: El id seguido de un guión, el título y su fecha de emisión entre paréntesis, por ejemplo: "1111 – Winter is coming (18/9/2011)".
- Criterio de igualdad: Dos episodios serán iguales si tienen el mismo identificador.

Temporada

- Propiedades:
  - Id, de tipo Integer, consultable. Identificador del episodio.

- FechaEmision, de tipo LocalDate, consultable y derivada. Esta propiedad será el resultado de buscar la fecha de emisión del primer capítulo.
- NumeroTemporada, de tipo Integer, consultable. Número de la temporada.
- NumeroEpisodios, de tipo Integer, consultable. Devuelve el número de episodios que tiene una temporada.
- Episodios, de tipo List<Episodio>, consultable. Lista de los episodios de la temporada.
- urlPoster, de tipo String, consultable y modificable. Dirección web con el poster de la temporada, si hay poster, si no, es nula.

- Constructores:
  - C1: recibe un parámetro por cada propiedad básica, excepto la url del poster, que se inicializa a null.
- Restricciones:
  - R1: No se permiten valores nulos para la propiedad episodios.
- Representación como cadena: El id seguido de un guion, el número de temporada y entre paréntesis el número de episodios. Finalmente, la fecha de emisión entre paréntesis, por ejemplo: "1111 – 1 (13) (18/9/2011)".
- Criterio de igualdad: Dos episodios serán iguales si tienen el mismo identificador.

SerieTV

- Nuevas Propiedades:
  - FechaPrimeraEmision, de tipo LocalDate, consultable y derivada. Coincidirá con la fecha de la emisión de la primera temporada.
  - FechaUltimaEmision, de tipo LocalDate, consultable y derivada. Coincidirá con la fecha de la emisión del último capítulo de la última temporada.
  - Generos, de tipo List<String>, consultable y modificable. Lista de géneros de la serie.
  - CadenasTV, de tipo List<String>, consultable y modificable. Lista de cadenas que la emiten.
  - Temporadas, de tipo List<Temporada>, consultable y modificable. Representa la lista de temporadas de la serie.
  - NúmeroTemporadas, de tipo Integer, consultable. Devuelve el número de temporadas grabadas de la serie.
  - UrlPoster, de tipo String, consultable y modificable. Dirección web con el poster de la serie, si hay poster, si no, es nula.
  - Creadores, de tipo List<MiembroStaff>, consultable y modificable. Lista de personas que crearon la serie.
  - duracionEpisodio, de tipo Duration, consultable y modificable. Duración de un episodio de la serie.
  - PaisesOrigen, de tipo List<String>, consultable y modificable. Lista de países de origen de la serie.
  - Sinopsis, de tipo String, consultable y modificable. Sinopsis de la serie.
- Constructores:
  - Todos los constructores deben incluir un parámetro que contenga las temporadas de la serie, el resto de parámetros se inicizarán a null, excepto las colecciones, que se inicializarán con una lista vacía.
  - Actualice los constructores para que tomen las nuevas propiedades básicas en lugar de las antiguas y sustituya los parámetros para la propiedad NumeroTemporadas por los correspondientes a Temporadas cuando sea necesario.
  - Tenga en cuenta que se debe seguir chequeando las restricciones originales del tipo.

Se pide:

a) Implemente las interfaces y clases necesarias para los tipos descritos.

b) Actualice la clase TestSerieTV (o constrúyala si no la hizo en su momento) para que incluya en el método main las líneas necesarias para comprobar que el tipo SerieTV funciona correctamente.

34. Se desea implementar el tipo Cine, relacionado con el tipo Sesion implementado anteriormente. Para su implementación usaremos la siguiente definición de sus propiedades y características:

Cine:

- Propiedades:
  - Nombre, de tipo String, consultable.
  - Dirección, de tipo String, consultable
  - Número de salas, de tipo Integer, consultable.
  - Accesible, de tipo Boolean, consultable. Indica si el cine es accesible para minusválidos.
  - Sesiones, de tipo SortedSet, consultable. Es un conjunto ordenado con las sesiones que se proyectan en el cine.
- Otras operaciones:
  - void nuevaSesion(Sesion s): Añade una nueva sesión al cine. La nueva sesión debe cumplir las siguientes restricciones:
    - El cine no debe contener actualmente una sesión igual a la nueva a añadir.
    - El número de sala donde se proyecta la nueva sesión debe ser menor o igual que el número de salas que tenga el cine.
  - void eliminaSesion(Sesion s): Elimina la sesión del cine. Si la sesión no pertenecía al cine la operación no tiene efecto.
- Constructores:
  - C1: recibe un parámetro por cada una de las propiedades básicas, menos para las sesiones, que se inicializará como un conjunto vacío.
- Restricciones:
  - R1: El cine debe disponer de al menos una sala
- Representación como cadena: el nombre del cine, el número de salas y si es accesible o no.
  Ejemplos: “Avenida 5 cines (5) – No accesible”, “Cines Nervión Plaza (5) - accesible”
- Criterio de igualdad: Dos cines serán considerados iguales si tienen el mismo nombre y mismo número de salas.
- Criterio de ordenación: Un cine será considerado mayor a otro si tiene mayor número de salas. A igualdad de número de salas se compararán según el orden alfabético de sus nombres.

Se pide:

a) Implemente la interfaz Cine y la clase CineImpl.
b) Implemente una clase TestCine que incluya un método main en el que se cree un cine y se muestre por la consola tanto la representación como cadena del mismo, como cada una de las propiedades por separado. Añada y elimine de dicho cine varias sesiones usando las operaciones *nuevaSesion* y *eliminaSesion*, tratando de tener en cuenta todos los posibles casos.

35. A continuación se proporciona la definición de un nuevo tipo EquipoFutbol para el cual se pide su implementación (interfaz y clase):

EquipoFutbol:

- Propiedades:
  - ***Nombre***, de tipo String, consultable. Esta propiedad no puede ser nula.
  - ***Ciudad***, de tipo String, consultable. Esta propiedad no puede ser nula.
  - ***Fecha de Fundación***, de tipo LocalDate, consultable. Esta propiedad no puede ser nula ni posterior al día actual de la creación del objeto.

- ***Número de Socios***, de tipo Integer, consultable y modificable. Esta propiedad no puede ser nula ni menor a una constante positiva.
- ***Jugadores***, de tipo SortedSet<JugadorFutbol>, consultable.

- Constructores: un constructor para inicializar todas las propiedades básicas y otro que cree un objeto EquipoFutbol sin jugadores.
- Criterio de igualdad: dos equipos serán considerados iguales si tienen el mismo nombre y la misma fecha de fundación.
- Orden natural: un equipo será considerado mayor que otro si tiene su año de fundación posterior. A igualdad de año de fundación, se ordenarán alfabéticamente por el nombre.
- Representación como cadena: La ciudad seguida de " – " y a continuación el nombre.
- Operaciones (se implementarán como nuevas propiedades del tipo, en ningún caso son métodos funcionales):
  - ***void nuevoJugador (JugadorFutbol jugador)***, añade un jugador al conjunto de jugadores del equipo. Si el jugador tiene valor nulo, la operación no tiene efectos.
  - ***void eliminaJugador (JugadorFutbol jugador)***, elimina un jugador al conjunto de jugadores del equipo.

- Restricciones:
  - **R1**: La fecha de fundación no puede ser posterior a la actual.
  - **R2**: El número de socios debe ser mayor o igual a una constante positiva.

Se pide:

a) Implemente la interfaz EquipoFutbol y la clase EquipoFutbolImpl.
b) Implemente una clase TestEquipoFutbol que incluya un método main en el que se cree un equipo, se le añadan jugadores, y se muestre por la consola tanto la representación como cadena del mismo, como cada una de las propiedades por separado.

36. Escriba la clase Ejercicio36 con un método main en el que:
    I. Cree dos conjuntos vacíos de Integer, añada al primero 7, 3, 9, 1, 5, 3 y al segundo 1, 2, 3, 4 y los muestre por pantalla.
    II. Muestre el cardinal de cada uno de los conjuntos.
    III. Pregunte si uno de los conjuntos contiene un entero dado que está y uno que no está.
    IV. Añada un entero al primer conjunto, compruebe el valor devuelto por la operación dependiendo de si el entero estaba ya en el conjunto o no y muestre el conjunto resultante.
    V. Elimine un entero de uno de los conjuntos. Muestre el resultado.
    VI. Calcule la unión, la intersección y la diferencia de los conjuntos y muestre los resultados.

37. Escriba la clase Ejercicio37 con un método main en el que:
    I. Cree dos listas vacías de Character, añada a la primera 'S', 'E', 'M', 'A', 'N', 'A' y a la segunda 'R', 'A', 'M', 'O' y las muestre por pantalla.
    II. Muestre el cardinal de cada una de las listas.
    III. Pregunte si una de las listas contiene una letra dada.
    IV. Pregunte en qué posición está una letra que sí está en una lista y otra que no está.
    V. Añada al final de la primera lista 'S', compruebe el valor devuelto por la operación y muestre la lista resultante.
    VI. Obtenga la sublista definida por dos posiciones, que incluya el último elemento.
    VII. Elimine todos los elementos de una lista entre dos posiciones dadas y muestre el resultado.
    VIII. Ordene la lista según el orden natural y muéstrela

IX. Calcule la unión, la intersección y la diferencia de las listas dadas y muestre los resultados. Observe qué diferencia hay con estas operaciones para conjuntos.
X. Invierta la lista y muéstrela.

38. En una clase de utilidad llamada UtilesArrays escriba el código de los siguientes métodos:
```
public static Integer [] sumaArrays (Integer []a, Integer  []b);
```

Devuelve un nuevo array que contiene la suma de los elementos de los arrays a y b. Si los arrays a y b no tienen la misma longitud, se elevará la excepción IllegalArgumentException.

39. En la clase de utilidad UtilesArrays escriba el código del siguiente método:
```
public static Integer [] crearArrayAleatorios
                      (Integer numElementos, Integer cotaInf, Integer cotaSup)
```

Devuelve un nuevo array de tamaño numElementos relleno con números enteros aleatorios en el intervalor [cotaInf, cotaSup). Si el número de elementos es negativo o la cota Inferior es mayor o igual a la inferior, se elevará la excepción *IllegalArgumentException*.

40. En la clase de utilidad UtilesString escriba el código del siguiente método:

```
public static Boolean esPalindromo (String cadena)
```

Devuelve cierto si la cadena es un palíndromo, es decir, si se lee igual de izquierda a derecha que al revés. Si la cadena tiene espacios en blanco, éstos no se tendrán en cuenta para ver si es un palíndromo. Si la cadena es nula se elevará la excepción IllegalArgumentException. Pruebe el método con las cadenas " oso", "radar", "la ruta natural ", "abba", "programación" y "si" (todas son palíndromos, excepto las dos últimas).