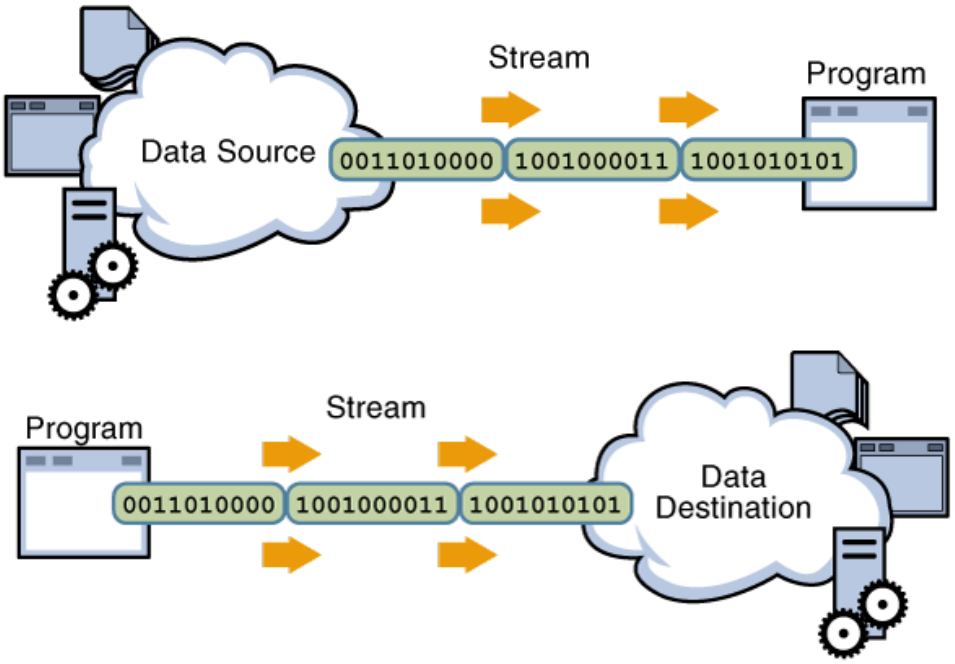
Ejercicios de la UD06



1. Ejercicios

1.1. Paquete: `UD06._1.gestorVuelos`

Se desea realizar una aplicación `GestorVuelos` para gestionar la reserva y cancelación de vuelos en una agencia de viajes. Dicha agencia trabaja únicamente con la compañía aérea Iberia, que ofrece vuelos desde/hacia varias ciudades de Europa. Se deben definir las clases que siguen, teniendo en cuenta que sus atributos serán privados y sus métodos sólo los que se indican en cada clase.

1. Implementación de la clase `Vuelo`, que permite representar un vuelo mediante los atributos:
    - `identificador` (`String`)
    - `origen` (`String`)
    - `destino` (`String`)
    - `hSalida` (un tipo que te permita controlar la hora, no es un `String` ni un `int`, etc.)
    - `hLlegada` (un tipo que te permita controlar la hora, no es un `String` ni un `int`, etc.)
    - Además, cada vuelo dispone de 50 asientos, es decir, pueden viajar, como mucho, 50 pasajeros en cada vuelo. Para representarlos, se hará uso de `asiento`, un array de `String` (nombres de los pasajeros) junto con un atributo `numP` que indique el número actual de asientos reservados. Si el asiento `i` está reservado, `asiento[i]` contendrá el nombre del pasajero que lo ha reservado. Si no lo está, `asiento[i]` será `null`. En el array `asiento`, las posiciones impares pertenecen a asientos de ventanilla y las posiciones pares, a asientos de pasillo (la posición 0 no se utilizará).

   En esta clase, se deben implementar los siguientes métodos:
    - `public Vuelo(String id, String orig, String dest, LocalTime hsal, LocalTime hlleg)`. **Constructor** que crea un vuelo con identificador, ciudad de origen, ciudad de destino, hora de salida y hora de llegada indicados en los respectivos parámetros, y sin pasajeros.
    - `public String getIdenificador()`. Devuelve el `identificador`
    - `public String getOrigen()`. Devuelve `origen`.
    - `public String getDestino()`. Devuelve `destino`.
    - `public boolean hayLibres()`. Devuelve `true` si quedan asientos libres y `false` si no quedan.
    - `public boolean equals(Object o)`. Dos vuelos son iguales si tienen el mismo identificador.
    - `public int reservarAsiento(String pas, char pref) throws VueloCompletoException`. Si el vuelo ya está completo se lanza una excepción. Si no está completo, se reserva al pasajero `pas` el primer asiento libre en `pref`. El carácter `pref` será '`P`' o '`V`' en función de que el pasajero desee un asiento de pasillo o de ventanilla. En caso de que no quede ningún asiento libre en la preferencia indicada (`pref`), se reservará el primer asiento libre de la otra preferencia. El método devolverá el número de asiento que se le ha reservado. Este método hace uso del método privado `asientoLibre`, que se explica a continuación.
    - `private int asientoLibre(char pref)`. Dado un tipo de asiento `pref` (pasillo '`P`' o ventanilla '`V`'), devuelve el primer asiento libre (el de menor numero) que encuentre de ese tipo. O devuelve `0` si no quedan asientos libres de tipo `pref`.
    - `public void cancelarReserva(int numasiento)`. Se cancela la reserva del asiento `numasiento`.
    - `public String toString()`. Devuelve una `String` con los datos del vuelo y los nombres de los pasajeros, con el siguiente formato:

      ```
      IB101 Valencia París 19:05:00 21:00:00
      Pasajeros:
      Asiento 1: Sonia Dominguez
      …
      Asiento 23: Fernando Romero
      ```

2. Diseñar e implementar una clase Java `TestVuelo` que permita probar la clase `Vuelo` y sus métodos. Para ello se desarrollará el método `main` en el que:

- Se cree el vuelo IB101 de Valencia a París, que sale a las 19:05 y llega a las 21:00
- Reservar:
    - Un asiento de ventanilla a "Miguel Fernández"
    - Un asiento de ventanilla a "Ana Folgado"
    - Un asiento de pasillo a "David Más"
- Mostrar el vuelo por pantalla
- Cancelar la reserva del asiento que indique el usuario.
- Mostrar de nuevo el vuelo por pantalla

3. Implementación de la clase `Compania` para representar todos los vuelos de una compañía aérea. Una Compañía tiene un nombre y puede ofrecer, como mucho, 10 vuelos distintos. Para representarlos se utilizará `listaVuelos`, un array de objetos `Vuelo` junto con un atributo `numVuelos` que indique el número de vuelos que la compañía ofrece en un momento dado. Las operaciones de esta clase son:
    - `public Compania(String n) throws FileNotFoundException`. Constructor de una compañía de nombre `n`. Cuando se crea una compañía, se invoca al método privado `leeVuelos()` para cargar la información de vuelos desde un fichero. Si el fichero no existe, se propaga la excepción `FileNotFoundException`
    - `private void leeVuelos() throws FileNotFoundException`. Lee desde un fichero toda la información de los vuelos que ofrece la compañía y los va almacenando en el array de vuelos `listaVuelos`. El nombre del fichero coincide con el nombre de la compañía y tiene extensión `.txt`. La información de cada vuelo se estructura en el fichero como sigue:

      ```
      <Identificador>
      <Origen>
      <Destino>
      <Hora de salida>
      <Minuto de salida>
      <Hora de llegada>
      <Minuto de llegada>
      ...
      ...
      ```

      Si el fichero no existe, se propaga la excepción `FileNotFoundException`.
    - `public Vuelo buscarVuelo(String id) throws ElementoNoEncontradoException`. Dado un identificador de vuelo `id`, busca dicho vuelo en el array de vuelos `listaVuelos`. Si lo encuentra, lo devuelve. Si no, lanza `ElementoNoEncontradoException`.
    - `public void mostrarVuelosIncompletos(String o, String d)`. Muestra por pantalla los vuelos con origen `o` y destino `d`, y que tengan asientos libres. Por ejemplo, vuelos con asientos libres de la compañía Iberia con origen Milán y destino Valencia:

      ```
      Iberia IB201 Milán Valencia 14:25:00 16:20:00
      Iberia IB202 Mílán Valencia 21:40:00 23:35:00
      ```

4. En la clase `GestorVuelos` se probará el comportamiento de las clases anteriores. En esta clase se debe implementar el método `main` en el que, por simplificar, se pide únicamente:
    - la creación de la compañía aérea `Iberia`. Se dispone de un fichero de texto "`Iberia.txt`", con la información de los vuelos que ofrece.
    - Reserva de un asiento de ventanilla en un vuelo de Valencia a París por parte de Manuel Soler Roca. Para ello:
        - Mostraremos vuelos con origen Valencia y destino París, que no estén completos.
        - Pediremos al usuario el identificador del vuelo en que quiere hacer la reserva.
        - Buscaremos el vuelo que tiene el identificador indicado. Si existe realizaremos la reserva y mostraremos un mensaje por pantalla. En caso contrario mostraremos un mensaje de error por pantalla.

1.2. Paquete: `UD06._2.maquinaExpendedora`

Se desea simular el funcionamiento de una máquina expendedora. Se trata de una expendedora sencilla que, por el momento, será capaz de dispensar únicamente un producto.

Su funcionamiento, a grandes rasgos, es el siguiente:

1. El cliente introduce dinero en la máquina. Al dinero introducido lo llamaremos `credito`.
2. Selecciona el producto que quiere comprar (ya hemos comentado que por el momento habrá un solo producto).
3. Si hay stock del producto seleccionado, la máquina dispensa el artículo elegido y devuelve el importe sobrante (diferencia entre el crédito introducido y el precio del producto).

Durante el proceso se pueden producir diversas incidencias, como por ejemplo, que el cliente no haya introducido suficiente crédito para comprar el producto, que no quede producto o que no haya cambio suficiente para la devolución. La máquina también da la posibilidad de solicitar la devolución del crédito sin realizar la compra.

1. Diseñar la clase `Expendedora` (proyecto `Expendedora`) con los atributos y métodos que se describen a continuación.
    - Atributos (privados)
        - `credito`: Cantidad de dinero (en euros) introducida por el cliente.
        - `stock`: Número de unidades que quedan en la máquina disponibles para la venta. Se reducirá con cada nueva venta.
        - `precio`: Precio del único artículo que dispensa la máquina (en euros).
        - `cambio`: Cambio del que dispone la máquina. El cambio disponible se reduce cada vez que se devuelve al cliente la diferencia entre el crédito introducido y el precio del producto comprado. El cambio nunca se ve incrementado por las compras de los clientes.
        - `recaudación`: Representa la suma de las ventas realizadas por la máquina (en euros). Se ve incrementada con cada nueva compra.
    - Métodos:
        - Constructor: `public Expendedora (double cambio, int stock, double precio)`. Crea la expendedora inicializando los atributos cambio, stock y precio con los valores indicados en los parámetros). El crédito y la recaudación serán cero.
        - Consultores:
            - Métodos consultores para los atributos crédito, cambio, y recaudación
            - Los consultores para el stock y el precio los haremos previendo que en el futuro la máquina pueda expender más de un tipo de producto. Para consultar el stock y el precio se indicará como parámetro el número de producto que se quiere consultar aunque, por el momento se ignorará el valor de dicho atributo.
            - `public getStock (int producto)` Devuelve el stock disponible del producto indicado. En esta versión simplificada se devolverá el valor del atributo stock, sea cual sea el valor de producto.
            - `public getPrecio (int producto)` Devuelve el precio del producto indicado. En esta versión simplificada se devolverá el valor del atributo precio, sea cual sea el valor de producto.
        - Modificadores: Para simplificar, consideramos que los atributos de la máquina solo van a cambiar por operaciones derivadas de su funcionamiento, por lo que no proporcionamos modificadores públicos
    - Otros métodos:
        - `public String toString()` Devuelve un `String` de la forma:

          ```
          Credito: 3.00 euros
          Cambio: 12.73 euros
          Stock: 12 unidades
          Recaudación: 127.87 euros
          ```

        - `public void introducirDinero(double importe)` Representa la operación mediante la cual el cliente añade dinero (crédito) a la máquina. Esta operación incrementa el crédito introducido por el cliente en el importe indicado como parámetro.
        - `public double solicitarDevolucion()` Representa la operación mediante la cual el cliente solicita la devolución del crédito introducido sin realizar la compra. El método devuelve la cantidad de dinero que se devuelve al cliente.

- `public double comprarProducto(int producto) throws NoHayCambioException, NoHayProductoException, CreditoInsuficienteException`. Representa la operación mediante la cual el cliente selecciona un producto para su compra. El método devuelve la cantidad de dinero que se devuelve al cliente.

  Si no se produce ninguna situación inesperada, se reduce el stock del producto, se devuelve el cambio, se pone el crédito a cero y se incrementa la recaudación.

  Si la venta no es posible se lanzará la excepción correspondiente a la situación que impide completar la venta.

2. La clase `Producto` permite representar uno de los artículos de los que vende una máquina expendedora. Para ello utilizaremos tres atributos privados `nombre` (`String`), `precio` (`double`) y `stock` (`int`), y los siguientes métodos:
   - `public Producto(String nombre, double precio, int stock)` Constructor que inicializa el producto con los parámetros indicados
   - Consultores de los tres atributos: `getNombre`, `getPrecio` y `getStock`
   - `public int decrementarStock()`: Decrementa en 1 el stock del producto y devuelve el stock resultante.
3. La clase `TestExpendedora` sirve para provar los métodos desarrollados en las clases `Expendedora` y `Producto`.
   - Crea un Objeto de tipo `Expendedora` e inicializalo con: 12 unidades de stock, 5 euros de cambio y un precio de 3.75 euros. Muestra por pantalla su estado actual.
   - Simula la introducción por parte del cliente de un billete de 5 euros y muestra el estado de la máquina `Expendedora`.
   - Simula la compra de un `producto` y muestra la cantidad devuelta.
   - Simula la introducción de una moneda de 2 euros y solicita la devolución sin realizar ninguna compra y muestra la cantidad devuelta.
   - Intenta realizar una compra sin tener suficiente crédito y gestiona la excepción.
   - Crea otro objeto de tipo `Expendedora` que inicialmente tenga 0 unidades de stock (el resto de valores a tu gusto), simula la compra de un producto teniendo suficiente crédito y cambio. Gestiona la excepción.
   - Crea un último objeto de tipo `Expendedora` que inicialmente tenga 0 euros de cambio (el resto de valores a tu gusto), simula la compra de un producto para el que la máquina tenga que devolver algún importe, gestiona la excepción.
   - Muestra las recaudaciones para las 3 máquinas expendedoras.
4. La clase `Surtido` representa una colección de productos. Para ello se usará un atributo `listaProductos`, array de `Productos`. El array se rellenará con los datos de productos extraidos de un fichero de texto y, una vez creado el surtido no será posible añadir o quitar productos. Así, el array de productos estará siempre completo y no es necesario ningún atributo que indique cuantos productos hay en el array.

   Se implementarán los siguientes métodos:
   - `public Surtido() throws FileNotFoundException` Crea el surtido con los datos de los productos que se encuentran en el fichero `productos.txt`. El fichero tiene el siguiente formato:

     ```
     <nº de productos>
     <nombre de producto> <precio> <stock>
     <nombre de producto> <precio> <stock>
     <nombre de producto> <precio> <stock>
     ...
     ```

     Como vemos, la primera línea del fichero indica el número de productos que contiene el surtido. Este dato lo usaremos para dar al array de productos el tamaño adecuado.
   - `public int numProductos()` Devuelve el número de productos que componen el surtido
   - `public Producto getProducto(int numProducto)`: Devuelve el producto que ocupa la posición `numProducto` del surtido. La primera posición válida es la `1`. La posición `0` no se utiliza.
   - `public String[] getNombresProductos()` Devuelve un array con los nombres de los productos. La posición `0` del array no se utilizará (será `null`)

5. Crea una copia de la clase `Expendedora` y llámala `ExpendedoraSurtido`. Añadir los atributos y hacer los cambios necesarios en la clase para que sea capaz de dispensar varios productos usando la nueva clase `Surtido`. Por ejemplo ya no tienen sentido los atributos stock y precio ya que pertenecen al `Surtido`. Añade también el método `public String toStringSurtido()`, que muestre por pantalla el listado de productos con su nombre, precio y stock para mostrar al cliente que productos puede elegir. El código del producto coincidirá con su posición al leer el surtido.
6. Crea una copia de la clase `TestExpendedora` y nómbrala `TestExpendedora2` para adaptarla a los cambios hechos en la clase `Expendedora` y usando la nueva posibilidad de comprar diferentes productos y usando solamente un único objeto `Expendedora`. Al final en lugar de mostrar la recaudación de las 3 máquinas expendedoras, muestra solo la de la única que hay y muestra el surtido.

2. Los flujos estándar

1. (clase `LeeNombre`) Escribir un programa que solicite al usuario su nombre y, utilizando directamente `System.in`, lo lea de teclado y muestre por pantalla un mensaje del estilo "Su nombre es Miguel". Recuerda que `System.in` es un objeto de tipo `InputStream`. La clase `InputStream` permite **leer bytes** utilizando el método `read()`. Será tarea nuestra ir construyendo un `String` a partir de los bytes leídos. Prueba el programa de manera que el usuario incluya en su nombre algún carácter “extraño”, por ejemplo el símbolo "€" ¿Funciona bien el programa? ¿Por qué?
2. (clase `LeeEdad`) Escribir un programa que solicite al usuario su edad y, utilizando directamente `System.in`, la lea de teclado y muestre por pantalla un mensaje del estilo "Su edad es 32 años". En este caso, será tarea nuestra construir un String a partir de los bytes leídos y transformarlo posteriormente en un entero.
3. (clase `CambiarEstandar`). La salida estándar (`System.out`) y la salida de errores (`System.err`) están asociadas por defecto con la pantalla. Se puede cambiar este comportamiento por defecto utilizando los métodos `System.setout` y `System.seterr` respectivamente. Investiga un poco cómo se utilizan, escribe un programa que asocie la salida estándar a al fichero `salida.txt` y la salida de errores al fichero `errores.txt` y, a continuación, escriba algún mensaje en cada uno de las salidas, por ejemplo `System.out.println(“El resultado es 20”);` y `System.err.println(“ERROR: Elemento no encontrado”);`
4. (`SumarEdades`)
    - Escribir método `void sumaEdades()` que lea de teclado las edades de una serie de personas y muestre cuanto suman. El método finalizará cuando el usuario introduzca una edad negativa.
    - Escribir un método `main` que llame al método anterior para probarlo.
    - Modificar el método `main` de forma que, antes de llamar al método `sumaEdades`, se cambie la entrada estándar para que tome los datos del fichero `edades.txt` en lugar de leerlos de teclado.

3. `InputStreamReader`

5. (leerByte) `System.in` (`InputStream`) está orientado a lectura de bytes. Escribe un programa que lea un byte de teclado y muestre su valor (int) por pantalla. Pruébalo con un carácter "extraño", por ejemplo '€'.
6. (leerCaracter) `InputStreamReader` (`StreamReader`) está orientado a carácteres. Escribe un programa que lea un carácter de teclado usando un `InputStreamReader` y muestre su valor (`int`) por pantalla. Pruébalo con un carácter "extraño", por ejemplo '€'. ¿Se obtiene el mismo resultado que en el ejercicio anterior?.

4. Entrada "orientada a líneas".

En los ejercicios anteriores, las limitaciones de la clase utilizada (`InputStream`), nos obliga a incluir en el programa instrucciones que detecten que el usuario ha terminado su entrada (ha pulsado **INTRO**). La clase `BuffereReader` dispone del método `readLine()`, capaz de leer una línea completa (la propia instrucción detecta el final de la línea) y devolver un `String`.

7.- Repite el ejercicio 1 utilizando un `BufferedReader` asociado a la entrada estándar. La clase `BufferedReader`, está orientada a leer caracteres en lugar de bytes. ¿Qué ocurre ahora si el usuario introduce un carácter "extraño" en su nombre?

8.- Repite el ejercicio 2 utilizando un `BufferedReader` asociado a la entrada estándar.

5. Lectura/escritura en ficheros

9. (EscribirFichero1) Escribe un programa que, usando las clases `FileOutputStream` y `FileInputStream`, (puedes usar más a parte de estas dos)
    - escriba tu nombre y altura en un fichero (`nombre.txt`).
    - lea el fichero creado y lo muestre por pantalla.
    - Si abrimos el fichero creado con un editor de textos, ¿su contenido es legible?
10. (EscribirFichero2) Repetir el ejercicio anterior utilizando las clases `FileReader` y `FileWriter`.

6. Uso de buffers

Los buffers hacen que las operaciones de lectura-escritura se realicen inicialmente en memoria y, cuando los buffers correspondientes están vacíos/llenos, se hagan definitivamente sobre el dispositivo.

11. (TestVelocidadBuffer) Vamos a probar la diferencia de tiempo que conlleva escribir datos a un fichero directamente o hacerlo a través de un buffer. Para ello, crea un fichero de 1 Mb (1000000 de bytes aprox.) usando un la clase `FileWriter` y mide el tiempo que tarda en crearlo. Posteriormente, crea un fichero de exactamente el mismo tamaño utilizando un `BufferedWriter` y mide el tiempo que tarda. ¿Hay diferencia?. Para medir el tiempo puedes utilizar `System.currentTimeMillis()`, inmediatamente antes y después de crear el fichero y restar los valores obtenidos.
12. (TestVelocidadBuffer2) Modifica el programa `TestVelocidadBuffer` para probar cómo afecta a la escritura con buffer la ejecución de la instrucción `flush()`. Esta instrucción fuerza el volcado del buffer a disco. ¿Disminuye la velocidad si tras cada operación de escritura ejecutamos flush()?
13. (TestVelocidadBuffer3) Modifica el programa `TestVelocidadBuffer` para probar cómo a la velocidad el tamaño del buffer. La clase `BufferedWriter` tiene un constructor que permite indicar el tamaño del buffer. Prueba con distintos valores.

7. Streams para información binaria

14. (Personas) Escribe un programa que, utilizando entre otras la clase `DataOutputStream`, almacene en un fichero llamado `personas.dat` la información relativa a una serie de personas que va introduciendo el usuario desde teclado:
    - `Nombre` (String)
    - `Edad` (entero)
    - `Peso` (double)
    - `Estatura` (double)

    La entrada del usuario terminará cuando se introduzca un nombre vacío.

    Nota: Utiliza la clase `Scanner` para leer desde teclado y los métodos `writeDouble`, `writeInt` y `writeUTF` de la clase `DataOutput/InputStream` para escribir en el fichero)

    Al finalizar el programa, abre el fichero resultante con un editor de texto (notepad o wordpad) ¿La información que contiene es legible?.
15. (AñadirPersonas) Modifica el programa anterior para que el usuario, al comienzo del programa, pueda elegir si quiere añadir datos al fichero o sobre escribir la información que contiene.
16. (MostrarPersonas) Realizar un programa que lea la información del fichero `personas.dat` y la muestre por pantalla. Para determinar que no quedan más datos en el fichero podemos capturar la excepción `EOFException`
17. (CalculosPersonas) Realizar un programa, similar al anterior, que lea la información del fichero `personas.dat` y muestre por pantalla la estatura que tienen de media las personas cuya edad está entre 20 y 30 años.

8. Streams de objetos. Serialización.

18. Paquete `GuardaLeeLibros`
    - (`Autor`) Crea la clase `Autor`, con los atributos **nombre**, **año de nacimiento** y **nacionalidad**. Incorpora un **constructor** que reciba todos los datos y el método `toString()`.
    - (`Libro`) Crea la clase `Libro`, con los atributos **titulo**, **año de edición** y **autor** (Objeto de la clase `Autor`). Incorpora un **constructor** que reciba todos los datos y el método `toString()`.
    - Escribe un programa `GuardaLibros` que cree tres libros y los almacene en el fichero `biblioteca.obj`.
    - Las clases deberán implementar el interfaz `Serializable`.
19. Paquete `GuardaLeeLibros`
    - Escribe un programa (`LeeLibros`) que lea los objetos del fichero `biblioteca.obj` y los muestre por pantalla (Libros y Autores).

9. Sockets

20. Programar un Servidor que reciba una fecha (previamente validada por el cliente) y nos diga cual es nuestro signo del zodíaco occidental y el animal que corresponde en el zodíaco oriental (animales).

10. Más ejercicios (Lionel)

Importante Para probar algunos de estos ejercicios debes utilizar el archivo `Documentos.zip`. Descárgalo del aula virtual y descomprímelo en la carpeta de cada proyecto que crees.

21. **Mostrar información de ficheros**

Implementa un programa que pida al usuario introducir por teclado una ruta del sistema de archivos (por ejemplo, `C:/Windows` o `Documentos`) y muestre información sobre dicha ruta (ver función más abajo). El proceso se repetirá una y otra vez hasta que el usuario introduzca una ruta vacía (tecla intro). Deberá manejar las posibles excepciones.

Necesitarás crear la función `void muestraInfoRuta(File ruta)` que dada una ruta de tipo `File` haga lo siguiente:

- Si es un archivo, mostrará por pantalla el nombre del archivo.
- Si es un directorio, mostrará por pantalla la lista de directorios y archivos que contiene (sus nombres). Deberá mostrar primero los directorios y luego los archivos.
- En cualquier caso, añade delante del nombre la etiqueta `[*]` o `[A]` para indicar si es un directorio o un archivo respectivamente.
- Si el path no existe lanzará un `FileNotFoundException`.

22. **Mostrar información de ficheros (v2)**

Partiendo de una copia del programa anterior, modifica la función `muestraInfoRuta`:

- En el caso de un directorio, mostrará la lista de directorios y archivos en orden alfabético. Es decir, primero los directorios en orden alfabético y luego los archivos en orden alfabético.
- Añade un segundo argumento `boolean info` que cuando sea `true` mostrará, junto a la información de cada directorio o archivo, su tamaño en bytes y la fecha de la última modificación. Cuando `info` sea `false` mostrará la información como en el ejercicio anterior.

23. **Renombrando directorios y ficheros**

Implementa un programa que haga lo siguiente:

- Cambiar el nombre de la carpeta `Documentos` a `DOCS`, el de la carpeta `Fotografias` a `FOTOS` y el de la carpeta `Libros` a `LECTURAS`
- Cambiar el nombre de todos los archivos de las carpetas `FOTOS` y `LECTURAS` quitándole la extensión. Por ejemplo, `astronauta.jpg` pasará a llamarse `astronauta`.

24. **Creando (y moviendo) carpetas**

Implementa un programa que cree, dentro de `Documentos`, dos nuevas carpetas: `Mis Cosas` y `Alfabeto`. Mueve las carpetas `Fotografias` y `Libros` dentro de `Mis Cosas`. Luego crea dentro de `Alfabeto` una carpeta por cada letra del alfabeto (en mayúsculas): `A`, `B`, `C`... `Z`. Te serán de ayuda los códigos numéricos ASCII: https://elcodigoascii.com.ar

25. **Borrando archivos**

Implementa un programa con una función `boolean borraTodo(File f)` que borre `f`: Si no existe lanzará una excepción. Si es un archivo lo borrará. Si es un directorio, borrará primero sus archivos y luego el propio directorio (recuerda que para poder borrar un directorio debe estar vacío). Devolverá `true` si pudo borrar el `File f` (`false` si no fué posible).

Prueba la función borrando las carpetas: `Documentos/Fotografias`, `Documentos/Libros` y `Documentos` (es decir, tres llamadas a la función, en ese orden).

Super extra challenge: Esta función, tal y como está definida, no borrará las subcarpetas que estén dentro de una carpeta (para ello habría que borrar primero el contenido de dichas subcarpetas). ¿Se te ocurre cómo podría hacerse?

26. **Máximo y mínimo**

Implementa un programa que muestre por pantalla los valores máximos y mínimos del archivo `numeros.txt`.

27. **Notas de alumnos**

El archivo `alumnos_notas.txt` contiene una lista de 10 alumnos y las notas que han obtenido en cada asignatura. El número de asignaturas de cada alumno es variable. Implementa un programa que muestre por pantalla la nota media de cada alumno junto a su nombre y apellido, ordenado por nota media de mayor a menor.

28. **Ordenando archivos**

Implementa un programa que pida al usuario un nombre de archivo `A` para lectura y otro nombre de archivo `B` para escritura. Leerá el contenido del archivo `A` (por ejemplo `usa_personas.txt`) y lo escribirá ordenado alfabéticamente en B (por ejemplo `usa_personas_sorted.txt`).

29. **Nombre y apellidos**

Implementa un programa que genere aleatoriamente nombres de persona (combinando nombres y apellidos de `usa_nombres.txt` y `usa_apellidos.txt`). Se le pedirá al usuario cuántos nombres de persona desea generar y a qué archivo **añadirlos** (por ejemplo `usa_personas.txt`).

30. **Diccionario**

    Implementa un programa que cree la carpeta `Diccionario` con tantos archivos como letras del abecedario (`A.txt`, `B.txt`... `Z.txt`). Introducirá en cada archivo las palabras de `diccionario.txt` que comiencen por dicha letra.

31. **Búsqueda en PI**

    Implementa un programa que pida al usuario un número de cualquier longitud, como por ejemplo "1234", y le diga al usuario si dicho número aparece en el primer millón de decimales del nº pi (están en el archivo `pi-million.txt`). No está permitido utilizar ninguna librería ni clase ni método que realice la búsqueda. Debes implementar el algoritmo de búsqueda tú.

32. **Estadísticas**

    Implementa un programa que lea un documento de texto y muestre por pantalla algunos datos estadísticos: nº de líneas, nº de palabras, nº de caracteres y cuáles son las 10 palabras más comunes (y cuántas veces aparecen). Prueba el programa con los archivos de la carpeta `Libros`.

    NOTA: *Para llevar la cuenta de cuántas veces aparece cada palabra puedes utilizar una HashTable. Una tabla hash es una estructura de datos tipo colección (como el ArrayList), que permite almacenar pares clave-valor. Por ejemplo {"elefante", 5} o {"casa", 10} son pares <String,Integer> que asocian una palabra (clave) con un nº entero (valor).*

11. Aún más ejercicios

1. (`CuentaLineas`) Escribe un programa que, sin utilizar la clase `Scanner` para contar las lineas, muestre el número de lineas que contiene un fichero de texto. El nombre del fichero se solicitará al usuario al comienzo de la ejecución (para esto si puedes usar la clase `Scanner`).
2. (`CuentaPalabras`) Escribe un programa que, sin utilizar la clase `Scanner` para contar las lineas, muestre el número de palabras que contiene un fichero de texto. El nombre del fichero se solicitará al usuario al comienzo de la ejecución (para esto si puedes usar la clase `Scanner`).

   > **Importante** Lee el fichero, línea a línea y utiliza la clase `StringTokenizer` o bien el método `split` de la clase `String` para averiguar el nº de palabras.

3. (`Censura`) Escribir un programa que sustituya por otras, ciertas palabras de un fichero de texto. Para ello, se desarrollará y llamará al método `void aplicaCensura(String entrada, String censura, String salida),` que lee de un fichero de entrada y mediante un fichero de censura, crea el correspondiente fichero modificado. Por ejemplo:

   Fichero de entrada:

   ```
   En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo
   que vivía un hidalgo de los de lanza en astillero
   ```

   Fichero de censura:

   ```
   lugar sitio
   quiero debo
   hidalgo noble
   ```

   Fichero de salida:

   ```
   En un sitio de la Mancha, de cuyo nombre no debo acordarme, no ha mucho tiempo que
   vivía un noble de los de lanza en astillero
   ```

   > **Importante** Valora la posibilidad de cargar el fichero de censura en un mapa o par clave, valor.

4. (`Concatenar1`) Escribe un programa que dados dos ficheros de texto `f1` y `f2` confeccione un tercer fichero `f3` cuyo contenido sea el de `f1` y a continuación el de `f2`.
5. (`Concatenar2`) Escribe un programa que dados dos ficheros de texto `f1` y `f2`, añada al final de `f1` el contenido de `f2`. Es decir, como el ejercicio anterior, pero sin producir un nuevo fichero.
6. (`Iguales`) Escribir un programa que compruebe si el contenido de dos ficheros es idéntico. Puesto que no sabemos de qué tipo de ficheros se trata, (de texto, binarios, ...) habrá que hacer una comparación byte por byte.
7. Escribe los siguientes métodos y programa:
   - Método `void generar()` que genere 20 números aleatorios enteros entre 1 y 100 y los muestre por pantalla.
   - Método `void media()` que lea de teclado 20 números enteros y calcule su media.
   - Programa que, modificando la entrada y la salida estándar, llame a `generar()` para que los datos se graben en un fichero y a continuación llame a `media()` de manera que se tomen los datos del fichero generado.
8. (`Notas`) Escribir un programa que almacene en un fichero binario (`notas.dat`) las notas de 20 alumnos. El programa tendrá el siguiente funcionamiento:
   - En el fichero se guardarán como máximo 20 notas, pero se pueden guardar menos. El proceso de introducción de notas (y en consecuencia, el programa) finalizará cuando el usuario introduzca una nota no válida (menor que cero o mayor que 10).
   - Si, al comenzar la ejecución, el fichero ya contiene notas, se indicará al usuario cuántas faltan por añadir y las notas que introduzca el usuario se añadirán a continuación de las que hay.

- Si, al comenzar la ejecución, el fichero ya contiene 20 notas, se le preguntará al usuario si desea sobrescribirlas. En caso afirmativo las notas que introduzca sustituirán a las que hay y en caso negativo el fichero no se modificará.

12. 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