Unidad 1. MANEJO DE FICHEROS

Estructuras de datos en Java:

java.lang.StringBuilder
java.util.StringTokenizer
java.util.Vector <E>
java.util.ArrayList <E>
java.util.Hashtable <K, V>
java.util.HashMap <K, V>

1. Crear una clase con un método main para hacer lo siguiente:

- Crear un StringBuilder con el String ”Vuenos dias” y mostrarlo por consola.
- Mostrar por consola su capacidad y longitud.
- Sustituir el primer carácter del StringBuilder anterior por ‘B’ y mostrarlo por consola.
- Agregar al final del StringBuilder anterior el numero entero 20 y el String “chicos”.
- Insertar en el StringBuilder anterior después de “dias” el String “tengan”. Mostrar por consola.
- Eliminar los seis primeros caracteres del StringBuilder y mostrar por consola.
- Almacenar en un String los cuatro últimos caracteres del StringBuilder y mostrarlo.
- Mostrar por consola la capacidad y la longitud del StringBuilder final.
- Crear un array de tres StringBuilder con vuestro nombre, primer apellido y segundo apellido.
- Visualizar el nombre escrito de derecha a izquierda.

2. Crear una clase con un método main para hacer lo siguiente:

- Solicitar al usuario una frase con varias palabras
- Mostrar el número de palabras de la frase.
- Mostrar cada palabra en una línea diferente.
- Almacenar en un array de StringBuilder las palabras de la frase.
- Mostrar los elementos del array anterior y la clase a la que pertenecen mediante los métodos:
  Class getClass() de java.lang Object y
  String getName() de java.lang.Class

3. Crear una clase con un método main para hacer lo siguiente:

- Crear un Vector que contenga los nombres de los planetas del sistema solar en orden alfabético (Jupiter, Marte, Mercurio, Neptuno, Pluton, Saturno, Tierra, Urano, Venus)
- Mostrar la capacidad y tamaño originales.
- Comprobar si el Vector contiene la cadena “Saturno”. Si la contiene, mostrar su posición.
- Mostrar por consola el primer y último elemento del Vector.
- Insertar “Logroño” justo después del elemento “Tierra”. Mostrar el Vector, su capacidad y su tamaño.
- Eliminar todos los elementos del Vector.
- Mostrar por consola la capacidad y el tamaño del Vector final.

NOTA: Si los planetas no se agregan al Vector por orden alfabético, pueden ordenarse mediante el método estático void sort(List lista) de la clase java.util.Collections.
El argumento del método debe ser una objeto de la interface List (equivale a una clase que implemente la interface List). Como la clase Vector implementa por defecto dicha interface es consistente.
Muy usado para ordenar vectores que almacenan cadenas de texto.
Es similar al método sort(..) de la clase Arrays.

4. En el ejemplo anterior, sustituir la clase Vector por ArrayList de objetos String.

5. Crear una clase con un método main para saber si un usuario ha aprobado o suspendido una prueba. El funcionamiento será el siguiente: el programa solicitará una clave de usuario y

- Si no existe la clave, el programa debe mostrar "Clave incorrecta" y finalizar.
- Si existe, se mostrará su calificación y un mensaje de “Enhorabuena” si ha aprobado o un "Lo siento" si ha suspendido. Después el programa finalizará.

Utilizar un Hashtable con una capacidad de 10 y un factor de carga del 80%. Contendrá los pares clave/valor de la siguiente tabla (las claves serán objetos Integer, los valores Strings):

clave	*valor*
1505	aprobado
2800	suspenso
1300	suspenso
1234	aprobado
2600	aprobado

6. En el ejemplo anterior, sustituir la clase Hashtable por HashMap.

7. Conversión de moneda. Pedir por teclado la moneda origen, la moneda destino y el importe a convertir. Las equivalencias, referidas al euro, se guardan en un HashMap.

8. Medir la aleatoriedad del método random: Generar 10000 números entre 0 y 19 y contar las apariciones de cada uno.

9. Realizar una clase CuentaPalabras con un método main que haga lo siguiente:
- Leer un fichero de texto sin formato del sistema local que solo contenga palabras.
- Contar el número de veces que se repite cada palabra.
- Ordenar alfabéticamente las palabras.
- Escribir en otro fichero de nombre *contador.txt* la palabra y el número de veces que aparece.

PISTA: Una posible solución podría basarse en este algoritmo:
1. Crear un HashMap para almacenar como claves las palabras del fichero origen y como valores el número de veces que se repiten.
2. Crear flujo y filtro para leer el fichero origen. Cada línea del mismo es una String, por tanto, puede crearse una StringTokenizer usando el delimitador por defecto. Luego, recorrer sus tokens e intentar obtener en el HashMap la frecuencia de repetición de cada palabra teniendo en cuenta que si no hay ningún valor asociado a la palabra analizada devuelve null (esto implicará que la palabra es la primera vez que aparece). Si es distinto de null, la palabra ya ha aparecido antes. ¿Qué hacemos? Se elimina el elemento del cuya clave coincide con la palabra y se agrega un nuevo elemento teniendo en cuenta que el valor será un Integer, que almacenará un entero una unidad mayor que el del elemento eliminado.
3. Crear un ArrayList con las claves del HashMap y ordenarlas mediante el método estático void sort(..) de la clase java.util.Collections.
4. Crear flujo y filtro para escribir en el fichero destino los pares del HashMap por orden alfabético, en base a las claves. No olvidar vaciar y cerrar el filtro con flush() y close() respectivamente.

Gestión de Ficheros. Paquete java.io

OPERACIÓN	***FLUJO (stream)***	***BUFFER / DATA / OBJECT***	***MÉTODOS***
Lectura de Bytes	FileInputStream	BufferedInputStream	int read() void close()
		DataInputStream	boolean readBoolean() byte readByte() … double readDouble() String readUTF()
		ObjectInputStream	Object readObject()
Escritura de Bytes	FileOutputStream	BufferedOutputStream	void write(int n) void flush() void close()
		DataOutputStream	void writeBoolean(boolean v) void writeByte(int v) … void writeDouble(double v) void writeUTF(String s)
		ObjectOutputStream	void writeObject(Object o)
Lectura de Caracteres	FileReader	BufferedReader	int read() String readLine() void close()
Escritura de Caracteres	FileWriter	BufferedWriter	void write(String s) void newLine() void flush() void close()
Ficheros de acceso aleatorio	RandomAccessFile	modoAcceso: r: solo lectura rw: lectura y escritura	long getFilePointer() long length() void seek(long posicion) int skipBytes(int n) boolean readBoolean() byte readByte() char readChar() … double readDouble() void writeBoolean(boolean v) void writeByte(int v) … void writeDouble(double v) void writeChars(String s)

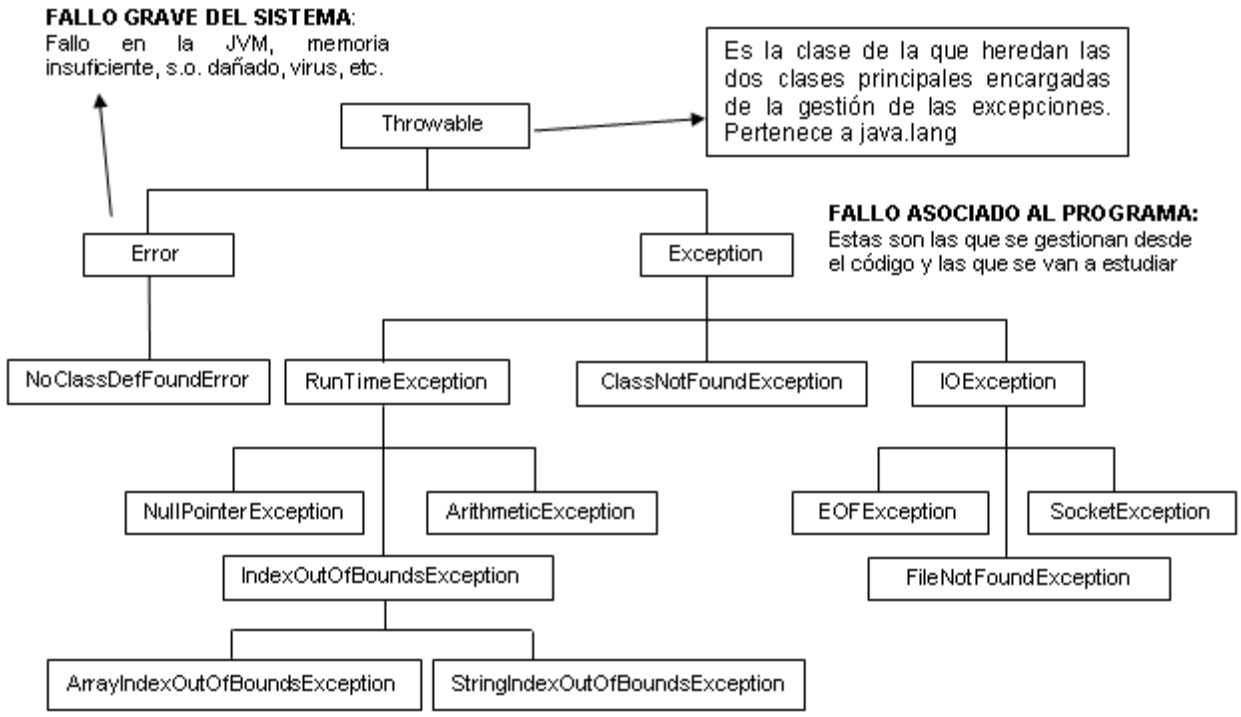
1. Crear un fichero de texto de varias líneas (*Origen.txt*). Utilizar los flujos y filtros de lectura y escritura de caracteres para crear un fichero copia del anterior.

2. Comparativa del tiempo de ejecución de un programa que lee un fichero de texto de 2 MB. Los métodos de lectura son: *a*) Carácter a carácter sin filtro, *b*) Carácter a carácter con filtro. *c*) Línea a línea

3. Mantenimiento del fichero de acceso aleatorio Empleados del apartado 1.6.3 según el menú:

- Listar todos los Empleados
- Consultar un Empleado
- Añadir un Empleado (al final)
- Modificar un Empleado

Excepciones



Causas de excepciones habituales:

- IOException: asociada a errores genéricos de entrada/salida de datos a un programa. De ella heredan otras excepciones. No suele producirse casi nunca. Es de gestión obligatoria.
- FileNotFoundException: asociada al hecho de no encontrar un fichero.
- EOFException: indica que se ha llegado al final de un fichero.
- NoClassDefFoundError: no se encuentra el .class de la clase principal. Es un error, no una excepción.
- ClassNotFoundException: carga incorrecta de una clase externa.
- ArithmeticException: engloba errores aritméticos: división por cero ...
- NullPointerException: se intenta acceder a una variable referenciada que no ha sido inicializada.
- ArrayIndexOutOfBoundsException: se produce al intentar acceder a un elemento de un array con un índice que es igual o mayor que su tamaño.
- StringIndexOutOfBoundsException: en un String, se intenta acceder a un carácter con un índice igual o mayor que su tamaño.
- NumberFormatException: se produce cuando se introduce un tipo de dato incorrecto en los argumentos de algunos métodos. Ejemplo: el método estático "int parseInt(String num)" espera recibir un número entero almacenado en forma de cadena. Hereda de IllegalArgumentException y ésta de RuntimeException.

Ejercicio

Un almacén de cafés tiene varias marcas de cafés. De cada marca ha comprado distintas partidas (lotes) en diferentes fechas y tenemos que considerar su fecha de caducidad. Las clases se definen así:

```
class Lote{
   double kilos;
   String caducidad;
   . . . . . .
}
```

```
class Cafe{
   String nombre;
   double precio;       // por kg
   double stock;        // stock total en kgs
   Vector<Lote> v;      // guarda los lotes del café
   . . . . . . . . .
}
```

Leer el fichero *MisCafes.txt* (ver ejemplo abajo) que contiene todas las marcas con sus lotes por orden creciente de caducidad (el primero es el más antiguo).
El programa visualiza el Control de Inventario tal como se muestra en el ejemplo.
También debe gestionar la Venta de Café: solicita la marca (que deberá estar en almacén), pide la cantidad a vender (que no debe ser mayor que el stock de la marca) y la suministra en orden cronológico de lotes, esto es, primero se toma del lote que va a caducar antes.

MisCafes.txt

```
COLOMBIA
8.95
10
22-09-2012
25.5
12-10-2012
0
KENIA
9.75
15.5
22-11-2012
0
JAVA
9.50
20
11-10-2012
100
11-11-2012
12.5
11-12-2012
0
MOKA
10.20
40
20-11-2012
50
12-12-2012
0
```

nombre
precio
kilos lote 1
caducidad lote 1
kilos lote 2
caducidad lote 2
fin de café
nombre
precio
. . . . . .

```
        ** INVENTARIO DE CAFES **
COLOMBIA a 8.95 euros/kg
        22-09-2012: 10.0 kg
        12-10-2012: 25.5 kg
  Stock Total: 35.5
KENIA a 9.75 euros/kg
        22-11-2012: 15.5 kg
  Stock Total: 15.5
JAVA a 9.5 euros/kg
        11-10-2012: 20.0 kg
        11-11-2012: 100.0 kg
        11-12-2012: 12.5 kg
  Stock Total: 132.5
MOKA a 10.2 euros/kg
        20-11-2012: 40.0 kg
        12-12-2012: 50.0 kg
  Stock Total: 90.0

        ** VENTA DE CAFE **
Marca pedida (F=fin): Jato
ERROR: marca desconocida.
Marca pedida (F=fin): JAVA
Kilos pedidos: 150
ERROR: stock insuficiente.
Marca pedida (F=fin): JAVA
Kilos pedidos: 50
... Vendidos 20.0 kg del lote de 11-10-2012
… Lote agotado.
... Vendidos 30.0 kg del lote de 11-11-2012

EXISTENCIAS FINALES:
JAVA a 9.5 euros/kg
        11-11-2012: 70.0 kg
        11-12-2012: 12.5 kg
  Stock Total: 82.5

Marca pedida (F=fin): F
```