Índice

Especificación Detallada

CtrlPresentacio

Descripción: Interficie grafica con la que el usuario interactua para elegir el algoritmo y denotar el fichero que se quiere comprimir/descomprimir y el archivo de destino.

Métodos

- `public void compress(int alg, String fileIn, String fileOut, short  qualityJPEG)`
  - **Descripción:** Da a elegir al usuario el algoritmo para la compresión y le pide el nombre del archivo a comprimir y el nombre del archivo comprimido que luego pasa al CtrlDomini.
  - **Return:** Es void por tanto no devuelve nada.
- `public void decompress(String fileIn, String fileOut)`
  - **Descripción:** Le pide al usuario el nombre del archivo comprimido y del archivo destino y se los pasa a CtrlDomini.
  - **Return:** Es void por tanto no devuelve nada.

CtrlDomini

Descripción: Controlador del dominio.

Atributos

- `private String fileIn`
  - **Descripción:** Directorio del archivo que se desea comprimir.
- `private String fileOut`
  - **Descripción:** Directorio del archivo en el que se desea guardar la compresión.
- `private Statistics stats`
  - **Descripción:** Objeto para crear y guardar la estadísticas generadas por la compresión.
- `private static CtrlDomini instance`
  - **Descripción:** Instancia del controlador Dominio (Singleton).

Métodos

- `public void compress(int alg, String fileIn, String fileOut, Short quality)`
  - **Descripción:** Dado un algoritmo, el nombre fichero de entrada y el nombre del fichero comprimido, ejecuta la compresión con el algoritmo pertienente.
  - **Párametros:**
    - alg: Algoritmo con el que comprimir.
    - fileIn: Nombre del archivo a comprimir.
    - fileOut: Nombre del archivo comprimido.
    - quality: Calidad de compresión para el JPEG (entre 1 y 100).
  - **Return:** Devuelve las estadísticas generadas para la compresión.
- `void compress(int alg, IO.Byte.reader input, IO.Bit.writer output, Short  quality)`
  - **Descripción:** Dado un algoritmo, el nombre fichero de entrada y el nombre del fichero comprimido, ejecuta la compresión con el algoritmo pertienente.
  - **Párametros:**
    - alg: Algoritmo con el que comprimir.
    - fileIn: Objeto de lectura del archivo a comprimir.
    - fileOut: Cbjeto de escritura del archivo comprimido.
    - quality: Calidad de compresión para el JPEG (entre 1 y 100).
  - **Return:** Devuelve las estadísticas generadas para la compresión.
- `public void decompress(String fileIn, String fileOut)`
  - **Descripción:** Dado un archivo comprimido y el nombre para el archivo descomprimido, descomprime el archivo usando el mismo algoritmo con el que se comprimió.
  - **Párametros:**
    - fileIn: Nombre del fichero comprimido.

    - fileOut: Nombre del fichero descomprimido.
  - **Return:** Devuelve las estadísticas de la descompresión.
- `void decompress(IO.Bit.reader input, IO.Byte.writer output, byte magicByte)`
  - **Descripción:** Dado un archivo comprimido y el nombre para el archivo descomprimido, descomprime el archivo usando el mismo algoritmo con el que se comprimió.
  - **Párametros:**
    - fileIn: Objeto de lectura del fichero comprimido.
    - fileOut: objeto de escritura del fichero descomprimido.
    - magicByte: Byte que identifica el algoritmo a utilizar.
  - **Return:** Devuelve las estadísticas de la descompresión.
- `private static String readableFileSize(double d)`
  - **Descripción:** Da el formato correcto al tamaño de un fichero (B,kB,etc).
  - **Párametros:**
    - d: Tamaño de un fichero en bytes.
  - **Return:** Tamaño del fichero en la magnitud que le corresponda.

LZSS

Descripción: Compresión y descompresión de archivos con LZSS

Atributos

- `final static int MAX_SIZE_SW`
  - **Descripción:** Tamaño máximo de la ventana corrediza.
- `final static int MAX_LENGTH_COINCIDENCE`
  - **Descripción:** Longitud máxima de una coincidencia.
- `public final static byte MAGIC_BYTE`
  - **Descripción:** Magic byte que identifica al compresor para saber con que algoritmo debemos descomprimir.
- `private final static int EOF`
  - **Descripción:** Int que indica el final del archivo.
- `final static byte[] slidingWindow = new byte[MAX_SIZE_SW]`
  - **Descripción:** Contiene como mucho los últimos MAX_SIZE_SW caracteres leídos del fichero y es donde buscamos coincidencias.
- `final static byte[] actualCharacters = new byte[MAX_LENGTH_COINCIDENCE]`
  - **Descripción:** estructura de datos que contiene los caracteres que vamos leyendo y comprobamos si esta secuencia ocurre en la ventana corrediza.

Métodos

- `public LZSS()`
  - **Descripció:** Creadora vacía LZSS.
  - **Return:** Objeto LZSS con *slidingWindow* y *actualCharacters* inicializados.
- `public void compress(final IO.Byte.reader input, final IO.Bit.writer  output)`
  - **Descripción:** Comprime un archivo utilizando el algoritmo LZSS.
  - **Parámetros:**
    - input: Objeto de lectura del archivo que se quiere comprimir.
    - output: Objeto de escritura al archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `public void decompress(final IO.Bit.reader input, final  IO.Byte.writer output)`
  - **Descripción:** Descomprime un archivo utilizando el algoritmo LZSS.
  - **Parámetros:**
    - input: Objeto de lectura del archivo comprimido.
    - output: Objeto de escritura al archivo descomprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `private void computeLPSArray(final int[] lps, int patLength)`
  - **Descripción:** Calcula el vector *lps*, que para cada posición del vector nos dice la longitud máxima del prefijo que también es sufijo hasta esa posición.

- **Parámetros:**
  - lps: Es un vector vacío que tras ejecutar esta función contiene para cada posición del vector la longitud máxima del prefijo que también es sufijo desde el principio hasta esa posición.
  - pathLength: Length del pattern para el que se quiere computar el array lps.
- **Return:** Es void por tanto no devuelve nada.

- `private int kmp(int currentACIndex, int currentSWIndex, boolean fullSW)`
  - **Descripción:** Usando el algoritmo Knuth-Morris-Pratt calcula índice de la primera ocurrencia de un patrón.
  - **Párametros:**
    - currentACIndex: La length del pattern,
    - currentSWIndex: El indice donde empieza el slidingWindow, ya que es circular.
    - fullSW: Indica el estado de la slidingWindow, true si lo está, false en caso contrario.
  - **Return:** Devuelve el indice, empezando por el final, de la primera ocurrencia de actualCharacters dentro de slidingWindow o -1 si actualCharacters no se encuentra dentro del slidingWindow.
- `private static double log2(final double n)`
  - **Descripción:** Calcula el logaritmo en base 2 de un real.
  - **Parámetros:**
    - n: Numero para el cual queremos calcular su logaritmo en base 2.
  - **Return:** Devuelve el logaritmo en base 2 de n.

LZ78

Descripción: Interfície con la que el usuario interactúa para elegir el algoritmo y denotar el fichero que se quiere comprimir/descomprimir y el archivo de destino.

Atributos

- `public LZ78()`
  - **Descripción:** Constructora vacía LZ78.
  - **Return:** Objeto LZ78.
- `private static Map<Integer, ArrayList<Byte>> decompress_dict = new HashMap<Integer, ArrayList<Byte>>();`
  - **Descripción:** Diccionario usando en la descompresión.
- `public final static byte MAGIC_BYTE`
  - **Descripción:** Magic byte que identifica al compresor para saber con que algoritmo debemos descomprimir.

Métodos

- `public void compress(final IO.Byte.reader input, final IO.Bit.writer output)`
  - **Descripción:** Comprime el archivo pasado por parámetro input y escribe la compresión en el parámetro output.
  - **Párametros:**
    - input: Objeto de lectura del archivo que se quiere comprimir.
    - output: Objeto de escritura al archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `private static void printArray (List<Pair <Integer, Byte>> arrayList, final IO.Bit.writer output)`
  - **Descripción:** Llamada para escribir en el archivo comprimido el array pasado como parámetro.
    - arraylist: ArrayList que contiene la codificación del archivo original.
    - output: Salida de tipo IO.Bit.writer para escribir en el archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `private static int bits_needed(final int n)`
  - **Descripción:** Calcula el numero de bits necesarios para codificar en base 2 el int pasado por parámetro.
  - **Parámetro:**
    - n : Numero integer del que se va a calcular cuantos bits son necesarios para codificarlo en base 2
  - **Return:** Devuelve el numero de bits necesarios para codificar en base 2 el int pasado por parámetro.
- `public void decompress(final IO.Bit.reader input, final IO.Byte.writer output)`

- **Descripción:** Descomprime el archivo comprimido pasado por input y escribe la descompresión por el parámetro output.
- **Párametros:**
  - input: Objeto de lectura del archivo comprimido.
  - output: Objeto de escritura al archivo descomprimido.
- **Return:** Es void por tanto no devuelve nada.

Subclase Pair

Descripción: Clase Pair para poder crear, en este caso, pairs de Integers y bytes.

Subclase Nodo

Descripción: Clase para crear un Nodo del árbol, el cual tiene un indice de tipo int y 256 hijos.

Subclase Tree

Descripción: Clase para inicializar un árbol con un método bool para llenarlo con Nodos codificando el archivo que se desea comprimir y devuelve true en caso de overflow.

LZW

Descripción: Compresión y descompresión de archivos con LZW.

Atributos

- `public final static byte MAGIC_BYTE`
  - **Descripción:** Magic byte que identifica al compresor para saber con que algoritmo debemos descomprimir.
- `private final static int DICTIONARY_SIZE`
  - **Descripción:** Tamaño inicial del diccionario.
- `private static Map<ArrayList<Byte>, Integer> compressionDictionary`
  - **Descripción:** Diccionario de compresión.
- `private static Map<Integer, ArrayList<Byte> > decompressionDictionary`
  - **Descripción:** Diccionario usado en la descompresión
- `private final static int EOF`
  - **Descripción:** Pseudo EOF
- `private final static int OVERFLOW`
  - **Descripción:** Overflow del diccionario

Métodos

- `public LZSS()`
  - **Descripción:** Creadora vacia LZW.
  - **Return:** Objeto LZW.
- `private static void createCompressionDictionary()`
  - **Descripción:** Crea el diccionario de compresión y lo inicializa con los valores unitarios.
  - **Return:** Es void por tanto no devuelve nada.
- `private static void createDecompressionDictionary()`
  - **Descripción:** Crea el diccionario de descompresión y lo inicializa con los valores unitarios.
  - **Return:** Es void por tanto no devuelve nada.
- `public void compress (IO.Byte.reader input, IO.Bit.writer output)`
  - **Descripción:** Comprime un archivo implementando un algoritmo LZW.
  - **Parámetros:**
    - input: Objeto de lectura del archivo que se quiere comprimir.
    - output: Objeto de escritura al archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `public void decompress (IO.Bit.reader input, IO.Byte.writer output)`
  - **Descripción:** Comprime un archivo implementando un algoritmo LZW.

    - **Parámetros:**
        - input: Objeto de lectura del archivo que se quiere descomprimir.
        - output: Objeto de escritura del archivo descomprimido.
    - **Return:** Es void por tanto no devuelve nada.
- `private static int bitsNeeded(final int n)`
    - **Descripción:** Calcula el numero de bits necesarios para codificar en base 2 el int pasado por parametro.
    - **Parámetros:**
        - n: Numero integer del que se va a calcular cuantos bits son necesarios para codificarlo en base 2.
    - **Return:** Devuelve el numero de bits necesarios para codificar en base 2 el int pasado por parámetro.
- `private static void writeCode (int code, final IO.Bit.writer output)`
    - **Descripción:** Escribe en output el entero n en 5 bits seguido del entero code representado en n bits.
    - **Parámetros:**
        - code: Numero integer que se quiere escribir
        - output: Salida de tipo IO.Bit.writer para escribir en el archivo comprimido.
    - **Return:** Es void por tanto no devuelve nada.

JPEG

Descripción: Compresión y descompresión de imágenes PPM con JPEG.

Atributos

- `private short quality`
    - **Descripción:** Calidad de compresión.
- `public final static byte MAGIC_BYTE`
    - **Descripción:** Magic byte del JPEG.

Métodos

- `public JPEG(final short quality)`
    - **Descripción:** Constructora JPEG con calidad, inicializa quality con la calidad dada.
    - **Parámetros:**
        - quality: Calidad de compresión.
    - **Return:** Devuelve un objeto JPEG con una calidad definida.
- `public void compress(final IO.Byte.reader input, final IO.Bit.writer  output)`
    - **Descripción:** Comprime una imagen PPM bloque a bloque.
    - **Parámetros:**
        - input: Objeto lector del fichero de entrada.
        - output: Objeto escritor fichero comprimido
    - **Return:** Es void por tanto no devuelve nada.
- `public void decompress(IO.Bit.reader input, IO.Byte.writer output)`
    - **Descripción:** Descomprime un fichero comprimido en JPEG y lo guarda la imagen resultante en un fichero PPM raw.
    - **Parámetros:**
        - input: Objeto lector del fichero comprimido.
        - output: Objeto escritor fichero descomprimido.
    - **Return:** Es void por tanto no devuelve nada.
- `static short[] readBlock(final Huffman huffAC, final Huffman huffDC, final  IO.Bit.reader file)`
    - **Descripción:** Lee un bloque codificado con las tablas Huffman.
    - **Parámetros:**
        - huffAC: Tabla Huffman de valores AC.
        - huffDC: Tabla Huffman de valores DC.
        - file: Fichero comprimido del que leer.
    - **Return:** Devuelve un bloque codificado sin Huffman.
- `private static short readHuffman(Huffman huff, IO.Bit.reader file)`
    - **Descripción:** Lee un código Huffman del fichero y lo decodifica.

    - **Parámetros:**
        - huff: Objeto tipo Huffman.
        - file: Objeto escritor fichero comprimido.
    - **Return:** Devuelve el código Huffman decodificado.
- `static void writeBlock(final short[] encoded, final Huffman huffAC, final Huffman huffDC, final IO.Bit.writer file)`
    - **Descripción:** Escribe un bloque codificado con las tablas Huffman.
    - **Parámetros:**
        - encoded: Codificado sin Huffman.
        - huffAC: Tabla Huffman de valores AC.
        - huffDC: Tabla Huffman de valores DC.
        - file: Fichero comprimido al que escribir.
    - **Return:** Es void por lo tanto no devuelve nada.
- `private static void write(int value, int l, IO.Bit.writer file)`
    - **Descripción:** Dado un valor, una longitud y un objeto de escritura, escribe el valor representado con la longitud dada en el objeto de escritura.
    - **Parámetros:**
        - value: Valor a escribir.
        - l: Longitud de valor que se escribe.
        - file: Objeto de escritura del archivo al que se escribe.
    - **Return:** Es void por tanto no devuelve nada.
- `private static short read(int length, IO.Bit.reader file)`
    - **Descripción:** Dada una longitud y un objeto de lectura, se leen los bits correspondientes a la longitud dada del objeto de lectura.
    - **Parámetros:**
        - l: Longitud que se lee.
        - file: Objeto de lectura del archivo del cual se lee.
    - **Return:** Short leído.
- `private static byte[][][] toYCbCr(byte[][][] channelBlocks)`
    - **Descripción:** Convierte el bloque dado en espacio de color RGB a YCbCr.
    - **Parámetros:**
        - channelBlocks: Bloque en espacio de color RGB.
    - **Return:** Bloque en espacio de color YCbCr.
- `private static byte[][][] toRGB(byte[][][] channelBlocks)`
    - **Descripción:** Convierte el bloque dado en espacio de color YCbCr a RGB.
    - **Parámetros:**
        - channelBlocks: Bloque en espacio de color YCbCr.
    - **Return:** Bloque en espacio de color RGB.
- `* private static byte doubleToByte(final double d)`
    - **Descripción:** Dado un double, devuelve dicho double en formato de byte.
    - **Parámetros:**
        - d: Double a transformar.
    - **Return:** El double en fromato de byte.

JPEGBlock

Descripción: Codifica y decodifica bloques 8x8 con JPEG.

Métodos

- `private static int bitLength(final short n)`
    - **Descripción:** Calcula úmero de bits necesarios para representar el short n.
    - **Parámetros:**
        - n: Short a representar.
    - **Return:** Devuelve el número de bits necesarios para representar n.

- `public static short[] encode(final short quality, final boolean  isChrominance, final byte[][] data)`
  - **Descripción:** Comprime un bloque 8x8 aplicando DCT, cuantización, zigZag y RLE.
  - **Parámetros:**
    - quality: Calidad de compresión (1-100).
    - isChrominance: Si es un bloque de Chrominance (si falso Luminance).
    - data: Bloque 8x8 a codificar.
  - **Return:** Devuelve el bloque 8x8 codificado en RLE.
- `public static byte[][] decode(final short quality, final boolean  isChrominance, final short[] data)`
  - **Descripción:** Deshace RLE, zigZag, quantización y DCT para obtener el bloque 8x8 original.
  - **Párametros:**
    - quality: Calidad de compresión (1-100).
    - isChrominance: Si es un bloque de Chrominance (si falso Luminance).
    - data: Bloque 8x8 a codificado en RLE.
  - **Return:** Devuelve el bloque 8x8 decodificado.

Subclase: DCT

Descripción: Codificación y decodificación del paso *Discrete Cosine Transform* de la compresión/descompresión JPEG para un bloque de 8x8.

Subclase: Quantization

Descripción: Codificación y decodificación del paso de cuantización con tablas predefinidas, ajustadas según la calidad de compresión especificada, para la compresión/descompresión JPEG de un bloque de 8x8.

Subclase: ZigZag

Descripción: Aplasta un bloque 8x8 bytes en zigZag.

Subclase: RLE

Descripción: Codifica/Decodifica en RLE (Run Length Encoding).

PpmImage

Descripción: Imagen PPM.

Subclase: Reader

Descripción: Lector de PpmImage.

Subclase: Writer

Descripción: Escritor de PpmImage.

Subclase: FileFormatException

Descripción: Excepción del formato de archivo.

Huffman

Descripción: codificación y decodificación Huffman con tablas predefinidas

Atributos

- `private Node root`
  - **Descripción:** Raíz del árbol Huffman.

- `private Map<Short, BitSetL> table`
  - **Descripción:** Tabla de Huffman.

Métodos

- `public Huffman(final boolean isAC, final boolean isChrominance)`
  - **Descripción:** Leer la tabla del disco en memoria y construye el árbol.
  - **Parámetros:**
    - isAC: Si cierto se lee la tabla AC, sino la DC.
    - isChrominance: Si cierto se lee la tabla de Chrominance, sino Luminance.
  - **Return:** Árbol de Huffman construido
- `private void readTable(final String filename)`
  - **Descripción:** Lee una tabla Huffman.
  - **Parámetros:**
    - filename: Nombre del archivo a leer.
  - **Return:** Es void por lo tanto no devuelve nada.
- `private void addToTree(final Short value, final BitSetL bs)`
  - **Decripción:** Añade el valor value en el árbol siguiendo el camino marcado por el BitSetL bs.
  - **Parámetros:**
    - value: Valor a añadir.
    - bs: Código Huffman correspondiente.
  - **Return:** Es void por lo tanto no devuelve nada.
- `public BitSetL encode(final Short value)`
  - **Descripción:** Devuelve el código Huffman asociado a un valor.
  - **Parámetros:**
    - value: Valor a buscar en la tabla.
  - **Return:** Código Huffman relacionado al valor pasado por parámetro.
- `public Node decode(final boolean b)`
  - **Descripción:** Devuelve el siguiente nodo del árbol desde la raíz.
  - **Parámetros:**
    - b: Bit a decodificar.
  - **Return:** Siguiente nodo del árbol desde la raíz.
- `public Node decode(final Node n, final boolean b)`
  - **Descripción:** Devuelve el siguiente nodo del árbol desde el Nodo pasado por párametro.
  - **Parámetros:**
    - n: Nodo del árbol.
    - b: Bit a decodificar.
  - **Return:** Siguiente nodo en el árbol.

Subclase: Node

Descripción: Nodo del arbol huffman.

Subclase: LookupException

Descripción: Excecpión de búqueda de un valor en la tabla.

Subclase: InvalidTableException

Descripción: Excepción de tabla invalida.

BitSetL

Descripción: BitSet con length() específica.

Atributos

- `private int _length_;`

- **Descripción:** Longitud del BitSetL.

Métodos

- `public BitSetL()`
  - **Descripción:** Constructora vacía, inicializa length a -1.
  - **Return:** Devuelve un BitSet con length -1.
- `public BitSetL(final int l)`
  - **Descripción:** Constructora con longitud, inicializa length a la longitud dada.
  - **Parmámetros:**
    - l: Longitud del bitset.
  - **Return:** Devuelve un BitSet con la length deseada.
- `public BitSetL(final int n, final int l)`
  - **Descripción:** Constructora con entero y longitud, inicializa length a la longitud dada y mete los l últimos bits del entero en el bitset.
  - **Parámetros:**
    - n: Entero a codificar.
    - l: Longitud del bitset.
  - **Return:** Devuelve un BitSet de longitud l con ultimos bits de n en él.
- `public BitSetL(final String s)`
  - **Descripción:** Constructora con String, inicializa length al tamaño del String y convierte la string en un bitset.
  - **Parámetros:**
    - s: String de 0 y unos.
  - **Return:** Devuelve un BitSet que es la conversión del String a un bitset y con la longitud de s.
- `public int asInt()`
  - **Descripción:** Devuelve el bitset como un entero (32 bits).
  - **Pre:** El bitset no tiene mas de 32 bits
  - **Return:** Bitset como entero.
- `public void flip()`
  - **Descripción:** Invierte los bits del bitset. (Solo hasta length).
  - **Return:** BitSet con los bits invertidos.
- `public void set(final int pos)`
  - **Descripción:** Convierte en true el bit en la posición dada.
  - **Parámetros:**
    - pos: Posición del bit a convertir en true.
  - **Return:** Es void por tanto no devuelve nada.
- `public void set(final int pos, final boolean val)`
  - **Descripción:** Convierte en true o false el bit en la posición dada en función del booleano *val*.
  - **Parámetros:**
    - pos: Posición del bit a modificar.
    - val: Valor booleano que se asignar al bit.
  - **Return:** Es void por tanto no devuelve nada.
- `public void clear(final int pos)`
  - **Descripción:** Convierte en false el bit en la posición dada.
  - **Parámetros:**
    - pos: Posicion del bit a convertir en true.
  - **Return:** Es void por tanto no devuelve nada.
- `public int length()`
  - **Descripción:** Devuelve la longitud del bitset.
  - **Return:** Longitud del bitset.

Folder

Descripción: Compresor y descompresor de carpetas.

Atributos

- `private static char EMPTY_FOLDER`
  - **Descripción:** Indicador de carpeta vacía.
- `private static char FILE`
  - **Descripción:** Indicador de archivo.
- `public final static byte MAGIC_BYTE`
  - **Descripción:** MagicByte Folder.

Métodos

- `private Folder()`
  - **Descripción:** Constructora por defecto.
  - **Return:** Objeto de la clase Folder.
- `public static void compress(String folderPath, IO.Bit.writer output)`
  - **Descripción:** Dada la dirección de la carpeta a comprimir y un objeto de escritura del archivo comprimido, ejecuta la compresión con el algoritmo pertienente.
  - **Parámetros:**
    - folderPath: Dirección de la carpeta a comprimir.
    - output: Objeto de escritura del archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `public static void decompress(String folderPath, IO.Bit.reader input)`
  - **Descripción:** Dada la dirección de la carpeta a descomprimir y un objeto de lectura del archivo comprimido, ejecuta la descompresión con el algoritmo pertienente.
  - **Parámetros:**
    - folderPath: Dirección de la carpeta a descomprimir.
    - input: Objeto de lectura del archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.

Subclase: FolderFormatException

Descripción: Excepcion de formato de carpeta.

Subclase: CompressFiles

Descripción: File visitor que comprime los archivos.

IO

Descripción: Clases IO para lectura y escritura de Char, Byte, y Bit con buffer.

Subclase: Char

Descripción: Lector/escritor char a char con buffer.

Subclase: reader **Descripción**: Extiende la clase *BufferedReader* para poder construir un buffer de lectura char a char a partir del archivo que recibe.

Subclase: writer **Descripción**: Extiende la clase *BufferedWriter* para poder construir un buffer de escritura char a char a partir del archivo en el que quiere escribir.

Subclase: Byte

Descripción: Lector/escritor byte a byte con buffer.

Subclase: reader **Descripción**: Extiende la clase *BufferedInputStream* para poder construir un buffer de lectura byte a byte a partir del archivo que recibe.

Subclase: writer **Descripción**: Extiende la clase *BufferedOutputStream* para poder construir un buffer de escritura byte a byte a partir del archivo en el que quiere escribir.

Subclase: Bit

Descripción: Lector/escritor bit a bit con buffer.

Subclase: reader **Descripción**: Permite leer bit a bit un fichero. Además incorpora la posibilidad de leer también bytes, chars, ints e incluso BitSetL.

Atributos

- `private final BufferedInputStream in`
  - **Descripción**: *BufferedInputStream* a través del cual leemos.
- `private int buffer`
  - **Descripción**: Buffer de 8 bits donde se van almacenando los bits leidos, ya que solo podemos leer un byte como mínimo.
- `private int n`
  - **Descripción**: Número de bits almacenados en el buffer.

Métodos

- `public reader(final String filename)`
  - **Descripción**: Constructora que incializa el *BufferedInputStream* para poder leer del fichero y inicializa también los atributos n y buffer.
  - **Return**: Es void por tanto no devuelve nada
- `private void fill()`
  - **Descripción**: Lee un byte del fichero, unidad mínima que se puede leer, y lo almacena en el buffer para que pueda ser leido bit a bit. También actualiza el valor de n a 8.
- `public boolean read()`
  - **Descripción**: Si quedan bits en el buffer retorna el siguient bit, sinó rellena el buffer leyendo un byte del fichero y retorna el bit de mayor peso.
  - **Return**: Retorna el siguiente bit leído del fichero.
- Existen varios *read[Byte/Char/Int/BitSetL]()* que leen del archivo un byte, char, int y BitSetL y devuelven el byte, char, int y BitSetL respectivamente. Además en el caso del BitSetL recibe como parámetro cuantos bits se quieren leer.
- `public void close()`
  - **Descripción**: Invocado al finalizar la lectura para cerrar el *BufferedInputStream*.
  - **Return**: Es void por tanto no devuelve nada

Subclase: writer **Descripción**: Permite escribir bit a bit a un fichero. Además implementa la posibilidad de escribir también bytes, chars, ints y BitSetL.

Atributos

- `private final BufferedOutputStream out`
  - **Descripción**: *BufferedOutputStream* a través del cual escribimos.
- `private int buffer`
  - **Descripción**: Buffer de 8 bits donde se van almacenando los bits que se quieren escribir.
- `private int n`
  - **Descripción**: Número de bits almacenados en el buffer.

Métodos

- `public writer(final String filename)`
  - **Descripción**: Constructora que incializa el *BufferedOutputStream* para que abra un fichero al que queremos escribir y inicializa también los atributos n y buffer.
  - **Return**: Es void por tanto no devuelve nada.

- `public void write(final boolean bit)`
  - **Descripción**: Escribe un bit en buffer, cuando el buffer esta lleno (un byte) se escribe por *out*.
  - **Return**: Es void por tanto no devuelve nada.
- Tiene varios métodos `write(...)` que escriben byte, char, int y BitSetL a través de *out*. Son todas `void` y por lo tanto no retornan nada.
- `private void writeMask(final int num, int mask)`
  - **Descripción**: Escribe un entero usando *mask* bits.
  - **Return**: Es void por tanto no devuelve nada.
- `private void clear()`
  - **Descripción**: Se invoca para escribir en el fichero los bits que hay en el buffer, si faltan bits para completar el byte se rellena con ceros en los bits de menor peso.
  - **Return**: Es void por tanto no devuelve nada.
- `public void flush()`
  - **Descripción**: Usado para llamar al método `clear()` y hace un *flush* en el *BufferedOutputStream*.
  - **Return**: Es void por tanto no devuelve nada.
- `public void close()`
  - **Descripción**: Llamar a `flush()` y cierra el *BufferedOutputStream*.
  - **Return**: Es void por tanto no devuelve nada.

Statistics

Descripción: Genera las estadísticas de compresión/descompresión.

Atributos:

- `private long startingTime`
  - **Descripción:** Tiempo de inicio.
- `private long endingTime`
  - **Descripción:** Tiempo de finalización.
- `private long iniFileSize`
  - **Descripción:** Tamaño inicial.
- `private long finFileSize`
  - **Descripción:** Tamaño final.

Métodos

- `public Statistics ()`
  - **Descripción:** Constructora vacía Statistics.
  - **Return:** Objeto Statistics con los parametros inicializados a 0.
- `public void setIniFileSize (final String filename)`
  - **Descripción:** Setter de iniFileSize.
  - **Parámetros:**
    - filename: Nombre del archivo de entrada cuyo tamaño se quiere consultar.
  - **Return:** Es void por tanto no devuelve nada.
- `public void setFinFileSize(final String filename)`
  - **Descripción:** Setter de finFileSize.
  - **Parámetros:**
    - filename: Nombre del archivo de entrada cuyo tamaño se quiere consultar.
  - **Return:** Es void por tanto no devuelve nada.