Laboratorio A.E.D.

Guillermo Román
guillermo.roman@upm.es
Lars-Åke Fredlund
lfredlund@fi.upm.es
Manuel Carro
manuel.carro@upm.es
Marina Álvarez
marina.alvarez@upm.es
Julio García
juliomanuel.garcia@upm.es
Tonghong Li
tonghong@fi.upm.es
José Ramón Sánchez
joseramon.sanchezp@fi.upm.es

Normas

- Fechas de entrega y penalización aplicada a la puntuación obtenida:

Hasta el Lunes 27 de noviembre, 23:59 horas	0 %
Hasta el Martes 28 de noviembre, 23:59 horas	20 %
Hasta el Miércoles 29 de noviembre, 23:59 horas	40 %
Hasta el Jueves 30 de noviembre, 23:59 horas	60 %
Después la puntuación máxima será 0	

- Se comprobará plagio y se actuará sobre los detectados
- Usad las horas de tutoría para preguntar sobre programación – son oportunidades excelentes para aprender

Entrega

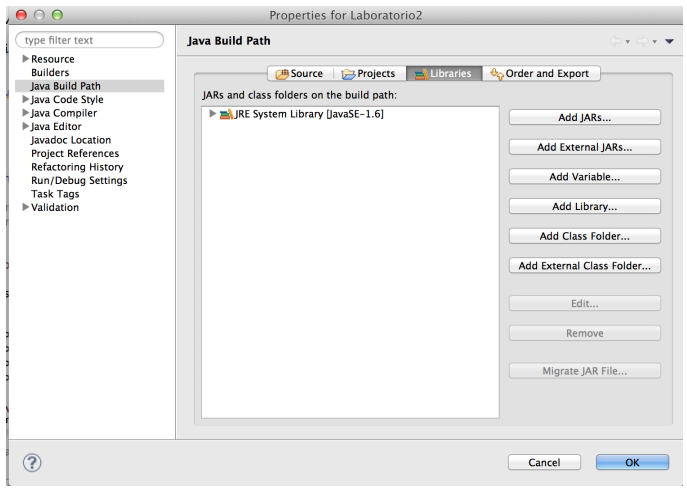
- Todos los ejercicios de laboratorio se deben entregar a través de la web `http://lml.ls.fi.upm.es/~entrega`.
- Los ficheros que hay que entregar son: `Huffman.java`.

Configuración previa

- Arrancad Eclipse
- Si trabajáis en portátil, podéis utilizar cualquier versión relativamente reciente de Eclipse. Debería valer cualquier versión a partir de la versión 3.7. Es suficiente con que instaléis la *Eclipse IDE for Java Developers*
- Cambiad a "Java Perspective".
- Cread un proyecto Java llamado `aed`:
  - Seleccionad separación de directorios de fuentes y binarios
- Cread un *package* `aed.huffman` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios y Entregas Individuales → Laboratorio 5 → `Laboratorio5.zip`; descomprimidlo
- Contenido de `Laboratorio5.zip`:
  - `TesterLab6.java`, `CharCode.java`, `Huffman.java`

Configuración previa al desarrollo del ejercicio.

- Importad al paquete `aed.huffman` las fuentes que habéis descargado (`TesterLab6.java`, `CharCode.java`, `Huffman.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios y Entregas Individuales). Para ello:
- Project → Properties → Java Build Path. Se abrirá una ventana como esta:



- Usad la opción “Add External JARs...”.
- Intentad ejecutar `TesterLab6.java`

Tarea: "Huffman encoding" usando árboles binarios

- La *Codificación Huffman* es una técnica de compresión de cadenas de caracteres (o, en general, de palabras de tamaño fijo de n bits)
- En lugar de usar 8 "bits" para representar cada carácter, el numero de "bits" puede ser distinto para cada carácter
- La idea es asignar un código lo más corto posible para aquellos caracteres que aparecen más frecuentemente en el texto (p.e. 'a') y códigos mas largos para caracteres con poca frecuencia de aparición (p.e. 'k').
- La compresión que proporciona la *codificación Huffman* depende mucho del texto inicial, pero se obtienen ahorros de entre el 20 % y el 90 % del tamaño inicial.

“Huffman encoding”

La clase `Huffman` tiene dos métodos para codificar, y decodificar, textos:

```
public String encode(String text);
public String decode(String encodedText);
```

Ejemplo:

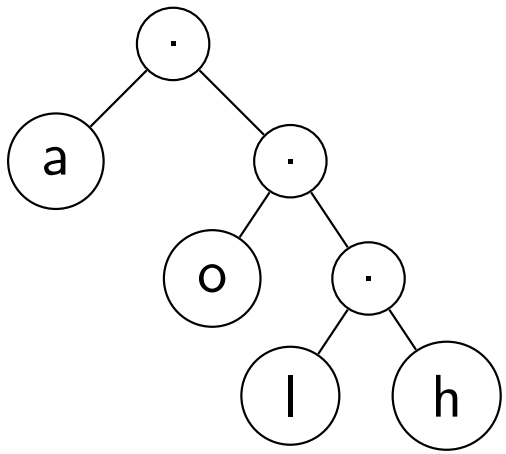
```
huffman.encode("hola");          ==>  "111101100"
huffman.decode("111101100")      ==>  "hola"
```

NOTA: La representación real de los códigos Huffman sería con una secuencia de bits, y no como un String de '1' y '0'. Sin embargo, para facilitar la tarea de programación de este laboratorio, usaremos Strings

“Huffman encoding”

La codificación Huffman se puede representar con un árbol binario donde los *caminos* “hacia la izquierda” representan un bit ’0’ y los caminos “hacia la derecha” representa un bit ’1’

Ejemplo:



Carácter	Codificacion
’a’	”0”
’o’	”10”
’l’	”110”
’h’	”111”

Como resultado `huffman.encode("hola"); ==> "111101100"`

’h’	’o’	’l’	’a’
111	10	110	0

Tarea: completar la clase `Huffman`

Hay tres métodos por completar, el constructor y encode y decode:

```
class Huffman {
  private BinaryTree<Character> huffmanTree;

  public Huffman(CharCode[] paths) {
    this.huffmanTree = new LinkedBinaryTree<Character>();
    // ...
  }

  public String encode(String text) { ... }
  public String decode(String encodedText) { ... }
}
```

`public Huffman(CharCode[] paths)`

- El constructor recibe un array de pares `CharCode` que empareja cada letra con su codificación:

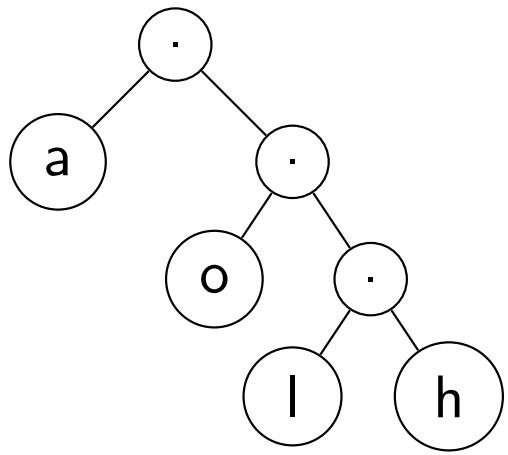
```
public class CharCode {
  private Character ch;
  private String code;
  // ... + getters
}
```

- Con él se crea el árbol para esos pares y se guarda en el atributo `huffmanTree`.

- Para crear el árbol hay que usar los métodos del interfaz `BinaryTree`:
  - `addRoot(E a)` que crea la raíz del árbol
  - `insertLeft(Position<E> v, E a)` que inserta un hijo izquierdo, y
  - `insertRight(Position<E> v, E a)` que inserta un hijo derecho.

`public Huffman(CharCode[] paths)`

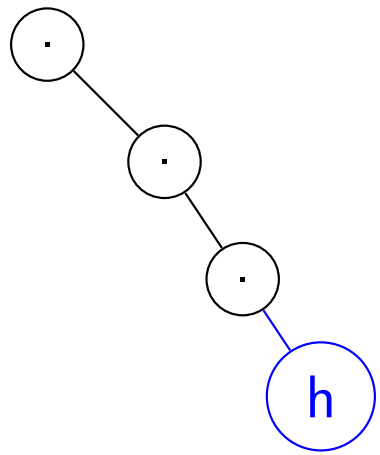
- Dado [<'h',"111">,<'o',"10">,<'l',"110">,<'a',"0">]
- El constructor debe crear en el atributo `huffmanTree` el árbol:



Los nodos internos contendrán un carácter ' ' (espacio en blanco) y los nodos externos el carácter correspondiente a la codificación Huffman

public Huffman(CharCode[] paths)

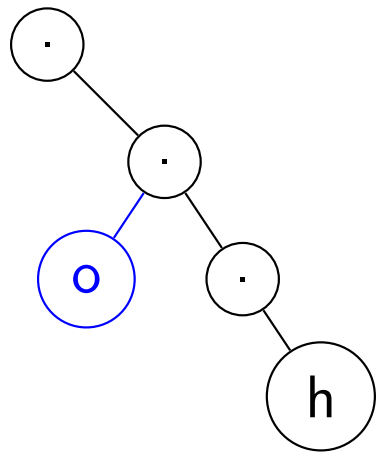
- Dado [<'h',"111">,<'o',"10">,<'l',"110">,<'a',"0">]
- Paso a paso...



Con [<'h ',"111">,...] creamos "derecha,derecha,derecha" y 'h'

public Huffman(CharCode[] paths)

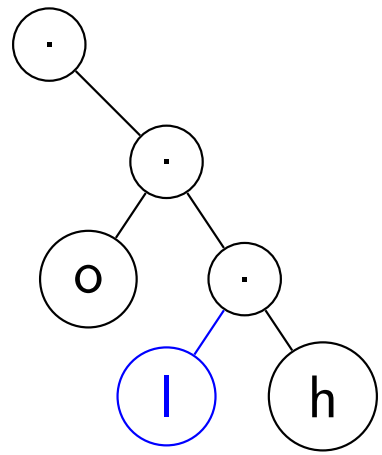
- Dado [<'h',"111">,<'o',"10">,<'l',"110">,<'a',"0">]
- Paso a paso...



Con [...,<' o ',"10">,...] avanzamos "derecha"
y creamos "izda" y 'o'

public Huffman(CharCode[] paths)

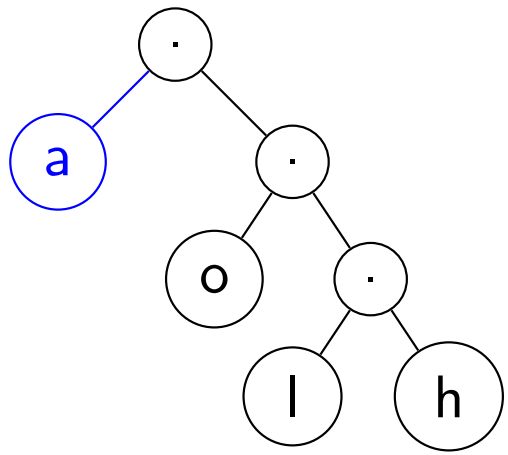
- Dado [<'h',"111">,<'o',"10">,<'l',"110">,<'a',"0">]
- Paso a paso...



Con [...,<' l ',"110">,...] avanzamos "derecha,derecha"
y creamos "izda" y 'l'

public Huffman(CharCode[] paths)

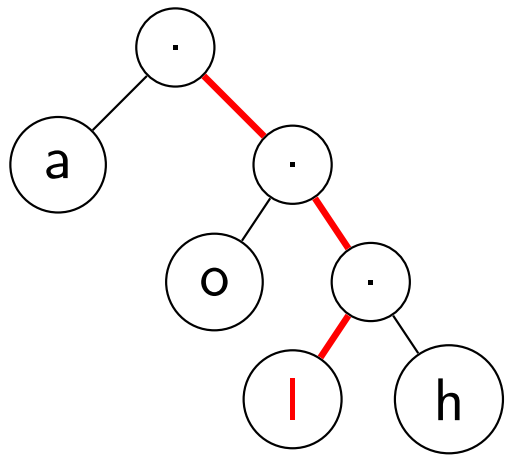
- Dado [<'h',"111">,<'o',"10">,<'l',"110">,<'a',"0">]
- Paso a paso...



Con [...,<' a',"0">] creamos "izquierda" y 'a'

encode(String text)

- Devuelve un `String` donde cada carácter se codifica por el camino (codificado como caracteres '0' y '1') para llegar al nodo con el carácter en el árbol Huffman
- Por ejemplo, para obtener el código del carácter 'l', hay que encontrar el camino desde la raíz al carácter 'l':



- Como el camino para llegar al nodo 'l' es "derecha, derecha, izquierda" su codigo es "110".

encode(String text)

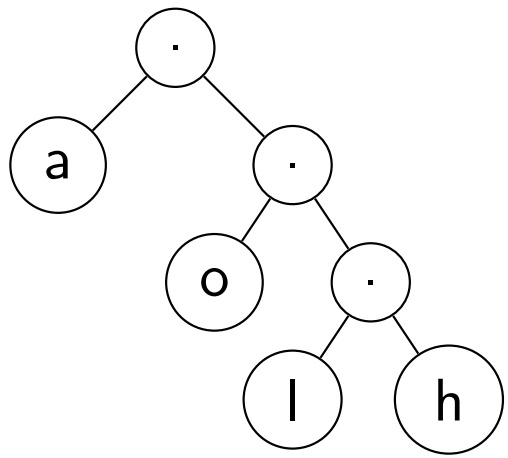
- El método `encode` implica buscar en el árbol de Huffman el carácter
- Nos interesa el camino utilizado para llegar (recordad el Individual 5)
- Incluimos la cabecera de un método privado que os puede ser útil para buscar el carácter `ch` en el árbol

```
String findCharacterCode(Character ch,
                         BinaryTree<Character> tree,
                         Position<Character> pos,
                         String path) {
```

- Recomendamos hacerlo recursivo, de tal forma que vaya recorriendo el árbol y, mientras no encuentre el carácter `ch` devuelva `null` y que cuando lo encuentre devuelva el camino completo de 0's y 1's

`decode(String encodedText)`

- Para traducir un `String` codificado hay que seguir el camino definido por la codificación hasta que llegar a un carácter en el árbol Huffman (que estarán siempre en las hojas)
- Una vez el encontrado un carácter, volver a empezar en el árbol para continuar con el resto de caracteres del String
- Por ejemplo, dado el árbol y el texto codificado "10111"



- Se sigue el camino "derecha, izquierda" (correspondiente al prefijo "10" del "10111") obteniendo 'o'
- Se continua con el resto del texto "111", obteniendo 'h'
- Por tanto: `decode("10111") => "oh"`

Notas

- Es **obligatorio** usar el atributo `huffmanTree` (un `BinaryTree`) para guardar el árbol Huffman.
- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar, y debe ejecutar `TesterLab6` correctamente sin mensajes de error
- Nota: una ejecución sin mensajes de error no significa que el método sea correcto (es decir, que funcione bien para cada posible entrada)
- Todos los ejercicios se comprueban manualmente antes de dar la nota final