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 4 de diciembre, 23:59 horas	0 %
Hasta el Martes 5 de diciembre, 23:59 horas	20 %
Hasta el Jueves 7 de diciembre, 23:59 horas	40 %
Hasta el Lunes 11 de diciembre, 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

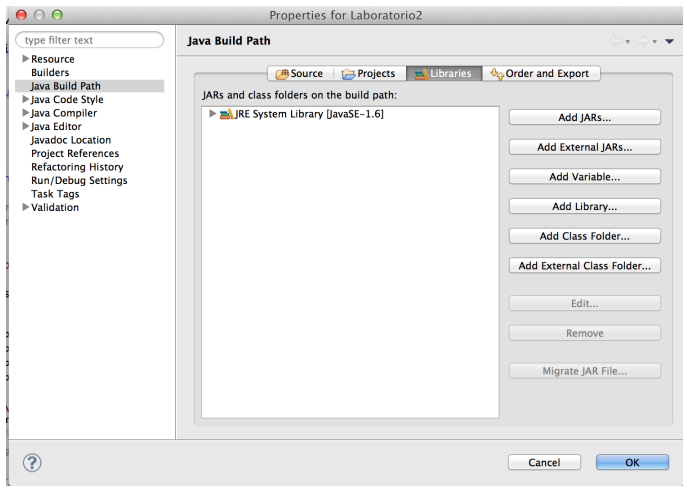
- Todos los ejercicios de laboratorio se deben entregar a través de la web `http://lml.ls.fi.upm.es/~entrega`.
- Los ficheros que hay que entregar son: `Huffman.java`,`AttachableLinkedBinaryTree.java`.

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.compress` en el proyecto `aed`, dentro de `src`
- Aula Virtual $\rightarrow$ AED $\rightarrow$ Laboratorios y Entregas Individuales $\rightarrow$ Laboratorio 6 $\rightarrow$ `Laboratorio6.zip`; descomprimidlo
- Contenido de `Laboratorio6.zip`:
  - `TesterLab6.java`, `AttachableBinaryTree.java`, `AttachableLinkedBinaryTree.java`, `Huffman.java`

Configuración previa al desarrollo del ejercicio.

- Importad al paquete `aed.compress` las fuentes que habéis descargado (`TesterLab6.java`, `AttachableBinaryTree.java`, `AttachableLinkedBinaryTree.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`

Cómo ejecutar una sola prueba

- Se puede ejecutar una sola prueba fácilmente cambiando la clase `RunOneTest.java` y ejecutándola
- Por defecto la clase contiene

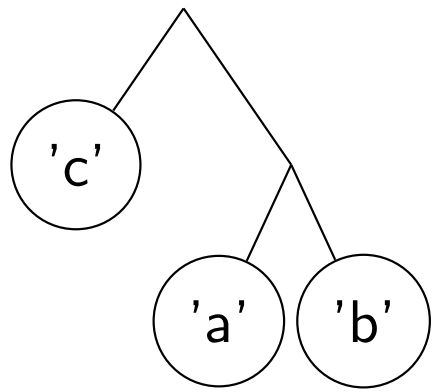
```
package aed.compress;

public class RunOneTest {
  public static void main(String args[]) {
    TesterLab6.test_5();
  }
}
```

  para poder correr el "test_5".
- Otra alternativa para ejecutar un test nada más es especificar su número como parámetro para el programa `TesterLab6` dentro Eclipse: "Run As" $\Rightarrow$ "Run Configurations" $\Rightarrow$ "TesterLab6" $\Rightarrow$ "Arguments".

Tarea: construyendo arboles "Huffman"

- En el último laboratorio implementamos la clase `Huffman` con un constructor `Huffman(CharCode[] paths)`
- Pero, ¿de dónde vienen los códigos asociados a cada carácter?
- Falta por descubrir cómo se puede "diseñar" un árbol Huffman dado un cierto texto. Esa es la tarea de hoy
- Ejemplo, un árbol Huffman dado el texto "`abbccc`":



Como el carácter `'c'` es el carácter más utilizado, el camino para llegar a `'c'` es el más corto

Tres subtareas

1. Implementar un método

   ```
   int[] countChars(String text)
   ```

   que cuenta el número de veces que un carácter ocurre en `text`

2. Implementar el método

   ```
   void attach(Position<E> pos,
               BinaryTree<E> leftTree,
               BinaryTree<E> rightTree)
   ```

   que permite adjuntar dos subárboles (uno izquierdo y uno derecho) debajo de una hoja (`pos`) en un árbol binario

3. Implementar el constructor

   ```
   public Huffman (String text)
   ```

   que implementa un algoritmo para construir un árbol Huffman "óptimo", usando el resultado del método `countChars`

Tarea 1: contar ocurrencias de caracteres

Hay que implementar el método

```
static int[] countChars(String text)
```

dentro el fichero `Huffman.java`.

- El método devuelve un array `arr = new int[256];` (el número de caracateres de la tabla ASCII)
- El contenido de `arr[ch]`, donde `ch` es un `char`, es el número de veces `ch` ocurre en `text`
- Si un carácter no ocurre en el texto, su posición en el array tendrá valor cero
- Por ejemplo, dado el texto "abbca":
  `arr['a']=2`, `arr['b']=2`, `arr['c']=1`,
  y para cada carácter `ch` $\notin \{$`'a'`, `'b'`, `'c'`$\}$: `arr[ch]=0`.

Tarea 2: adjuntar subárboles en un árbol binario

Hay que implementar el método

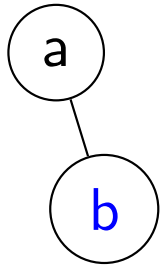
```
void attach(Position<E> pos,
            BinaryTree<E> leftTree,
            BinaryTree<E> rightTree)
```

dentro la clase `AttachableLinkedBinaryTree`, que extiende la clase `LinkedBinaryTree` con este método

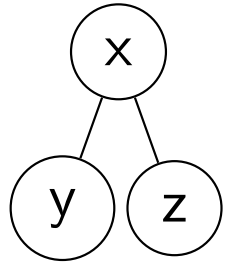
- Dado un árbol (`this`), y una hoja del árbol (`pos`), el método debe añadir el árbol `leftTree` como el subárbol izquierdo de `pos`, y el árbol `rightTree` como el subárbol derecho
- Para añadir un árbol como un subárbol de `pos` hay construirlo paso por paso (se recomienda recursivo), usando:
  - `insertLeft(pos,element)` – para insertar un nuevo nodo como hijo izquierdo a `pos`
  - `insertRight(pos,element)` – para insertar un nuevo nodo como hijo derecho a `pos`

Tarea 2: implementar `attach`: Ejemplo

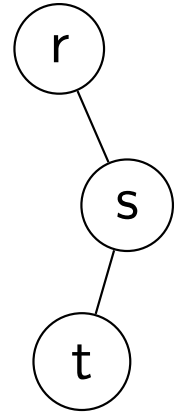
- Dados los siguiente árboles:



`tree`

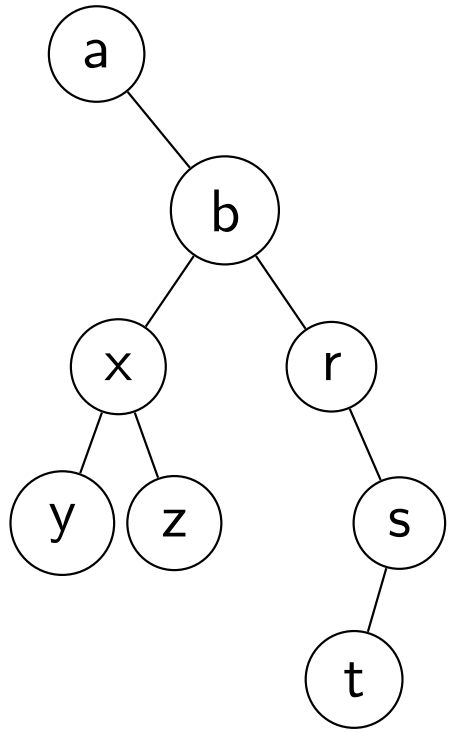


`lt`



`rt`

- El resultado de la llamada a `tree.attach(posb,lt,rt)` da el siguiente árbol:



Tarea principal: construir un árbol Huffman

- Hay que implementar el constructor

  ```
  Huffman(String text)
  ```

- Recibe un `String` y llama al método llama al método `constructHuffmanTree` para crear el árbol
- El método `constructHuffmanTree(text)` llamará al método `countChars(text)` para calcular las frecuencias de aparición y devolverá el árbol de Huffman construido (ver el algoritmo de la siguiente slide)
- En dicho árbol, los caminos a los caracteres que ocurren más frecuentemente en `text` son más cortos que los caminos que ocurren a caracteres con menos frecuencia

Algoritmo para crear el árbol Huffman

Dado un text, `int[] charCode` $\leftarrow$ `countChars(`*text*`)`
Crea `PriorityQueue<Integer,BinaryTree<Character>> Q`

for all *c* in `0...charCode.length` **do**
  if `charCode[c]` $>$ 0 **then**
    create a single-node binary tree *T* storing c
    insert T into Q with the key `charCode[c]`
  end if
end for
while `Q.size() > 1` **do**
  left $\leftarrow$ `Q.dequeue()`
  right $\leftarrow$ `Q.dequeue()`
  Create a new binary tree T with left subtree *left*
  and right subtree *right*
  Insert T into Q with key *left*`.getKey()` + *right*`.getKey()`
end while
return the value of the only remaining entry of the queue

Notas

- Los árboles que vayáis creando en el método `constructHuffmanTree` deben ser de tipo `AttachableLinkedBinaryTree` para poder juntarlos con el método `attach`
- 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