Laboratorio A.E.D. Laboratorio 4

Guillermo Román
guillermo.roman@upm.es

Lars-Åke Fredlund
lfredlund@fi.upm.es

Manuel Carro
mcarro@fi.upm.es

Marina Álvarez
marina.alvarez@upm.es

Julio García
juliomanuel.garcia@upm.es

Tonghong Li
tonghong@fi.upm.es

Sergio Paraíso
sergio.paraiso@upm.es

Normas

- Fechas de entrega y penalización:

Hasta el Lunes 18 de Noviembre, 23:59 horas	0 %
Hasta el Martes 19 de Noviembre, 23:59 horas	20 %
Hasta el Miércoles 20 de Noviembre, 23:59 horas	40 %
Hasta el Jueves 21 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

  `http://costa.ls.fi.upm.es/entrega`

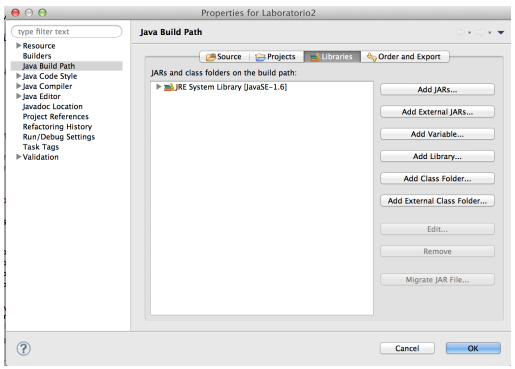
- Los ficheros (2) que hay que subir son `HashTable.java` y `Person.java`.

Configuración previa

- Arrancad Eclipse
- Si trabajáis en portátil, podéis utilizar cualquier versión reciente de Eclipse. Es suficiente con que instaléis la *Eclipse IDE for Java Developers*.
- Cambiad a "Java Perspective".
- Debéis tener instalado al menos Java JDK 8.
- Cread un proyecto Java llamado `aed`:
  - Seleccionad separación de directorios de fuentes y binarios.
  - **No debéis elegir la opción de crear el fichero** `module-info.java`
- Cread un *package* `aed.hashtables` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios y Entregas Individuales → Laboratorio 4 → `Laboratorio4.zip`; descomprimidlo
- Contenido de Laboratorio4.zip:
  - `HashTable.java`, `Person.java`, `HashEntry.java`, `TesterLab4.java`

Configuración previa

- Importad al paquete `aed.hashtables` los fuentes que habéis descargado (`HashTable.java`, `Person.java`, `HashEntry.java`, `TesterLab4.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios y Entregas Individuales).

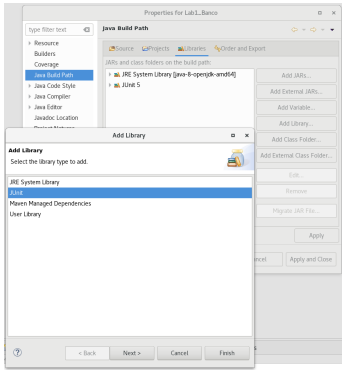


Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda
- Usad la opción “Add External JARs...”.
- Si vuestra instalación distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

Configuración previa

- Añadid al proyecto `aed` la librería `JUnit 5`



Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda;
- Usad la opción “Add Library...” → Seleccionad “Junit” → Seleccionad “JUnit 5”
- Si vuestra instalación distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`
- En la clase `TesterLab4`tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterLab4`, pulsando el botón derecho sobre el editor, seleccionar “Run as...” → “JUnit Test”
- NOTA: Si al ejecutar, no aparece la vista “JUnit”, podéis incluirla en “Window” → “Show View” → “Java” → “JUnit”

Documentación de la librería aedlib.jar

- La documentación de la API de `aedlib.jar` está disponible en

`http://costa.ls.fi.upm.es/entrega/aed/docs/aedlib/`

- Opcionalmente, se puede añadir la documentación de la librería a Eclipse:
  - En el "Package Explorer": "Referenced Libraries" → aedlib.jar y elige la opción "Properties". Se abre una ventana donde se puede elegir "Javadoc Location" y ahí se pone como "javadoc location path:"

`http://costa.ls.fi.upm.es/entrega/aed/docs/aedlib/`

y presionar el botón "Apply and Close"

Tarea de hoy (1): implementar un *map*

- Un *map* es una estructura de datos que guarda valores (de tipo `V`) asociados a claves (de tipo `K`)
- Interfaz `Map`:

```
public interface Map<K,V> {
    public boolean isEmpty();
    public int size();
    public boolean containsKey(Object key);
    public V get(K key);
    public V put(K key, V value);
    public V remove(K key);
    public Iterable<K> keys();
    public Iterable<Entry<K,V>> entries();
}
```

- **Tarea**: completar `HashTable.java`, que implementa un *Map*, usando una "Hash Table" (una tabla de dispersión) con "separate chaining" para resolver las colisiones – consulta el guión y las transparencias para más detalles

Ejemplo

```
HashTable<Integer,String> h =
   new HashTable<Integer,String>(5);  // size 5, key Integer,
                                      // value String
h.size();          =>  0       // returns number of entries
h.put(2,"hola"); =>  null    // no previous value for key 2
h.put(2,"hi");     =>  "hola" // returns the previous value
                               // for key 2
h.size();          =>  1       // only one value per key
h.get(3);          =>  null    // no value with key 3 exists
h.get(2);          =>  "hi"
h.remove(2);       =>  "hi"    // returns the value associated
                               // with the removed key
h.size();          =>  0       // The entry with key 2 was removed

h.put(1,"dulce"); h.put(5,"navidad");

Iterable<Integer> it = h.keys(); // returns iterable over keys
for(Integer i : h.keys()) {print(i);}   // prints 1 5
```

Tarea de hoy (1): implementar una "Hash Table"

- El atributo `table` dentro `HashTable.java` almacena los datos del "hash table", y es del tipo:

  ```
  private NodePositionList<Entry<K,V>>[] table;
  ```

  Un array cuyos elementos son *listas* de "entries" `Entry<K,V>` que asocian una clave (tipo `K`) con un valor (tipo `V`)

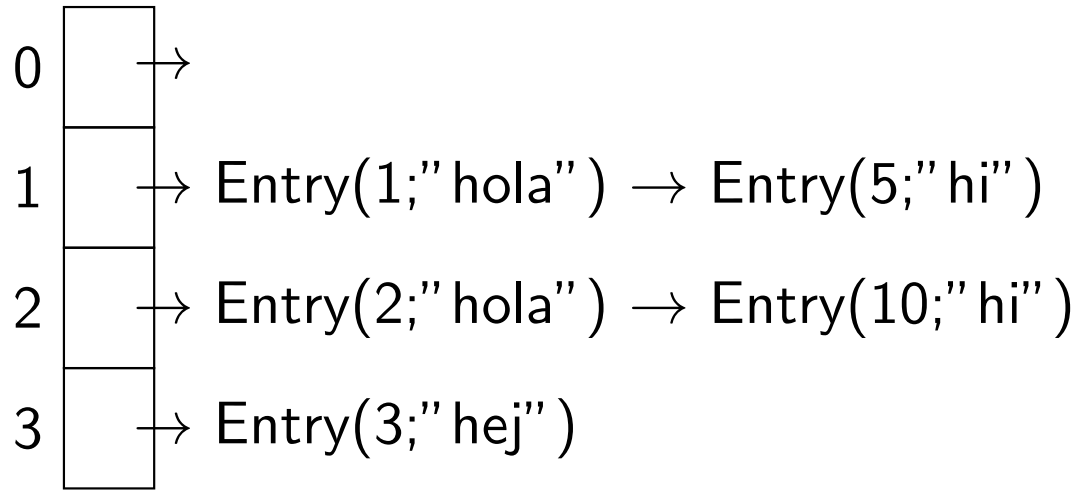
- La interfaz `Entry` tiene los métodos:

  ```
  public interface Entry<K,V> {
    public K getKey();
    public V getValue();
  }
  ```

- Usad la clase `HashEntry` para crear nuevas entries.

Ejemplo Hash Table

- Suponemos que la tabla tiene tamaño 4, y las claves son de tipo `Integer` y los valores de tipo `String`:

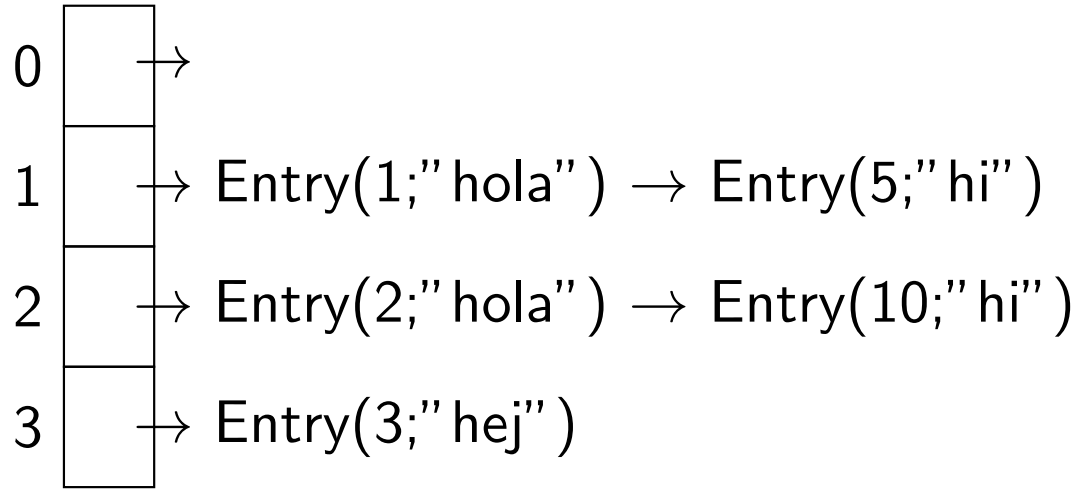


- Cada elemento del array es una lista de entries.
  - Recordad que las listas se usan para resolver las *colisiones* de dos claves en la misma posición del array
- Usad el método `index(Object key)` para aplicar la función de compresión y calcular la posición del array para una entry
- La función de compresión a aplicar en `index(Object key)` es (**hay que completar el método en HashTable.java**):

$$index(key) \equiv abs(key.hashCode()) \ modulo \ tamaño(table)$$

Ejemplo Hash Table

- La tabla tiene tamaño 4:



- $index(key) \equiv abs(key.hashCode())\ modulo\ tamaño(table)$

- Calcular los índices:

Integer(n)	n.hashCode()	index(n)
1	1	1
2	2	2
3	3	3
5	5	1
10	10	2

Tarea de hoy (2): implementar `hashCode` y `equals`

- Todos los objetos de Java tiene un método `hashCode` que devuelve un "hash code" (un int) para el objeto
- **Hay que completar los métodos `hashCode` y `equals` en la clase `Person`** (que se usa en el Tester).
- Una persona tiene un DNI, nombre, y dos apellidos:

```
public class Person {
  private String DNI;
  private String nombre;
  private String apellido1;
  private String apellido2; // ... y getters
}
```

- Usad como `hashCode` para un objeto de tipo `Person` el `hashCode` del atributo `DNI`
- Para implementar `equals` podéis elegir de comparar sólo el atributo `DNI` de dos `Person` (que deberían ser únicos), o comparar todos los atributos de la clase `Person`.

Notas

- Es **obligatorio** usar el atributo `table` para guardar los elementos del Hash table
- Aunque es posible mejorar la implementación de `index` (en `HashTable.java`) **es obligatorio usar la implementación propuesta** ya que el Tester comprueba que se usa dicha implementación
- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar, y debe ejecutar `TesterLab4` 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
- La clases y las interfaces usado en el laboratorio están documentados en `http://costa.ls.fi.upm.es/entrega/aed/docs/hashtables`