Laboratorio A.E.D. Laboratorio 2

Guillermo Román
guillermo.roman@upm.es

Lars-Åke Fredlund
larsake.fredlund@upm.es

Manuel Carro
manuel.carro@upm.es

Marina Álvarez
marina.alvarez@upm.es

Julio García
juliomanuel.garcia@upm.es

Tonghong Li
tonghong.li@upm.es

Sergio Paraíso
sergio.paraiso@upm.es

Juan José Moreno
juanjose.moreno@upm.es

Luis Miguel Danielsson
lm.danielsson@alumnos.upm.es

Normas

- Fechas de entrega y nota máxima alcanzable:

Hasta el miércoles 13 de octubre, 12:00 horas	10
Hasta el jueves 14 de octubre, 12:00 horas	8
Hasta el viernes 15 de octubre, 9:00 horas	6
Hasta el viernes 15 de octubre, 23:59 horas	4
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://vps142.cesvima.upm.es`

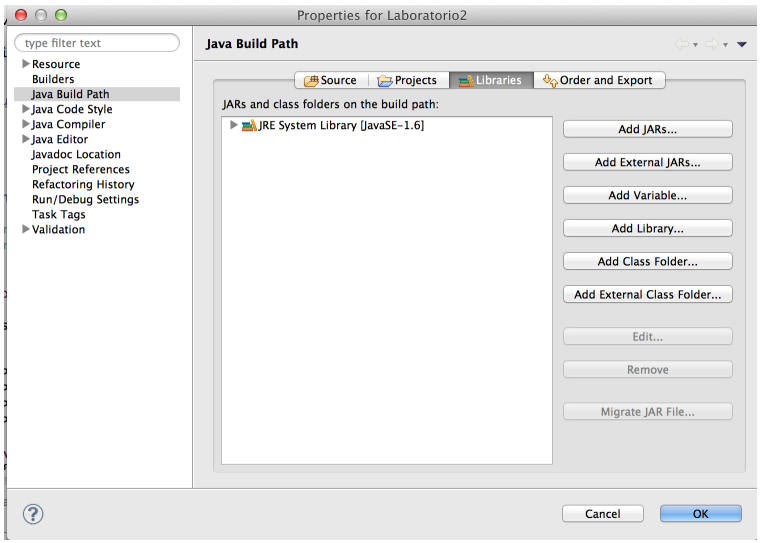
- El fichero que hay que subir es `PositionListOrderedMap.java`.

Configuración previa

- Arrancad Eclipse
- 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.orderedmap` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios → Laboratorio 2 → `Laboratorio2.zip`; descomprimidlo
- Contenido de Laboratorio2.zip:
  - `StringComparator.java`, `EntryImpl.java`, `TesterLab2.java`, `OrderedMap.java`, `PositionListOrderedMap.java`

Configuración previa

- Importad al paquete `aed.orderedmap` los fuentes que habéis descargado (`StringComparator.java`, `EntryImpl.java`, `TesterLab2.java`, `OrderedMap.java`, `PositionListOrderedMap.java`)
- Si no lo habéis hecho, añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios).

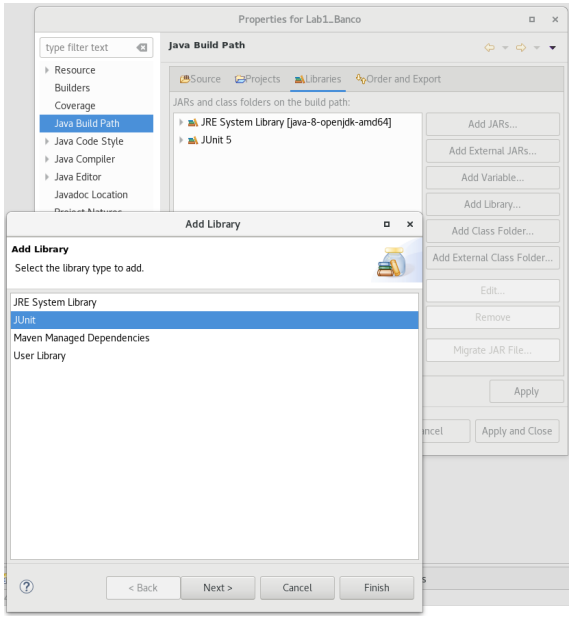


Para ello:

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

Configuración previa

- Si no lo habéis hecho, 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 instalacion distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

- En la clase `TesterLab2` tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterLab2`, 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` esta disponible en `http://costa.ls.fi.upm.es/~entrega/aed/docs/aedlib/`
- Tambien se puede añadir la documentación de la librería a Eclipse (*no es obligatorio*): 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 buton "Apply and Close"

Map<K,V>

Map

La clase Map<K, V> permite asociar claves (objetos de la clase K) con valores (objetos de la clase V). Implementa funciones finitas:

$$\texttt{Map<K,V>() m} \quad \equiv \quad m : K \to V$$

```
public int size();
public boolean isEmpty();
public V put(K key, V value);
public V get(K key);
public V remove(K key);
...
```

OrderedMap<K,V>

OrderedMap

Una Map que incluye una noción de orden para las claves. En cada objeto Map hay una clave menor, mayor y claves inmediatamente anterior y posterior a otra dada.

Implementar parte de la interfaz `OrderedMap<K,V>`

Acabar la codificación del constructor y de los métodos del interfaz:

```
public interface OrderedMap<K,V> {
  public boolean containsKey(K key);

  public V get(K key);
  public V put(K key, V value);
  public V remove(K key);

  public int size();
  public boolean isEmpty();

  public Iterable<K> keys();

  public Entry<K,V> floorEntry(K key);
  public Entry<K,V> ceilingEntry(K key);
}
```

- La documentación detallada está en el fichero `OrderedMap.java`.

es.upm.aedlib.Entry<K,V>

- Implementa un par de elementos (una clave y un valor).
- Sólo tiene constructor y *getters*. No tiene modificadores.

```
public interface Entry<K,V> {

    /* Returns the entry key. */
  public K getKey();

    /* Returns the entry value. */
  public V getValue();
}
```

- La clase `EntryImpl<K,V>` implementa la interfaz `Entry<K,V>` y tiene un constructor `EntryImpl(K k, V v)`.

Ejemplo de funcionamiento

```
OrderedMap<String, Integer> m =
    new PositionListOrderedMap<String,Integer>(new StringComp());

m.put("Jose", 23);          // m = [(Jose,23)]
m.put("Carlos", 35);        // m = [(Carlos,35),(Jose,23)]
m.put("Miriam", 25);        // m = [(Carlos,35),(Jose,23),(Miriam,25)]
m.remove("Jose");           // returns 23;  m = [(Carlos,35),(Miriam,25)]
m.remove("Jose");           // returns null; m = [(Carlos,35),(Miriam,25)]
m.put("Jose", 27);          // m = [(Carlos,35),(Jose,27),(Miriam,25)]
m.floorEntry("Carlos");     // returns (Carlos,35)
m.floorEntry("Carolina");   // returns (Carlos,35)
m.floorEntry("Blas");       // returns null
m.ceilingEntry("Leo");      // returns (Miriam,25)
m.ceilingEntry("Miriam");   // returns (Miriam,25)
m.ceilingEntry("Nando");    // returns null
```

Requisitos de implementación de `PositionListOrderedMap`

- La clase `PositionListOrderedMap` debe implementar `OrderedMap`:

```
public class PositionListOrderedMap<K,V> implements OrderedMap<K,V>
```

- La clase dispondrá de un único constructor que recibe un comparador `Comparator<K>` que sirve para ordenar las claves.
- Los métodos con un parámetro `K key` (`get`, `put`, ...) deben lanzar la excepción `IllegalArgumentException` si `key` es `null`.
- Usad el atributo `elements` de la clase `PositionList<Entry<K,V>>` para guardar **ordenados, usando el comparador de claves recibido en el constructor,** los pares $\langle$`clave, valor`$\rangle$.
- En un objeto `OrderedMap` no puede haber dos entradas con claves iguales (porque implementa una función).
- Podéis añadir los atributos y métodos privados que consideréis necesarios.

Requisitos generales

- Evitad código repetido o muy similar.
- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar.
- Debe ejecutar `TesterLab2` 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).
- Las entregas se comprueban manualmente antes de dar la nota final.

Nota

La técnica propuesta es poco eficiente. Hay métodos mejores de implementar un `Map` y mantener un orden entre las claves.

Consejos / Recordatorios

- Localizar una `Entry` usando una clave es una operación común. El código que os damos sugiere el método privado

  ```
  Position<Entry<K,V>> findKeyPlace(K key)
  ```

  que devuelve dónde *debería* estar *key* (ver descripción en el código). Es delicada de usar, pero simplifica el código, asi que os recomendamos codificarla y utilizarla.
- También puede ser útil definir el método

  ```
  Position<Entry<K,V>> keyPosition(K key)
  ```

  que devuelva la posición de *key* o **null** si no está. Tiene un uso más limitado.

¡Seguid estos consejos para poder conseguir la calificación máxima!