Laboratorio A.E.D. Ejercicio Laboratorio 2

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

Normas.

- Fechas de entrega y la penalización aplicada a la puntuación obtenida sobre 10:

Hasta el Lunes 9 de octubre, 23:59 horas	0 %
Hasta el Martes 10 de octubre, 23:59 horas	20 %
Hasta el Miércoles 11 de octubre, 23:59 horas	40 %
Hasta el Jueves 12 de octubre, 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

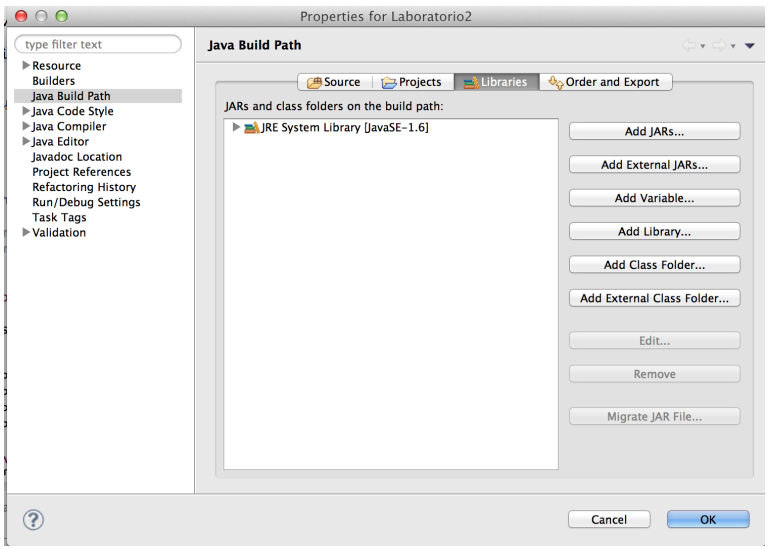
- Todos los ejercicios de laboratorio se deben entregar a través de la web `http://lml.ls.fi.upm.es/~entrega`.
- El fichero que hay que subir es `AsignaturaAdmin.java`.

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.secretaria` en el proyecto `aed`, dentro de `src`
- Aula Virtual $\rightarrow$ AED $\rightarrow$ Laboratorios y Entregas Individuales $\rightarrow$ Laboratorio 2 $\rightarrow$ `Laboratorio2.zip`; descomprimidlo
- Contenido de `Laboratorio2.zip`:
  - `AsignaturaAdmin.java`, `TesterLab2.java`, `Pair.java`, `InvalidMatriculaException.java`

Configuración previa al desarrollo del ejercicio.

- Importad al paquete `aed.secretaria` los fuentes que habéis descargado (`AsignaturaAdmin.java`, `TesterLab2.java`, `Pair.java`, `InvalidMatriculaException.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios y Entregas Individuales). Para ello:
- Project $\rightarrow$ Properties $\rightarrow$ Java Build Path. Se abrirá una ventana como esta:



- Usad la opción "Add External JARs...".
- Intentad ejecutar `TesterLab2.java`

Documentación de la librería aedlib.jar

- La documentación de la API de la librería aedlib.jar esta disponible en `http://lml.ls.fi.upm.es/~entrega/aed/docs/aedlib/`
- También se puede añadir la documentación de la librería a Eclipse (*no es obligatorio*): en el "Package Explorer": "Referenced Libraries" $\rightarrow$ `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://lml.ls.fi.upm.es/~entrega/aed/docs/aedlib/` y presionar el botón "Apply and Close"

Tarea: Implementar la clase `AsignaturaAdmin`

- El objetivo de este laboratorio es desarrollar un sistema para administrar la matriculación de alumnos en una asignatura y la asignacion de notas
- Es similar al laboratorio 1, pero presenta algunas diferencias:
  - Los métodos son similares, pero con diferencias *importantes* en cuanto a sus parámetros, valores devueltos y funcionamiento.
  - Es **obligatorio** trabajar con `PositionList`.
  - Un alumno únicamente se identifica por el núnero de matrícula.
- `AsignaturaAdmin` incluye un atributo

  ```
  private PositionList<Pair<String,Integer>> notas;
  ```

  Se debe usar este atributo para relacionar los alumnos matriculados (los `String`) y las notas (los `Integer`)
  - La clase `Pair` dispone del constructor `Pair(Matr,Nota)` y los "setters" y "getters": `getLeft()`, `getRight()`, etc.
- La documentacion detallada de los métodos a completar se encuentra en el fichero `AsignaturaAdmin.java`

La clase AsignaturaAdmin

```
public class AsignaturaAdmin {
 private String nombreAsignatura;
 private PositionList<Pair<String,Integer>> notas;

 AsignaturaAdmin (String nombreAsignatura) // Constructor

 String getNombreAsignatura()

 PositionList<String> matricular(PositionList<String> matriculas)

 PositionList<String> desmatricular(PositionList<String> matriculas)

 boolean estaMatriculado(String matricula)

 boolean tieneNota(String matricula) throws InvalidMatriculaException

 int getNota(String matricula) throws InvalidMatriculaException

 void setNota(String matricula, int nota) throws InvalidMatriculaException

 PositionList<String> alumnosEnRango(int minNota, int maxNota)

 double notaMedia()
}
```

Reglas de la Implementación

- Está permitido, incluso es recomendable, añadir métodos auxiliares
- NO está permitido añadir nuevos atributos
- NO se debe modificar el contenido de las estructuras de datos recibidas como parámetros en los métodos
- Los métodos que devuelven una lista (p.e., `matricularAlumnos`) deben devolver una lista **nueva**
- El parámetro de entrada `matriculas` nunca será **null** y no contendrá elementos **null** ni elementos duplicados

Consejos:

- Un alumno puede estar matriculado, pero no tener una nota. Recomendamos el uso del valor **null** como nota (dentro el atributo `notas`) para gestionar este caso
- En el caso de una `PositionList`, mantener la estructura ordenada no reduce la complejidad de la búsqueda binaria, con lo que recomendamos, por simplicidad de código, no mantener la lista ordenada

Ejemplos

```
AsignaturaAdmin a = new AsignaturaAdmin("AED");

a.notaMedida()              ---> devuelve 0

a.matricular(["11","22"]) ---> notas contiene:
                               [Pair("11",null),Pair("22",null)]
                               devuelve
                               ["11","22"]

a.setNota("22",5)           ---> notas contiene:
                               [Pair("11",null),Pair("22",5)]

a.matricular(["33","11"]) ---> notas contiene:
                               [Pair("11",null),Pair("22",5),
                                Pair("33",null)]
                               devuelve
                               ["33"]

a.setNota("33",2)           ---> notas contiene:
                               [Pair("11",null),Pair("22",5),
                                Pair("33",2)]

a.notaMedia()               ---> devuelve 3.5       (5+2 / 3)
```

Ejemplos

```
a.estaMatriculado("22")    ---> devuelve true
a.tieneNota("22")          ---> devuelve true
a.getNota("22")            ---> 5

a.estaMatriculado("11")    ---> devuelve true
a.tieneNota("11")          ---> devuelve false
a.getNota("11")            ---> InvalidMatriculaException

a.estaMatriculado("00")    ---> devuelve false
a.tieneNota("00")          ---> InvalidMatriculaException
a.getNota("00")            ---> InvalidMatriculaException

a.desmatricular(["11","44","33"]) --->
                notas contiene: [Pair("22",5),Pair("33",2)]
                devuelve:       ["11"]

a.alumnosEnRango(1,4)      ---> ["33"]   (como 1 <= 2 <= 4)
```

Notas

- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar
- Debe ejecutar `TesterLab2.java` 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