Laboratorio A.E.D. Ejercicio Individual 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 penalización:

Hasta el Martes 20 de septiembre, 12:00 horas	0
Hasta el Miércoles 21 de septiembre, 12:00 horas	20 %
Hasta el Jueves 7 22 septiembre, 12:00 horas	40 %

  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

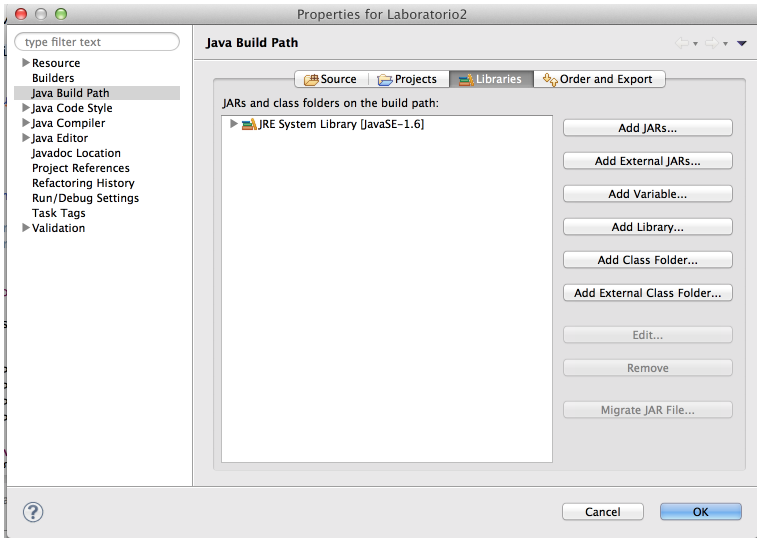
- Todos los ejercicios de laboratorio se deben entregar a través de `https://deliverit.fi.upm.es`
- El fichero que hay que subir es `IndexedListCheckSumUtils.java`.

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.invididual2` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios → Individual 2 → Individual2.zip; descomprimidlo
- Contenido de Individual2.zip:
  - `TesterInd2.java`, `IndexedListCheckSumUtils.java`

Configuración previa

- Importad al paquete `aed.invididual2` los fuentes que habéis descargado (`TesterInd2.java`, `IndexedListCheckSumUtils.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios).

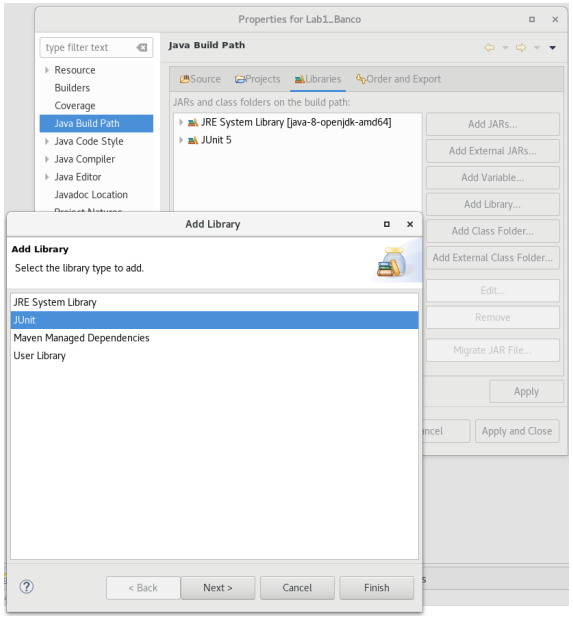


Para ello:

- 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

- 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 `TesterInd2` tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterInd2`, 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/teaching/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/teaching/aed/docs/aedlib/`

    y presionar el buton "Apply and Close"

Tarea 1: Añadir "checksums" a una lista indexada

- Se pide implementar el método

```
public static IndexedList<Integer>
  indexedListCheckSum(IndexedList<Integer> list,
                      int n)
```

  dentro la clase IndexedListCheckSumUtils (en el paquete `aed.invididual2`) que recibe una lista indexada de enteros `list` y un `int n`.

- El método devuelve una lista indexada **nueva**, que contiene los mismos datos que `list`, pero que después de cada *n* elementos de la lista original añade un elemento "*checksum*".
- El valor de ese "*checksum*" es la suma de los *n* elementos anteriores.
- Si el número de elementos de `list` no es un múltiplo de *n*, se añade un *checksum* al final de la lista nueva sumando los números que hay después del último *checksum*.

Ejemplos

```
indexedListCheckSum([1,4,3],3)   --> [1,4,3,8]
// checksum en posicion 3 (8 == 1+4+3) en la lista nueva devuelta

indexedListCheckSum([1,2,7,4],2) --> [1,2,3,7,4,11]
// checksums en posiciones 2 (3 == 1+2) y 5 (11 == 7+4)

indexedListCheckSum([1,2,7],2)   --> [1,2,3,7,7]
// checksums en posiciones 2 (3 == 1+2) y 4 (7 == 7)

indexedListCheckSum([1,2,3],1)   --> [1,1,2,2,3,3]

indexedListCheckSum([],1)        --> []
// array vacio

indexedListCheckSum([1],88)      --> [1,1]
```

Tarea 2: Comprobar que una lista tiene "checksums"

- Se pide implementar el método

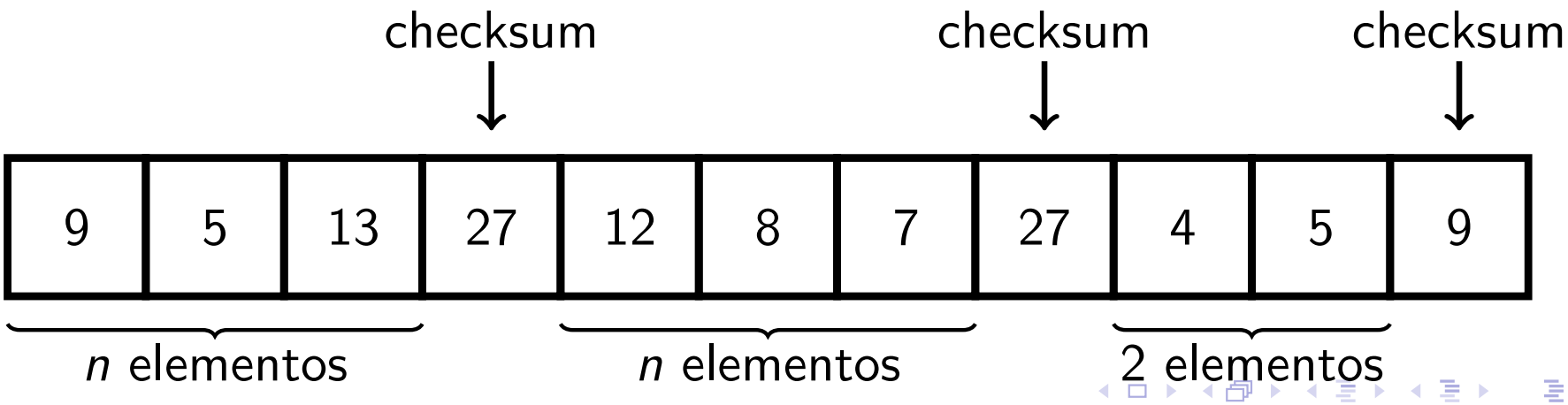
```
public static boolean
  checkIndexedListCheckSum(IndexedList<Integer> list,
                           int n)
```

  dentro la clase IndexedListCheckSumUtils que recibe una lista indexada de enteros `list` y un `int n`.
- El método comprueba que la lista `list` tiene los *checksums* correctos en las en las posiciones correspondientes.

Tarea 2: Comprobar que una lista tiene "checksums"

- Para ser correcto, la lista tiene que poder ser separada en segmentos repetidos de n elementos, y cada uno de estos segmentos siempre estará seguido por su "checksum" (la suma de los elementos anteriores pertenecientes al segmento).
- Sin embargo, el último segmento podría tener menos que n elementos y tener *checksum*.
- El método debe comprobar que todos los checksum, incluyendo el del último segmento, son correctos y devolver `false` si hay alguno que no es correcto.

Un ejemplo de una lista correcta, con $n = 3$, sería:



Ejemplos

```
checkIndexedListCheckSum([1,4,3,8],3)   --> true
// checksum BUENA en posicion 3 (8 == 1+4+3)

checkIndexedListCheckSum([1,4,3,9],3)   --> false
// checksum MALO en posicion 3 (9 != 1+4+3)

checkIndexedListCheckSum([1,4,3],3)   --> false
// falta un checksum despues del elemento 3

checkIndexedListCheckSum([1,2,3,8,9,17,10,10],2) --> true
// checksums BUENAS en posiciones 2 (3 == 1+2), 5 (17 == 8+9) y 7

checkIndexedListCheckSum([],5) --> true

checkIndexedListCheckSum([8,8],5) --> true
// checksum BUENA en posicion 1 (8 == 8)

checkIndexedListCheckSum([1,1,2,2,3,3],1) --> true
// checksums BUENAS en posiciones 1, 3, 5
```

Notas importantes

- El valor de `list` no será `null`
- El parámetro `n` siempre es mayor que cero
- No se debe modificar la estructura de datos `list` recibida como parámetro.
- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar.
- Debe ejecutar `TesterInd1` correctamente sin mensajes de error
- Nota: un test sin mensajes de error **no** significa que el método sea correcto (es decir, que funcione bien para cualquier posible entrada).
- Todos los ejercicios se comprueban manualmente antes de dar la nota final.