Laboratorio A.E.D. Ejercicio Laboratorio 6

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 penalización:

Hasta el martes 30 de noviembre, 12:00 horas	0 %
Hasta el miércoles 1 de diciembre, 12:00 horas	20 %
Hasta el jueves 2 de diciembre, 12:00 horas	40 %
Hasta el viernes 3 de diciembre, 12:00 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://vps142.cesvima.upm.es`

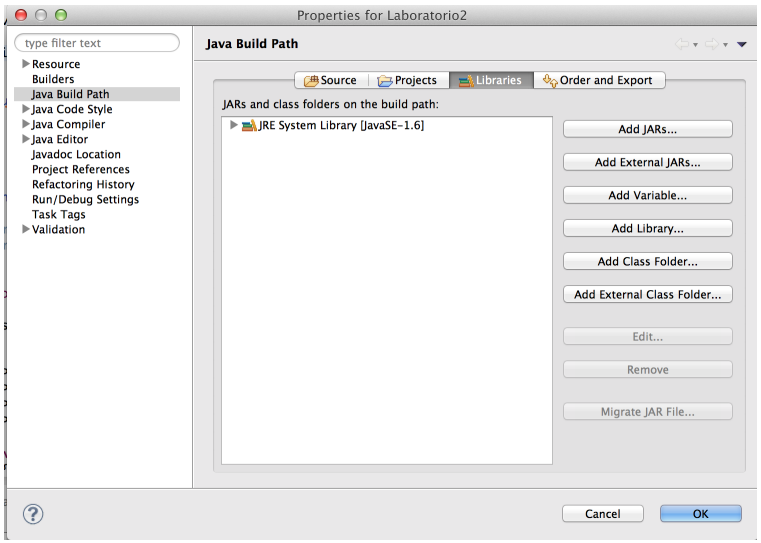
- Los ficheros que hay que subir es `IncomingFlightsRegistry.java`, `Tests.java`.

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.airport` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios y Entregas Individuales → Laboratorio 6 → `Laboratorio6.zip`; descomprimidlo
- Contenido de Laboratorio6.zip:
  - `TesterLab6.java`, `FlightArrival.java`, `IncomingFlightsRegistry.java`, `Tests.java`

Configuración previa

- Importad al paquete `aed.airport` los fuentes que habéis descargado (`TesterLab6.java`, `FlightArrival.java`, `IncomingFlightsRegistry.java`, `Tests.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios y Entregas Individuales).

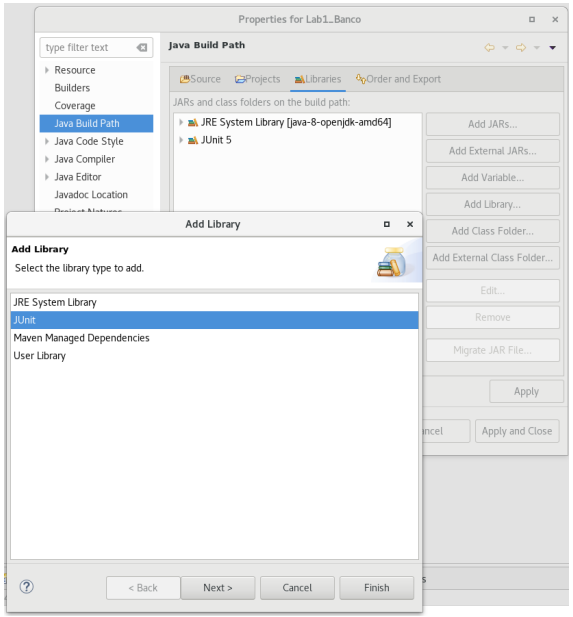


Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda
- Usad la opción “Add External JARs...”.
- Si vuestra 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 `TesterLab6` tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterLab6`, 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/`
- 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"

Tarea 1: Un registro para aviones que van a aterrizar

Terminar la clase `IncomingFlightRegistry` que implementa un registro de aviones llegan a un aeropuerto para aterrizar.

- Registro: recibe información sobre la hora estimada de llegada de los aviones.
- Torre de control: usa el registro para autorizar aterrizajes o retrasarlos si hay conflictos.

La clase `IncomingFlightRegistry`

Métodos a implementar:

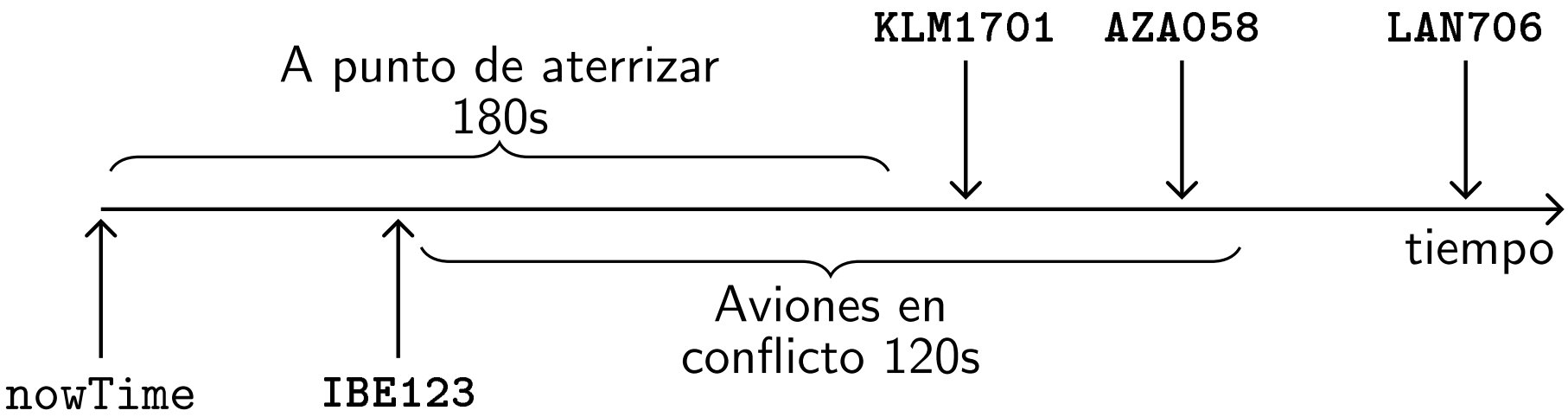
- **void** `arrivesAt(`**String** `flight,` **long** `arrivalTime)`
  Un avión (`flight`) llama a esta método para anunciar su hora de llegada estimada al aeropuerto. El método añade el avión al registro o cambiar su hora de llegada si había sido registrado previamente.
- `Long arrivalTime(`**String** `flight)`
  Devuelve la hora estimada de llegada del avión o `null` si el avión no esta registrado.
- **void** `flightDiverted(`**String** `flight)`
  El avión se ha redirigido a otro aeropuerto y hay que borrarlo del registro si existe.
- `PositionList<FlightArrival> arriving(`**long** `nowTime)`
  La torre de control llama a este método para determinar qué aviones están próximos a aterrizar. Ver siguiente transparencia.

El método arriving(long nowTime)

```
PositionList<FlightArrival> arriving(long nowTime)
```

- Un avión está *a punto de aterrizar* si la diferencia entre `nowTime` y su hora de llegada es menor o igual que 180 (segundos).
- Dos aterrizajes están *en conflicto* si la diferencia entre sus horas de llegada es menor o igual que 120 (segundos).
- Si hay un avión *a punto de aterrizar*, debe devolverse una lista con el avión que aterrice primero como primer elemento, seguido por los aviones cuyos aterrizajes están en conflicto con el del primero en orden de hora de llegada. Si no hay avión *a punto de aterrizar*, devuelve la lista vacía.
- El método borra del registro los aviones devueltos en la lista.

arriving(nowTime): ejemplo



- El primer avión a punto de aterrizar es **IBE123** – esta en el intervalo $[\texttt{nowTime}, \texttt{nowTime} + 180s]$.
- Hay dos aviones en conflicto con el aterrizaje de **IBE123**: **KLM1701** y **AZA058** – están el intervalo $[\texttt{IBE123}, \texttt{IBE123} + 120s]$.
- **LAN706** no esta en conflicto con el aterrizaje de **IBE123**.
- El método devuelve la lista [**IBE123**,**KLM1701**,**AZA058**].

La clase FlightArrival

```
package aed.airport;
import es.upm.aedlib.Pair;

public class FlightArrival extends Pair<String,Long> {
  public FlightArrival(String flight, long time) {
    super(flight, time);
  }

  public String flight() {
    return getLeft();
  }

  public long arrivalTime() {
    return getRight();
  }
}
```

Ejemplo

```
IncomingFlightsRegistry aeropuerto =
  new IncomingFlightsRegistry();

// Vuelo "IBE123" aterriza en la hora 1000
aeropuerto.arrivesAt("IBE123",1000);
aeropuerto.arrivalTime("IBE123");  => 1000

// No hay info sobre KLM123
aeropuerto.arrivalTime("KLM123");  => null

// Vuelo "IBE123" aterriza en la hora 1200 (retraso)
aeropuerto.arrivesAt("IBE123",1200);
aeropuerto.arrivalTime("IBE123");  => 1200

// Vuelo "IBE123" mandado a otro aeropuerto
// Despues no se saba nada de "IBE123"
aeropuerto.flightDiverted("IBE123");
aeropuerto.arrivalTime("IBE123");  => null
```

Ejemplo continuado

```
// Vuelo "IBE345" aterriza en la hora 1000, y KLM111 1400.
aeropuerto.arrivesAt("IBE345",1000);
aeropuerto.arrivesAt("KLM111",1400);
// Vuelo "IBE789" aterriza en la hora 10500
aeropuerto.arrivesAt("IBE789",1050);

// Ningun vuelo llega en el interval 400..580
aeropuerto.arriving(400); => []

// Cuando la hora es 990, hay un avion IBE345 que aterriza
//  proximamente. Y hay otro avion IBE789 en conflicto.
aeropuerto.arriving(990); =>
 [FlightArrival("IBE345",1000), FlightArrival("IBE789",1050)])

// Despues de llamar a arriving Solo queda KLM111:
aeropuerto.arrivalTime("IBE345"); => null
aeropuerto.arrivalTime("IBE789"); => null
aeropuerto.arrivalTime("KLM111"); => 1400
```

Importante: una implementación *eficiente*

- Para conseguir la máxima puntuación es necesario asegurar que la complejidad en caso peor para n aviones registrados es:

`arrivesAt`	$O(log\ n)$
`arrivalTime`	$O(1)$
`flightDiverted`	$O(log\ n)$
`arriving`	$O(log\ n)$

  Es preciso elegir bien las estructuras de datos que implementan el registro. Asumid que:
  - Hay un número máximo, constante, de aviones en conflicto con cualquier avión que esté a punto de aterrizar.
  - Las tablas de dispersión tiene complejidad $O(1)$ en sus operaciones básicas (`put` y `get`)
- En la corrección se valorará el uso de memoria de la implementación y que no reimplemente estructuras de datos básicas.
- Puede ser una buena idea implementar el registro usando una combinación de dos estructuras de datos.

Tarea 2: Aprender a usar JUnit para probar APIs

- La segunda tarea de hoy es aprender a usar la librería JUnit 5 `https://junit.org/junit5/` para comprobar el comportamiento de un API.
- Concretamente, hay implementar dos pruebas en el fichero `Tests.java` para comprobar la funcionalidad del registro de aviones.

Propiedad #1

1. Si un avión `"avion"` anuncia su llegada a la hora 1050 haciendo una llamada a `arrivesAt`,
2. y después cambia su hora de llegada anunciada a la hora 1200 haciendo una llamada a `arrivesAt`
3. comprueba que la hora devuelta por la llamada `arrivalTime("avion")` es 1200

Propiedad #2

1. Si un avión `"avion1"` anuncia su llegada a la hora 20 haciendo una llamada a `arrivesAt`,
2. y después otro avión `"avion2"` anuncia su llegada a la hora 10 haciendo una llamada a `arrivesAt`,
3. la llamada a `arriving(0)`, devuelve una lista con el avion `"avion2"` como primer elemento.

Escribiendo pruebas para Junit5

- Cada prueba se implementa en un método `public` de tipo `void`.
- Antes de la cabecera de un método de prueba se pone una linea con la anotación "@Test".
- Para comprobar que el valor devuelto por un método es correcto, y señalar un error si no lo es, se puede llamar al método

  `assertEquals(Object expected, Object actual)`

  donde `expected` es el valor correcto y `actual` es el valor devuelto
- Es necesario importar los paquetes:

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
```

- Hay muchas más aserciones. Para conocerlas: `https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html`.

Ejemplo de una Prueba

Como ejemplo, implementamos la prueba de que

> *"Si se registra un avión y después se cancela, no tiene hora de llegada"*

```
@Test
public void testRegistryCancel() {
 IncomingFlightsRegistry airport = new IncomingFlightsRegistry();
 airport.arrivesAt("IBE3835",3600);
 airport.flightDiverted("IBE3835");

 // Check that there is no entry for IBE3835 in registry
 assertEquals(null, airport.arrivalTime("IBE3835"));
}
```

Notas

- Solo esta permitido el uso de estructuras de datos disponible en la biblioteca `aedlib` – **no** las de `java.util`.
- Debe ejecutar `TesterLab6` correctamente y 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