Laboratorio A.E.D. 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

Normas

- Fechas de entrega y penalización asociada:

Hasta el viernes 9 de diciembre, 23:59 horas	0 %
Hasta el lunes 12 de diciembre, 23:59 horas	20 %
Hasta el martes 13 de diciembre, 23:59 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

`http://deliverit.fi.upm.es`

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

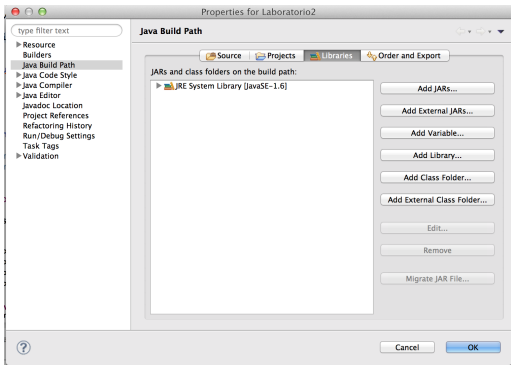
Configuración previa

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

Configuración previa

IMPORTANTE: Actualizad `aedlib.jar` a la última versión

- Importad al paquete `aed.urgencias` los fuentes que habéis descargado
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios y Entregas Individuales).

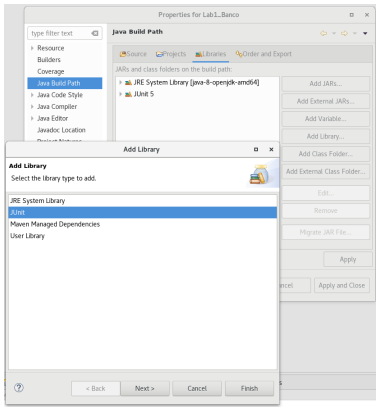


Para ello:

- 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`



  - 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/teaching/aed/docs/aedlib/`

- También 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: Organizar un servicio de Urgencia

- Hay que implementar un sistema informático para manejar las colas de espera para recibir atención médica en un servicio de urgencias de un hospital.
- El interfaz `Urgencias` detalla los métodos a implementar.
- La primera tarea es *implementar* la interfaz `Urgencias` en una clase `UrgenciasAED.java` nueva que tenéis que crear. La clase debe estar en un fichero *nuevo*, `UrgenciasAED.java`.

Paciente

Un `Paciente` tiene los atributos y *getters* y/o *setters*:

- un DNI
- una prioridad, es decir, cómo de urgente es su atención, desde 0 (lo más urgente) a 10 (lo menos urgente)
- dos "timestamps" – `tiempoAdmision` y `tiempoAdmisionEnPrioridad` que guardan la hora de llegada del paciente a urgencias, y el momento en el que se estableció la prioridad (serán iguales en el momento de la admisión)

```
public class Paciente {
  private String DNI;
  private int prioridad;
  private int tiempoAdmision;
  private int tiempoAdmisionEnPrioridad;

  public Paciente(String DNI, int prioridad,
                  int tiempoAdmision,
                  int tiempoAdmisionEnPrioridad) { ... }
}
```

Completar la clase Paciente

- Falta completar los siguientes métodos en la clase Paciente:

  public int compareTo(Paciente paciente) { ... }
  public boolean equals(Object obj) { ... }
  public int hashCode() { ... }

- Usad solo el `DNI` para comparar pacientes en `equals`, y para calcular `hashCode()`.
- Para comparar la urgencia de atender pacientes en `compareTo` se usa:
  - primero la prioridad (prioridad numericamente menor es mas urgente),
  - y si son iguales, el tiempo de admisión en la prioridad (mas tiempo de espera es mas urgente),
  - y si son iguales, el tiempo de admisión en las urgencias.
- Si es mas urgente atender a un paciente `p1` que un paciente `p2`, entonces `p1.compareTo(p2)` debe devolver un valor negativo (y positivo si `p2` es mas urgente que `p1`).

Implementación de la clase `Urgencias`

- **Para esta tarea la eficiencia se considera fundamental.**
- En `Urgencias.java` se documentan los métodos a implementar, y la **complejidad teórica máxima permitida, en el peor de los casos (upper-bound)**.
- Consideramos que las operaciones `put` y `get` en las tablas hash tienen complejidad $O(1)$.
- Varios métodos de las urgencias tienen un parámetro `int hora` que representa la hora en la que se realiza la llamada al método

La interfaz `Urgencias`

- Operaciones con máximo $O(log\ n)$:

```
public interface Urgencias {
  // Admitir un nuevo paciente en urgencias
  public Paciente admitirPaciente(String DNI, int prioridad, int hora)
    throws PacienteExisteException;

  // Paciente sale de las urgencias (sin ser atendido). El paciente se
  // borra de la estructuras de datos de las urgencias.
  public Paciente salirPaciente(String DNI, int hora)
    throws PacienteNoExisteException;

  // Se atende al primer paciente en la cola. El paciente se borra de las
  // estructuras de datos de las urgencias.
  public Paciente atenderPaciente(int hora);

  // Se cambia la prioridad de un paciente.
  public Paciente cambiarPrioridad(String DNI, int nuevaPrioridad, int hora)
    throws PacienteNoExisteException;
}
```

La interfaz `Urgencias`

- Operaciones con máximo $O(1)$:

```
public interface Urgencias {

  // Devuelve un paciente buscando usando el DNI.
  public  Paciente getPaciente(String DNI);

  // Devuelve un par con (i) la suma de los tiempos de espera desde la
  // admision hasta ser atendido, para todos los pacientes que han
  // sido atendidos, y (ii) el numero de pacientes atendidos.
  public Pair<Integer,Integer> informacionEspera();
}
```

La interfaz `Urgencias`

- Operaciones con máximo $O(n \log n)$:

```
public interface Urgencias {

  // Aumenta la prioridad de los pacientes que han esperado mas que
  // maxTiempoEspera en su prioridad actual.
  public void aumentaPrioridad(int maxTiempoEspera, int hora);

  // Devuelve un objeto Iterable ordenado segun el orden en la cola
  public Iterable<Paciente> pacientesEsperando();
}
```

Ejemplo

```
Urgencias u = new UrgenciasAED();

u.admitirPaciente("61969645T",6,1);
u.admitirPaciente("82772887P",6,10);
u.admitirPaciente("74939234Y",1,20);

u.atenderPaciente(30);  ==>  "74939234Y"  (prioridad mas alta)
u.atenderPaciente(40);  ==>  "61969645T"  (llegada mas pronto)

u.admitirPaciente("31825348F",9,50);
u.salirPaciente("82772887P",55);
u.atenderPaciente(60);  ==>  "31825348F"  (anterior salio)

u.admitirPaciente("61569231M",9,61);
u.admitirPaciente("91862887R",2,62)
u.cambiarPrioridad("61569231M",0,63);

u.atenderPaciente(70);  ==>  "61569231M"  (prioridad mas alta)
```

Ejemplo

```
Urgencias u = new UrgenciasAED();

u.admitirPaciente("61969645T",6,1);
u.admitirPaciente("82772887P",6,5);
u.admitirPaciente("74939234Y",2,10);

u.cambiarPrioridad("82772887P",1,20);

u.pacientesEsperando(); ==> ["82772887P","74939234Y","61969645T"

u.atenderPaciente(30);  ==> "82772887P"
u.atenderPaciente(40);  ==> "74939234Y"

u.tiempoMedioEspera();  ==> Pair(55,2)

// Aumenta la prioridad para las pacientes que han esperado mas
// de 10 unidades de tiempo; la hora actual es 100.
u.aumentaPrioridad(10,100);
```

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 (test) el comportamiento de APIs.
- Concretamente, hay que añadir algunas pruebas (tests) al fichero `Tests.java` para comprobar determinadas funcionalidades del API de Urgencia.

Propiedades a comprobar #1

1. Comprueba que tras haber admitido a un paciente P_1 y después a un paciente P_2, ambos con la misma prioridad, una llamada a `atenderPaciente()` devuelve el paciente P_1
2. Comprueba que tras haber admitido a un paciente P_1 y después a otro P_2, ambos con la misma prioridad, una llamada al método `atenderPaciente()` devuelve el paciente P_1 primero y una segunda llamada devuelve el paciente P_2
3. Comprueba que después de haber admitido a un paciente P_1 con prioridad 5, y después a un paciente P_2 con prioridad 1, una llamada a `atenderPaciente()` devuelve el paciente P_2

Propiedades a comprobar #2

4. Comprueba que después de admitir a un paciente P_1 y otro P_2, ambos con la misma prioridad, tas una llamada a `salirPaciente()` con el DNI del paciente P_1 como argumento, llamar al método `atenderPaciente()` devuelve el paciente P_2
5. Comprueba que tas admitir a un paciente P_1 y después a un paciente P_2, ambos con prioridad 5, y después haber llamado al método `cambiarPrioridad()` con el DNI de P_2 y la nueva prioridad a 1, una llamada al método `atenderPaciente()` devuelve el paciente P_2

Escribiendo pruebas para Junit5

- Cada prueba se implementa en un método `public` sin argumentos y que no devuelve nada, es decir, de tipo `void`.
- Justo antes de la cabecera del método de prueba se pone una línea con la anotación "@Test".
- Para comprobar que un valor devuelto por un método es correcto (y para notificar de 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 observado (devuelto)
- Es necesario importar los paquetes:

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

- Hay muchas mas "assertions"; si alguien tiene curiosidad: `https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html`.

Ejemplo de una Prueba

"Si admitimos solo a un paciente, una llamada al método `atenderPaciente()` devuelve el mismo paciente":

```
package aed.urgencias;

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

public class Tests {
  @Test
  public void testAdmitir() throws PacienteExisteException
  {
    Urgencias u = new UrgenciasAED();
    u.admitirPaciente("111", 5, 1);
    Paciente p = u.atenderPaciente(10);

    // Check expected DNI ("111") equals observed DNI (p.getDNI())
    assertEquals("111", p.getDNI());
  }
}
```

Notas

- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar, y debe ejecutar `TesterLab6` 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