Laboratorio A.E.D. Laboratorio 6

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

Sergio Paraíso
sergio.paraiso@upm.es

Normas

- Fechas de entrega y penalización:

Hasta el Lunes 2 de Diciembre, 23:59 horas	0 %
Hasta el Martes 3 de Diciembre, 23:59 horas	20 %
Hasta el Miércoles 4 de Diciembre, 23:59 horas	40 %
Hasta el Jueves 5 Diciembre, 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

  `http://costa.ls.fi.upm.es/entrega`

- Los ficheros que hay que subir son `UrgenciasAED.java`,`Tests.java`.

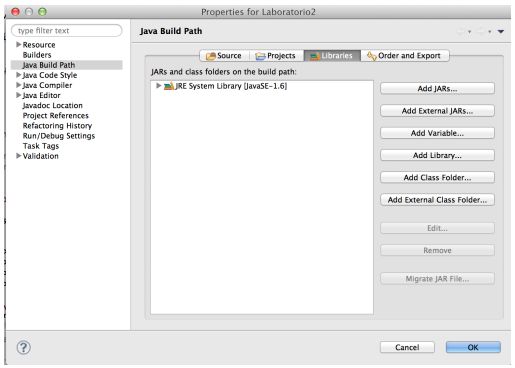
Configuración previa

- 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`, `NoHayPacienteExc.java`, `PacienteYaAdmitidoExc.java`, `Paciente.java`, `PrioridadComparator.java`, `LlegadaUrgenciasComparator.java`, `Time.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).

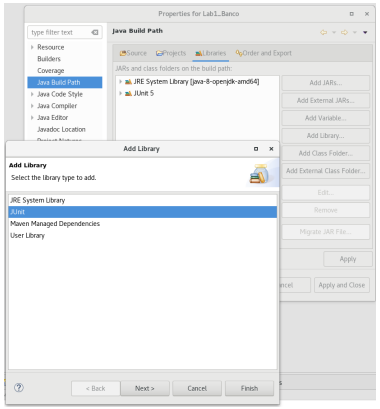


Para ello:

- 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/entrega/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/entrega/aed/docs/aedlib/`

y presionar el buton "Apply and Close"

Tarea 1: Organizar un servicio de Urgencia

- Concretamente hay que implementar un sistema informático para manejar las colas de espera para recibir atención médica en un servicio de urgencia de un hospital.
- El interfaz `Urgencias` detalla los métodos que debéis 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`.

Tarea para hoy: Implementar `Urgencias`

```
public interface Urgencias {
  // Admitir un nuevo paciente en urgencias
  void admitirPaciente(String DNI, int prioridad);
  // El paciente abandona las urgencias o lanza NoHayPacienteExc si no
  // esta el paciente en las urgencias
  void salirPaciente(String DNI) throws NoHayPacienteExc;
  // Un paciente cambia su prioridad y actualiza su hora de llegada
  // a la hora actual o lanza NoHayPacienteExc si no esta el paciente
  void cambiarPrioridad(String DNI, int nuevaPrioridad) throws NoHayPacient
  // Consulta el proximo paciente en ser atendido
  Paciente getProximoPaciente() throws NoHayPacienteExc;
  // Devuelve y elimina de la cola el proximo paciente en ser atendido
  Paciente atenderPaciente() throws NoHayPacienteExc;
  // Aumenta prioridad de pacientes que llevan un cierto tiempo esperando
  void aumentaPrioridad(long maxTiempoEspera);
  // Devuelve la lista de pacientes ordenada por la hora de llegada
  public Iterable<Paciente> getPacientesPorOrdenDeLlegada();
}
```

Pacientes

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

- un DNI
- una prioridad, es decir, cómo de urgente es su atencion, desde 0 (lo más urgente) a 10 (lo menos urgente)
- dos "timestamps" – `horaLlegadaUrgencia` y `horaLlegadaPrioridad` 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)

NOTA: Para actualizar la hora de llegada en la implementación del método `Urgencias.cambiarPrioridad` usad el método estático `Time.currentTimeMillis()` para obtener la hora actual

```
public class Paciente {
  private String DNI;
  private int prioridad;
  private long horaLlegadaAUrgencias;
  private long horaLlegadaAPrioridad;
  public Paciente(String DNI, int prioridad) {...}
}
```

Implementación de la Urgencia (`UrgenciasAED`)

- Es **obligatorio** usar una cola con prioridad para guardar los pacientes con la siguiente declaración:

```
private PriorityQueue<Paciente,...> cola;
```

- Para crear una cola con prioridad de pacientes es necesario definir el orden entre dos pacientes
  - Un paciente p_1 debe ser atendido antes que un paciente p_2 si su prioridad es mayor
  - en caso de tener la misma prioridad, el paciente que haya ha llegado antes a ese nivel de prioridad
- Recomendamos el uso de la clase `PrioridadComparator` al inicializar la cola con prioridad:

```
cola = new PriorityQueue<Paciente,...>
       (new PrioridadComparator());
```

  que implementa ese criterio de ordenación entre pacientes
- Se puede asumir que nunca se llama a `admitirPaciente` con dos DNI iguales.
- Se puede asumir que el parámetro prioridad está entre 0 y 10.

Implementar `salirPaciente` y `cambiarPrioridad`

Para implementar estos métodos tenemos que poder borrar cualquier paciente, o cambiar su prioridad usando su `Entry`.
¿Cómo podemos implementar estos métodos?

Implementar `salirPaciente` y `cambiarPrioridad`

Para implementar estos métodos tenemos que poder borrar cualquier paciente, o cambiar su prioridad usando su `Entry`.
¿Cómo podemos implementar estos métodos?

Opción 1: (lo más ineficiente): saca todos los pacientes de la cola hasta encontrar el paciente buscado, saca de la cola el paciente deseado (vuelve a meterlo con la nueva prioridad en el caso de `cambiarPrioridad`), y vuelve a meter en la cola todos los pacientes que hayas sacado

Implementar `salirPaciente` y `cambiarPrioridad`

Para implementar estos métodos tenemos que poder borrar cualquier paciente, o cambiar su prioridad usando su `Entry`. ¿Cómo podemos implementar estos métodos?

Opción 2: (algo más eficiente): una cola de prioridad implementa `Iterable<Entry<K,V>`. Itera sobre los elementos de la cola hasta que encuentres la `Entry` con el paciente buscado, haz los cambios en el paciente usando sus "setters", y llama despues al método `cola.replaceKey(...)` para recolocarlo en la cola con prioridad

Implementar `salirPaciente` y `cambiarPrioridad`

Para implementar estos métodos tenemos que poder borrar cualquier paciente, o cambiar su prioridad usando su `Entry`.
¿Cómo podemos implementar estos métodos?

Opción 3: (lo más eficiente): crea un atributo extra `Map<String,Entry<K,V>>` que asocia el DNI del paciente (el `String`) con la entry del paciente en la cola (la devuelve el método `enqueue`), y actualiza el map durante llamadas a los métodos `admitirPaciente`, `salirPaciente` y `atenderPaciente`. Usad el DNI para conseguir la `Entry` de un paciente en el map, realiza los cambios mediante sus setters (como en la opción 2), y llama después al método `cola.replaceKey(...)` para recolocarlo en la cola con prioridad

Implementación de la Urgencia (`UrgenciasAED`)

- El método `aumentaPrioridad(long maxTiempoEspera)` aumenta la prioridad en una unidad (si no es la máxima) para *todos* pacientes que han esperado mas que `maxTiempoEspera` en su nivel de prioridad actual.
- El método `getPacientesPorOrdenDeLlegada()` devuelve un iterable que permite iterar sobre todos los pacientes que se encuentran esperando, acorde a su orden de llegada a la urgencias (primero los pacientes que llevan esperando más tiempo).
  - Disponéis de `LlegadaUrgenciasComparator` que implementa dicha comparación

Ejemplo

```
Urgencias u = new UrgenciaAED();

u.admitirPaciente("61969645T",6);
u.admitirPaciente("82772887P",6);
u.admitirPaciente("74939234Y",1);
u.getProximoPaciente();  ==>  "74939234Y" (prioridad mas alta)

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

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

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

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

Ejemplo

```
Urgencias u = new UrgenciaAED();

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

u.getProximoPaciente();  ==>  "74939234Y" (prioridad mas alta)

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

u.getProximoPaciente();  ==>  "82772887P" (prioridad mas alta)

u.getpacientesPorOrdenDeLlegada() ==> ["61969645T","82772887P",
                                       "74939234Y"]
```

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 un numero de pruebas (tests) al fichero `Tests.java` para comprobar algunas funcionalidades del API de Urgencia.

Propiedades a comprobar #1

1. Comprueba que después haber primero admitido un paciente P_1 y después un paciente P_2, ambos con la misma prioridad, una llamada al método `getProximoPaciente` devuelve el paciente P_1
2. Comprueba que después admitir un paciente P_1 y después un paciente P_2, ambos con la misma prioridad, una llamada al método `atenderPaciente` devuelve el paciente P_1 primero, y después, comprueba que una segunda llamada al método `getProximoPaciente` devuelve el paciente P_2
3. Comprueba que después haber admitido un paciente P_1 con prioridad 5, y después un paciente P_2 con prioridad 1, una llamada al método `getProximoPaciente` devuelve el paciente P_2

Propiedades a comprobar #2

4. Comprueba que después haber admitido un paciente P_1 y un paciente P_2, ambos con la misma prioridad, después de una llamada al método `salirPaciente` con el DNI del paciente P_1 como argumento, una llamada al método `getProximoPaciente` devuelve el paciente P_2
5. Comprueba que después haber admitido un paciente P_1 y después 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 `getProximoPaciente` devuelve el paciente P_2

Escribiendo pruebas para Junit5

- Cada prueba se implementa en un método `public` sin argumentos y no devolviendo nada, es decir, de tipo `void`.
- Antes que la cabecera de una método de prueba se pone una linea con la anotación "@Test".
- Para comprobar que un valor devuelto por un método es correcto, y señalizando un error de la prueba 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

Como ejemplo implementamos la prueba de que "Si admitimos un paciente, una llamada al método `getProximoPaciente` devuelve el mismo paciente".

```
@Test
public void testAdmitirGetProximo() throws NoHayPacienteException {
  Urgencias u = new UrgenciaAED();
  admitirPaciente("111", 5);
  Paciente p = getProximoPaciente();

  // Check expected DNI ("111") == 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
- El interfaz `Urgencias` esta documentado en `http://costa.ls.fi.upm.es/entrega/aed/docs/urgencias`