Concurrencia
Prácticas 1 y 2

Grado en Ingeniería Informática/ Grado en Matemáticas e Informática/ 2ble. grado en Ing. Informática y ADE
Convocatoria de Semestre feb–jun 2018–2019

Normas

- La fecha límite de la entrega preliminar (no obligatoria) de la práctica 1 es el **17 de mayo de 2019** a las 23:59:59. Los resultados de someter a más pruebas y revisión de código las entregas anteriores a dicha fecha se publicarán el día **21 de mayo**.
- La fecha límite de la entrega preliminar (no obligatoria) de la práctica 2 es el **31 de mayo de 2019** a las 23:59:59 y los resultados se publicarán el día **4 de junio**.
- Después de estas entregas preliminares revisaremos los sistemas de entrega y publicaremos nuevas pruebas en el sistema de entrega para ambas prácticas.
- **Todas las prácticas**, incluso aquellas en las que no se han detectado problemas, deben ser entregadas de nuevo antes de la fecha límite que será el **10 de junio de 2019** a las 23:59:59.
- La práctica se realiza en grupos de dos alumnos. Los ficheros entregados (`EnclavamientoMonitor.java` y `EnclavamientoCSP.java`) deberán tener un comentario

  ```
  //Grupo: Nombre Alumno 1 (Matrícula Alumno 1), Nombre Alumno 2 (Matrícula Alumno 2)
  ```

  al principio de los ficheros, como definición del grupo. **Nota: sólo entrega un alumno por grupo**, no es necesario que entreguen los dos miembros.
- Deberá mencionarse explícitamente el uso de recursos (código, algoritmos específicos, esquemas de implementación, etc.) que no hayan sido desarrollados por el alumno o proporcionados como parte de asignaturas de la carrera.
- Os recordamos que **todas** las prácticas entregadas pasan por un proceso automático de detección de copias.

1. Enclavamiento

De acuerdo con la Wikipedia en castellano,

> Un enclavamiento es un dispositivo que permite controlar la circulación en una estación de ferrocarril. Es capaz de manejar las señales, los desvíos, los calces[1] y las semibarreras. Además, impide el cambio de los elementos anteriores si la nueva posición se encuentra en una configuración incompatible con la de otro elemento.

En esta práctica vais a tener que escribir un programa concurrente capaz de controlar las señales de un sencillo enclavamiento como el que veis en la figura 1. Dicho enclavamiento protege un cruce de una carretera de un solo sentido (norte) con una vía de tren de un solo sentido (este).

El enclavamiento está conectado a una serie de disponsitivos de control y señalización. A saber:

Tres balizas Una baliza es un dispositivo ubicado en la vía y que permite, entre otras cosas, detectar el paso de un tren. La baliza número 1 se usa para saber que el tren se acerca al cruce, la número 2 para saber que el tren está ya muy cerca del cruce, y la número 3 para saber que el tren ya ha pasado el cruce y se aleja.

[1] Aparato de vía utilizado para inmovilizar o evitar el escape involuntario de vehículos ferroviarios

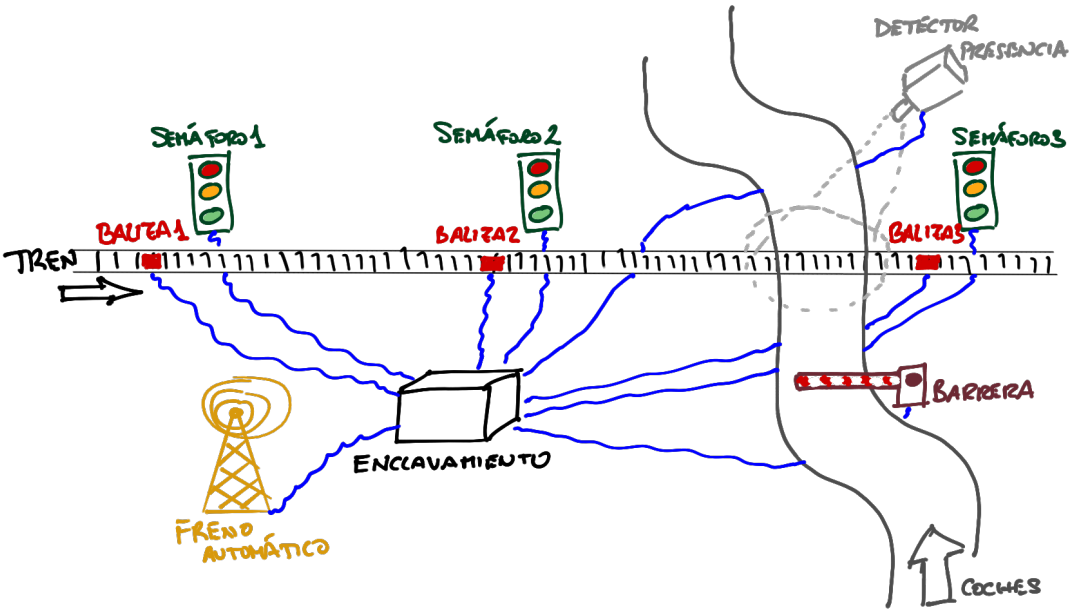

Figura 1: Enclavamiento

Tres semáforos Los semáforos se usan para indicar al maquinista si puede seguir (verde), si debe hacerlo con cuidado (amarillo), o si debe parar el tren (rojo). Cada semáforo (1..3) se ubica un poco después de la baliza correspondiente.

Un detector de presencia Detecta la presencia de coches en el cruce.

Una barrera Ubicada justo antes del cruce, se usa para obligar a los coches a detenerse.

Un sistema de freno automático Puede transmitir una señal de freno automático a los trenes que se encuentren entre las balizas 1 y 3.

La idea es que vuestro programa concurrente evite accidentes. Para ello, además de aplicar sentido común, se deberán seguir algunas reglas concretas. En lo que sigue, denominamos *segmento* al tramo de vía comprendido entre dos balizas consecutivas.

- Los semáforos deberán estar en:

  Verde cuando por delante no hay peligro (tren o coche) a dos segmentos de distancia (ni en el siguiente segmento, ni en el siguiente al siguiente).

  Amarillo cuando por delante hay peligro a dos segmentos de distancia.

  Rojo cuando por delante hay peligro a un segmento de distancia.

  El semáforo 3 siempre estará en verde.

- La barrera deberá permanecer cerrada cuando hay un tren entre la baliza 1 y la 3.

- Se activará el freno automático cuando un tren entre en un segmento con otro tren o coche.

- El programa deberá permitir que los trenes y los coches circulen cuando no hay peligro.

1.1. API de dispositivos

El acceso a los dispositivos de control y señalización se realiza a través de la siguiente API:

```
public interface Control {
  /**
   * Operación bloqueante, el proceso que la llama se queda bloqueado
   * hasta que un tren pasa sobre la baliza n.
  */
  void detectarTren(int n);
  /**
   * Enumerado para los colores de los semáforos.
   */
  public static enum Color {ROJO, AMARILLO, VERDE}
  /**
   * Operación no bloqueante, rápida, enciende la luz indicada en el
   * semáforo n.
   */
  void encenderSemaforo(int n, Color color);
  /**
   * Operación bloqueante, el proceso que la invoca se queda bloqueado
   * hasta que el detector de presencia detecta un cambio con respecto
   * a la presencia indicada.  Por ejemplo, si se invoca con
   * detectarPresencia(true) se bloquea hasta que haya un coche en el
   * cruce, si se invoca con detectarPresencia(false) se bloquea hasta
   * detectar que no hay ningún coche en el cruce.
   */
  void detectarPresencia(boolean presencia);
  /**
   * Operación lenta, el proceso que las invoca queda bloqueado hasta
   * que la barrera está abierta.
   */
  void abrirBarrera();
  /**
   * Operación lenta, el proceso que las invoca queda bloqueado hasta
   * que la barrera está cerrada.
   */
  void cerrarBarrera();
  /**
   * Operación no bloqueante, rápida, envía una señal al tren para
   * accionar o desactivar el freno de cualquier tren entre la baliza
   * 2 y 3
   */
  void accionarFreno(boolean activo);
}
```

1.2. Diseño

Las operaciones que son bloqueantes o lentas marcan en general la necesidad de tener un proceso que las atienda. Para este problema, se han considerado necesarios los siguientes procesos:

- Un proceso por baliza para detectar el paso de los trenes.
- Un proceso por semáforo para controlar el semáforo.
- Un proceso para detectar la presencia o no de coches en el cruce.
- Un proceso para abrir o cerrar la barrera.
- Un proceso para accionar el freno.

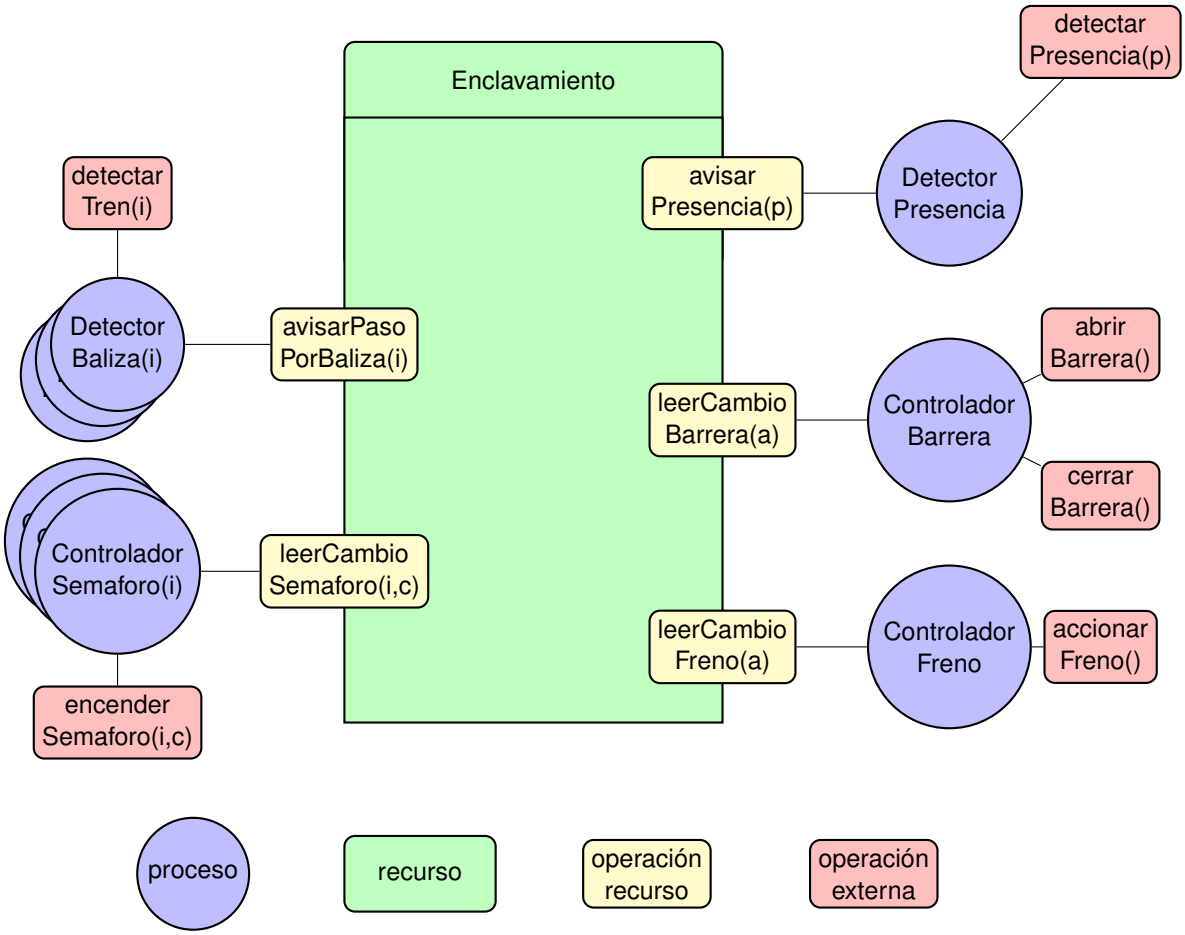

Figura 2: Grafo de procesos/recursos.

En la figura 2 se muestra la arquitectura de la aplicación en forma de grafo de procesos y recursos. Como se puede observar, los procesos se comunicarán a través de un recurso compartido implementado como una instancia de `Enclavamiento` cuya especificación formal puede encontrarse en la sección 1.2.2. El programa concurrente, con el código de los procesos, puedes encontrarlo en `Cruce.java` entre el código de apoyo entregado para la práctica. Mostramos en la siguiente sección, sección 1.2.1 los bucles principales de cada uno de ellos. Inmediatamente después, en la sección 1.2.2, se puede encontrar la especificación formal del recurso compartido.

1.2.1. Código de los procesos

En la figura 3 se muestran las partes más relevantes del código de los diferentes procesos mencionados anteriormente.

1.2.2. Especificación formal del recurso

La especificación formal del recurso compartido se muestra en la figura 4. Algunas explicaciones sobre la representación elegida:

- self.presencia: es un booleano que indica si se ha detectado presencia de coches en el cruce.
- self.tren: es una *tabla*[2] que indica el número de trenes en el segmento despues de la baliza 1, después de la baliza 2 y despues de la baliza 3 (self.tren(1), self.tren(2) y self.tren(3), respectivamente). El valor self.tren(0) es espúreo (aunque de alguna forma indica el número de trenes que

[2] En forma de función total.

Procesos DetectorBaliza

```
while (true) {
  dispositivos.detectarTren(i);
  enclavamiento.avisarPasoPorBaliza(i);
}
```

Procesos ControladorSemaforo

```
Control.Color color = Control.Color.ROJO;
while (true) {
  dispositivos.encenderSemaforo(i, color);
  color = enclavamiento.leerCambioSemaforo(i, color);
}
```

Proceso DetectorPresencia

```
boolean presencia = true;
while (true) {
  dispositivos.detectarPresencia(presencia);
  enclavamiento.avisarPresencia(presencia);
  presencia = !presencia;
}
```

Proceso ControladorBarrera

```
boolean abierta = true;
while (true) {
  if (abierta) {
    dispositivos.abrirBarrera();
  }
  else {
    dispositivos.cerrarBarrera();
  }
  abierta = enclavamiento.leerCambioBarrera(abierta);
}
```

Proceso ControladorFreno

```
boolean accionado = false;
while (true) {
  dispositivos.accionarFreno(accionado);
  enclavamiento.leerCambioFreno(accionado);
  accionado = !accionado;
}
```

Figura 3: Código de las tareas que acceden al enclavamiento y sus dispositivos asociados.

han llegado) pero es muy útil para simplificar las fórmulas en la especificación de las operaciones.

- self.color: es una tabla *tabla* que indica el color esperado para el semáforo 1, 2 y 3 (self.color(1), self.color(2) y self.color(3), respectivamente). El valor self.color(0) es espúreo y no importa su valor.
- Se ha definido al final de la especificación un predicado para especificar los valores correctos de los colores: ColoresCorrectos.

C-TAD Enclavamiento
OPERACIONES
ACCIÓN avisarPresencia: $\mathbb{B}$*[e]*
ACCIÓN leerCambioBarrera: $\mathbb{B}$*[e]*× $\mathbb{B}$*[s]*
ACCIÓN leerCambioFreno: $\mathbb{B}$*[e]*× $\mathbb{B}$*[s]*
ACCIÓN leerCambioSemaforo: *Id[e]* × *Color[e]* × *Color[s]*
ACCIÓN avisarPasoPorBaliza: *Id[e]*

SEMÁNTICA
DOMINIO:
TIPO: $Enclavamiento = (presencia : \mathbb{B} \times tren : Id \to \mathbb{Z} \times color : Id \to Color)$
DONDE: Id = {0,1,2,3}
Color = Rojo | Amarillo | Verde
INICIAL: $\neg$self.presencia $\wedge$ $\forall\, i \in \text{Id} \bullet (\text{self.tren}(i) = 0 \wedge \text{self.color}(i) = Verde)$
INVARIANTE: self.tren(1) $\geq 0 \wedge$ self.tren(2) $\geq 0 \wedge$ ColoresCorrectos

CPRE: Cierto
avisarPresencia(p)
POST: self.presencia = $p \wedge$
self.tren = self^{pre}.tren $\wedge$
ColoresCorrectos

CPRE: $actual \neq (\text{self.tren}(1) + \text{self.tren}(2) = 0)$
leerCambioBarrera(*actual***,** *esperado***)**
POST: self = self^{pre} $\wedge$ $esperado \Leftrightarrow (\text{self.tren}(1) + \text{self.tren}(2) = 0)$

CPRE: $actual \neq (\text{self.tren}(1) > 1 \vee \text{self.tren}(2) > 1 \vee \text{self.tren}(2) = 1 \wedge \text{self.presencia})$
leerCambioFreno(*actual***,** *esperado***)**
POST: self = self^{pre} $\wedge$
$esperado \Leftrightarrow (\text{self.tren}(1) > 1 \vee \text{self.tren}(2) > 1 \vee \text{self.tren}(2) = 1 \wedge \text{self.presencia})$

PRE: $i \neq 0$
CPRE: $actual \neq \text{self.color}(i)$
leerCambioSemaforo(*i***,** *actual***,** *esperado***)**
POST: self = self^{pre} $\wedge$ $esperado = \text{self.color}(i)$

PRE: $i \neq 0$
CPRE: Cierto
avisarPasoPorBaliza(*i***)**
POST: self.presencia = self^{pre}.presencia $\wedge$
self.tren = self^{pre}.tren $\oplus$ $\{i-1 \mapsto \text{self}^{pre}.\text{tren}(i-1) - 1,$
$i \mapsto \text{self}^{pre}.\text{tren}(i) + 1\} \wedge$
ColoresCorrectos

DONDE: ColoresCorrectos =
self.color(1) = Rojo $\Leftrightarrow$ self.tren(1) $> 0 \wedge$
self.color(1) = Amarillo $\Leftrightarrow$ (self.tren(1) = 0 $\wedge$ (self.tren(2) $> 0 \vee$ self.presencia)) $\wedge$
self.color(1) = Verde $\Leftrightarrow$ (self.tren(1) = 0 $\wedge$ self.tren(2) = 0 $\wedge$ $\neg$self.presencia) $\wedge$
self.color(2) = Rojo $\Leftrightarrow$ (self.tren(2) $> 0 \vee$ self.presencia) $\wedge$
self.color(2) = Verde $\Leftrightarrow$ (self.tren(2) = 0 $\wedge$ $\neg$self.presencia) $\wedge$
self.color(3) = Verde

Figura 4: Especificación formal del recurso compartido.

2. Prácticas

2.1. Primera práctica

La entrega consistirá en una implementación del recurso compartido en Java usando la clase `Monitor` de la librería `cclib`. La implementación a realizar debe estar contenida en un fichero llamado `EnclavamientoMonitor.java` que implementará la interfaz `Enclavamiento`.

2.2. Segunda práctica

La entrega consistirá en una implementación del recurso compartido en Java mediante paso de mensajes síncrono, usando la librería JCSP. La implementación deberá estar contenida en un fichero llamado `EnclavamientoCSP.java` que implementará la interfaz `Enclavamiento`.

3. Información general

La entrega de las prácticas se realizará **vía WWW** en la dirección `http://lml.ls.fi.upm.es/entrega`.

El código que finalmente entreguéis (tanto para memoria compartida como para paso de mensajes) no debe realizar **ninguna** operación de entrada/salida.

Para facilitar la realización de la práctica están disponibles en `http://babel.upm.es/teaching/concurrencia` varias unidades de compilación:

- `Enclavamiento.java`: define la interfaz común a las distintas implementaciones del recurso compartido.
- Para poder simular vuestra implementación están disponibles los ficheros:
  - `Control.java` y `ControlSimulado.java` – simulación de los dispositivos de control y señalización
  - `Cruce.java` – programa concurrente de acuerdo al diseño dado (es tarea vuestra cambiar la instanciación de `Enclavamiento` para usar la implementación de monitores o la de paso de mensajes)
- `EnclavamientoCSP.java`: esqueleto para la práctica 2, que incluirá la mayor parte del código *vernacular* de JCSP.
- `PreconditionFailedException.java`: representa una excepcion que vuestra implementacion del recurso deberia lanzar cuando no se cumple una precondition.

Por supuesto durante el desarrollo podéis cambiar el código que os entreguemos para hacer diferentes pruebas, depuración, etc., pero el código que entreguéis debe poder compilarse y ejecutar sin errores junto con el resto de los paquetes entregados **sin modificar estos últimos**. Podéis utilizar las librerías estandar de Java y las librerias auxiliares que estén disponibles para asignaturas previas (p.ej. la librería aedlib.jar de Algoritmos y Estructuras de Datos disponible en `http://lml.ls.fi.upm.es/~entrega/aed/aedlib`), para la definición de estructuras de datos.

El programa de recepción de prácticas podrá rechazar entregas que:

- Tengan errores de compilación.
- Utilicen otras librerías o aparte de las estándar de Java y las que se han mencionado anteriormente.
- No estén suficientemente comentadas. Alrededor de un tercio de las líneas deben ser comentarios **significativos**. No se tomarán en consideración para su evaluación prácticas que tengan comentarios ficticios con el único propósito de rellenar espacio.
- No superen unas pruebas mínimas de ejecución, integradas en el sistema de entrega.