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 2024–2025

Normas

- Os recordamos que **todas** las prácticas entregadas pasan por un proceso automático de detección de copias.

Calendario de entregas y revisiones

28 de mayo 23:59:59	Fecha límite de entrega de la práctica 1
4 de junio 23:59:59	Fecha límite de entrega de la práctica 2

1. Simulador de tráfico en carretera

En esta práctica vamos a escribir un simulador de tráfico en carretera. Nuestro simulador tratará los siguientes aspectos.

- El simulador **simulará** el tráfico en una **única carretera**.
- La carretera tiene **varios carriles**.
- La carretera estará **dividida en segmentos ordenados**.
- Los coches **entran** en la carretera en el **primer segmento** y abandonan la carretera **saliendo del último segmento**.
- Cada coche tendrá un **identificador** y una **velocidad**[1].
- Un coche **no puede entrar** en la carretera si todos los carriles del primer segmento están ocupados.
- Un coche **no puede avanzar** al siguiente segmento si todos los carriles del siguiente segmento están ocupados.
- Un coche **no puede cambiar de carril** dentro de un segmento.
- Un coche **puede cambiar de carril** al avanzar de un segmento al siguiente y puede hacerlo **a qualquier carril**.
- El **tiempo** que un coche está en un segmento depende de la **velocidad** del coche.
- Un coche siempre **puede salir**.

1.1. Diseño

En nuestro programa concurrente, cada coche va a ser simulado por un proceso. Esta decisión es la que marca nuestro diseño:

- Cada proceso que simula un coche ejecutará operaciones sobre un recurso compartido en el que informa de sus *intenciones*. Utilizaremos las operaciones del recurso compartido para *frenar* a los coches cuando no puedan moverse al siguiente punto.
- Además de los procesos *coche* incluiremos un proceso encargado de hacer avanzar el tiempo de la simulación. Este tipo de procesos es habitual en los sistemas de simulación para permitir controlar la velocidad de la simulación.

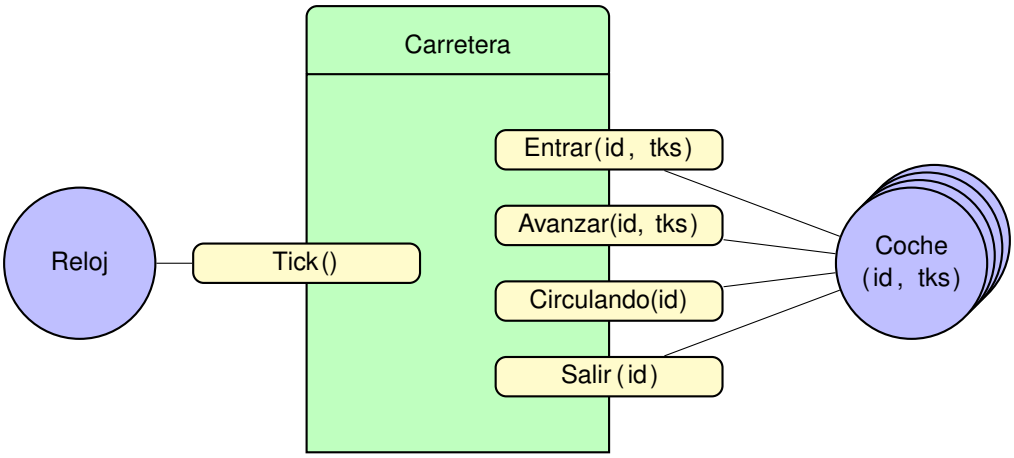

Figura 1: Grafo de procesos/recursos.

En la figura 1 se muestra la arquitectura del programa concurrente 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 Carretera. Las secciones 1.1.1 y 1.1.2 describen los procesos y el recurso compartido.

1.1.1. Código de los procesos

El código de los procesos que mostramos en la figura 3 es una abstracción de lo que se puede ver en el código de apoyo. El código de apoyo tiene llamadas que permiten actualizar la interfaz gráfica, una tarea *irrelevante* para la concurrencia.

Para poder entender el código de los procesos resultará útil consultar la figura 2 con la interfaz en Java del recurso compartido. Los aspectos más relevantes para entender los procesos son:

- Los coches utilizan `entrar` para entrar en el primer segmento: quedan **bloqueados si no hay carril libre**.
- Los coches utilizan `avanzar` para avanzar al siguiente segmento: quedan **bloqueados si no hay carril libre** en el siguiente segmento.
- Para avanzar dentro de un segmento, los coches utilizan `circulando`: quedan **bloqueados**, simulando el avance en el segmento en el que están, **hasta que alcanzan el final del segmento**.
- Los coches indican su velocidad en el segmento con el parámetro `tks` de las operaciones `entrar` y `avanzar`: número de ***ticks* necesarios para atravesar el segmento**.
- Los coches utilizan `salir` para abandonar el último segmento: **nunca quedan bloqueados**.
- El proceso del reloj simplemente genera los *ticks* que marcan la *velocidad* del simulador (en el ejemplo un *tick* por cada 1000 milisegundos). Cada ejecución de `tick` **provoca que los coches *avancen* en sus segmentos** (les quedará un *tick* menos para llegar al final).

1.1.2. Especificación formal del recurso

La especificación formal del recurso compartido se muestra en la figura 4. Las siguientes líneas ofrecen explicaciones sobre la representación elegidas:

- Cada coche estará representado por una cadena de caracteres única: tipo Id.
- El tipo Tks representa *ticks* de reloj: naturales.
- El tipo Pos representa posiciones en la carretera como una tupla con el segmento (componente s desde 1 hasta SEGMENTOS) y el carril ocupados (componente c desde 1 hasta CARRILES).

[1] La velocidad de un coche se dará como el tiempo que un coche tarda en atravesar un segmento.

```java
public interface Carretera {
  /**
   * Un coche pide permiso para entrar en el primer segmento de la
   * carretera con una determinada velocidad
   *
   * @param id identificador del coche
   * @param tks número de ticks necearios para atravesar un segmento (velocidad)
   *
   * @return posición (segmento/carril) que ocupa el coche, ver clase Pos
   */
  Pos entrar(String id, int tks);

  /**
   * Un coche pide permiso para entrar en el siguiente segmento con
   * una determinada velocidad.
   *
   * @param id identificador del coche
   * @param tks número de ticks necearios para atravesar un segmento (velocidad)
   *
   * @return posición (segmento/carril) que ocupa el coche, ver clase Pos
   */
  Pos avanzar(String id, int tks);

  /**
   * Un coche "circula" a lo largo del segmento en el que está. La
   * operación termina cuando el coche ha llegado al final del segmento.
   *
   * @param id identificador del coche
   */
  void circulando(String id);

  /**
   * Un coche abandona el último segmento.
   *
   * @param id identificador del coche
   */
  void salir(String id);

  /**
   * Hace avanzar el tiempo de forma que a cada coche en la carretera
   * le queda un tick menos para llegar al final de su segmento.
   */
  void tick();
}
```

Figura 2: Interfaz en Java del recurso compartido.

Procesos Coche

```
public class Coche extends Thread {
  private Carretera cr;
  private String id;
  private int segmentos;
  private int tks;

  public Coche(Carretera cr,
               String id,
               int segmentos,
               int tks) {
    this.cr = cr;
    this.id = id;
    this.segmentos = segmentos;
    this.tks = tks;
  }

  public void run() {
    cr.entrar(id, tks);
    cr.circulando(id);
    for (int i = 0; i < segmentos; i++) {
      cr.avanzar(id, tks);
      cr.circulando(id);
    }
    cr.salir(id);
  }
}
```

Procesos Reloj

```
public class Reloj extends Thread {
  private static int MS_POR_TICK
    = 1000;
  private Carretera cr;

  public Reloj(Carretera carretera) {
    this.cr = carretera;
  }

  public void run() {
    while (true) {
      try { sleep(MS_POR_TICK); }
      catch (Exception e) { }
      cr.tick();
    }
  }
}
```

Figura 3: Código de los procesos que acceden a la carretera.

- El dominio, y por lo tanto **self**, será una **función parcial** de identificadores de coche en tuplas posiciones/ticks:
  - Si $x \notin$ **dom self** entonces el coche x no está en la carretera.
  - Si **self**$(x) = (y, z)$ entonces el coche x está en la posición y y le quedan z *ticks* para llegar al final del segmento.
  - **self**(x).p.s representa el segmento en el que está el coche x, **self**(x).p.c el carril y **self**(x).t el número de *ticks* que le quedan para llegar al final del segmento.
  - La **invariante** dice que **dos coches no pueden ocupar la misma posición**. Las propiedades invariantes sólo están para entender mejor el recurso por lo que **no es necesario comprobarlas**.
- Las **precondiciones** sólo están para documentar el protocolo que los *coches* **van a respetar**:
  $$\mathsf{entrar(id,tks)}\ \overbrace{\mathsf{circulando(id)\ avanzar(id,\ tks)}\ \ldots\ \mathsf{circulando(id)\ avanzar(id,\ tks)}}^{\text{SEGMENTOS}}\ \mathsf{salir\ (id)}$$
  Por lo tanto **no es necesario comprobar las precondiciones** en la implementación.
- La definición auxiliar CarrilesLibres al final de la especificación, devuelve el conjunto de carriles libres en un segmento (seg) de una carretera (crt).

C-TAD Carretera
ACCIÓN Entrar: Id[e] × Tks[e] × Pos[s]
ACCIÓN Avanzar: Id[e] × Tks[e] × Pos[s]
ACCIÓN Circulando: Id[e]
ACCIÓN Salir: Id[e]
ACCIÓN Tick:

DONDE
TIPO Id = **String**
TIPO Tks = $\mathbb{N} - \{0\}$
TIPO Pos = (s : 1..SEGMENTOS × c : 1..CARRILES)

SEMÁNTICA
DOMINIO
TIPO Carretera = Id $\nrightarrow$ (p : Pos × t : Tks)
INVARIANTE $\forall$ i1 $\in$ **dom self** • $\forall$ i2 $\in$ **dom self** • (**self**(i1).p = **self**(i2).p $\Rightarrow$ i1 = i2)

PRE: id $\notin$ **dom self**
CPRE: |CarrilesLibres(**self**, 1)| > 0
Entrar(id, tks, pos)
POST: pos.s = 1 $\wedge$ pos.c $\in$ CarrilesLibres(**self**^{pre}, 1)
$\wedge$ **self** = **self**^{pre} $\cup$ {id $\mapsto$ (pos, tks)}

PRE: id $\in$ **dom self** $\wedge$ **self**(id).p.s < SEGMENTOS $\wedge$ **self**(id).t = 0
CPRE: |CarrilesLibres(**self**, 1 + **self**(id).p.s)| > 0
Avanzar(id, tks, pos)
POST: pos.s = 1 + **self**^{pre}(id).p.s $\wedge$ pos.c $\in$ CarrilesLibres(**self**^{pre}, pos.s)
$\wedge$ **self** = **self**^{pre} $\oplus$ {id $\mapsto$ (pos, tks)}

PRE: id $\in$ **dom self**
CPRE: **self**(id).t = 0
Circulando(id)
POST: **self** = **self**^{pre}

PRE: id $\in$ **dom self** $\wedge$ **self**(id).p.s = SEGMENTOS $\wedge$ **self**(id).t = 0
CPRE: **cierto**
Salir(id)
POST: **self** = {id} $\lhd$ **self**^{pre}

CPRE: **cierto**
Tick
POST: **dom self** = **dom self**^{pre}
$\wedge$ $\forall$ i $\in$ **dom self** • **self**(i) = (**self**^{pre}(i).p, **max**(0, **self**^{pre}(i).t − 1))

DONDE
CarrilesLibres(crt, seg) $\equiv$ 1..CARRILES − {c $\in$ 1..CARRILES | $\exists$ i $\in$ **dom** crt • crt(i).p = (seg,c)}

Recordatorio sintáctico: $|S|$ es el número de elementos de un conjunto S, $N..M$ es el conjunto de enteros entre N y M ambos incluidos, $F \cup \{X \mapsto Y\}$ es una función como F añadiendo (X, Y), $F \oplus \{X \mapsto Y\}$ es una función como F cambiando la imagen de X por Y, $S \lhd F$ es una función como F quitando los valores de S del dominio.

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 `CarreteraMonitor.java` que implementará la interfaz Carretera.

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 `CarreteraCSP.java` que implementará la interfaz Carretera.

3. Información general

La entrega de las prácticas se realizará **vía WWW** en la dirección

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

Para facilitar la realización de la práctica están disponibles en el moodle de la asignatura varias unidades de compilación:

- `Carretera.java`: define la interfaz común a las distintas implementaciones del recurso compartido.
- `Pos.java`: clase usada para representar posiciones segmento/carril.
- `CarreteraSim.java`: simulador gráfico de una carretera. Es un programa concurrente con procesos que siguen el diseño dado (es tarea vuestra cambiar la instanciación de Carretera para usar la implementación de monitores o la de paso de mensajes, buscad "`new Carretera...`").
- `CarreteraMonitores.java`: esqueleto para la práctica 1.
- `CarreteraCSP.java`: esqueleto para la práctica 2, que incluirá la mayor parte del código *vernacular* de JCSP (se publicará más tarde).
- `cclib-0.4.9.jar`: biblioteca con mecanismos de concurrencia de la asignatura.
- `jcsp.jar`: biblioteca Communicating Sequential Processes for Java (JCSP) de la Universidad de Kent.

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 bibliotecas estandar de Java y las bibliotecas auxiliares que están en el código de apoyo (se incluye `aedlib.jar`, biblioteca de la asignatura Algoritmos y 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.