Concurrencia
Prácticas 1 y 2

Grado en Ingeniería Informática UPM Convocatoria de Segundo semestre 2013-2014

Normas

- La fecha límite de entrega de la práctica 1 es el domingo **25 de mayo de 2014** a las 23:59:59.
- La fecha de límite de entrega de la práctica 2 es el domingo **1 de junio de 2014** a las 23:59:59.
- A partir del **27 de mayo de 2014** publicaremos resultados de someter a pruebas las entregas de la práctica 1, y a partir del **3 de junio de 2014** los de la práctica 2.
- Las prácticas con problemas podrán ser reentregadas hasta el **8 de junio de 2014**.
- 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. Los involucrados en la copia de una práctica tendrán las prácticas anuladas para el año académico en curso.

1. Logística

Una empresa gestiona sus pedidos de forma completamentamente automática: un grupo de robots recorren las naves de almacenaje y recogen los productos que componen cada pedido. Nuestro equipo de desarrollo es el responsable de programar el subsistema que controla el movimiento de los robots entre las naves. El problema ha sido descrito de la siguiente forma:

- El número de naves es `Robots.N_NAVES` numeradas desde `0` hasta `Robots.N_NAVES-1`.
- Cada nave i, excepto la primera, tiene un pasillo de entrada y su salida, excepto la de la última, está conectada al pasillo de entrada de la siguiente nave $(i+1)$ tal y como indica la figura 1.
- Los robots entran en la nave 0 y salen por la nave `Robots.N_NAVES-1` avanzando siempre de una nave a la siguiente.
- Los suelos de cada nave de almacenaje no son capaces de soportar un peso superior a `Robots.MAX_PESO_EN_NAVE` por lo que un robot no debe entrar en una nave si el peso total del mismo, carga incluida, más el peso de todos los robots dentro de la nave supera dicho límite (robot en el pasillo de la nave 2 en la figura 1).
- El proceso de carga se realiza en una zona segura para los suelos de la nave.
- La capacidad de los pasillos está limitada a un robot de tal forma que un robot no puede acceder a un pasillo si éste está ocupado (robots en la nave 1 en la figura 1).

Para poder controlar los movimientos de los robots tenemos a nuestra disposición la siguiente interfaz:

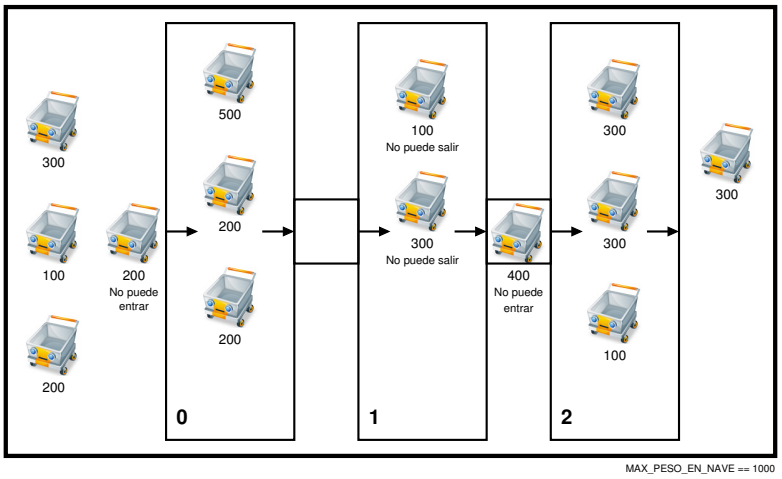

Figura 1: Naves y movimientos de robots

```
class Robots {
  public static final N_ROBOTS = ...;
  public static final N_NAVES = ...;
  public static final MAX_PESO_EN_NAVE = ...;
  public static final PESO_EN_VACIO = ...;
  /* El robot rid entra en el pasillo de la nave nid sin comprobar si
   * se encuentra ocupado. Si el robot no tiene acceso a dicho pasillo
   * (ej. no se encuentra en la nave nid - 1) entonces la orden es
   * ignorada y retorna inmediatamente.  Esta orden termina de
   * ejecutarse cuando el robot ha terminado de entrar en el
   * pasillo. */
  public static void entrarEnPasillo(int rid, int nid) {
    ...
  }
  /* El robot rid entra en la nave nid sin comprobar sobrepeso. Si el
   * robot no estaba en el pasillo de entrada de la nave nid entonces
   * la orden es ignorada y retorna inmediatamente. Esta orden termina
   * de ejecutarse cuando el robot ha accedido a la nave nid. */
  public static void entrarEnNave(int rid, int nid) {
    ...
  }
  /* El robot rid realiza la carga programada en la nave nid y
   * devuelve el peso total del robot (que siempre es inferior a
   * MAX_PESO_EN_NAVE). Si el robot no estaba en la nave nid entonces
   * la orden es ignorada y retorna inmediatamente.  Esta orden
   * termina de ejecutarse cuando el robot tiene toda la carga y se
   * posiciona en una zona segura (el incremento de peso no supone
   * problema para el suelo de la nave). */
  public static int cargar(int rid, int nid) {
    ...
  }
}
```

Vamos a implementar un sistema concurrente para controlar el movimiento de los robots entre las naves. El sistema debe maximizar la eficienca (esencialmente evitando esperas innecesarias) de los robots evitando exceder el peso en las naves y la doble ocupación de los pasillos.

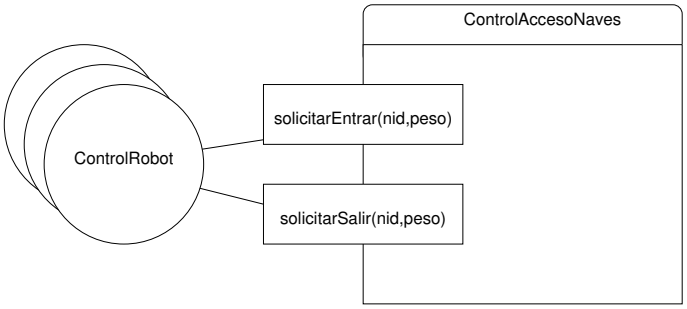

Figura 2: Estructura de procesos

1.1. Diseño

Tendremos un proceso para controlar cada robot. La comunicación y sincronización es responsabilidad de un gestor. La arquitectura del sistema se muestra en la figura 2.

La idea es que el estado interno del recurso compartido sea suficientemente rico como para determinar cuándo los robots pueden progresar. Más concretamente, el estado conoce el peso en cada nave y la ocupación de los pasillos. La especificación del gestor se encuentra en la figura 3.

2. Prácticas

2.1. Primera práctica

La entrega consistirá en una implementación del recurso compartido en Java usando la clase `Monitor` de `cclib`. La implementación a realizar debe estar contenida en un fichero llamado `ControlAccesoNavesMonitor.java` que implementará la interfaz `ControlAccesoNaves`. (ver sec. 3).

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 `ControlAccesoNavesCSP.java` que implementará la interfaz `ControlAccesoNaves`. (ver sec. 3).

3. Información general

El texto de estas prácticas se encuentra en `http://babel.upm.es/teaching/concurrencia`.

La entrega del código 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:

- `Robots.java`: la librería de manejo de los robots.
- `ControlAccesoNaves.java`: define la interfaz común a las distintas implementaciones del recurso compartido.

C-TAD ControlAccesoNaves
OPERACIONES
ACCIÓN solicitarEntrar: *Nave[i]* × *Peso[i]*
ACCIÓN solicitarSalir: *Nave[i]* × *Peso[i]*

SEMÁNTICA
DOMINIO:
TIPO: $ControlAccesoNaves = (peso: Nave \rightarrow Peso \times ocupado: Nave \rightarrow \mathbb{B})$
$Nave = 0 \,..\, N\_NAVES - 1$
$Peso = 0 \,..\, MAX\_PESO\_NAVE$

INICIAL: $\forall\, n \in Nave \bullet \text{self}.peso(n) = 0 \wedge \neg \text{self}.ocupado(n)$
INVARIANTE: $\forall\, n \in Nave \bullet \text{self}.peso(n) \leq MAX\_PESO\_NAVE$

CPRE: $p + \text{self}.peso(n) \leq MAX\_PESO\_NAVE$
solicitarEntrar(n,p)
POST: $\text{self}.peso = \text{self}^{pre}.peso \oplus \{n \mapsto \text{self}^{pre}.peso(n) + p\}$
$\wedge\ (n > 0 \Rightarrow \text{self}.ocupado = \text{self}^{pre}.ocupado \oplus \{n \mapsto \text{Falso}\})$
$\wedge\ (n = 0 \Rightarrow \text{self}.ocupado = \text{self}^{pre}.ocupado\})$

CPRE: $n = N\_NAVES - 1 \vee \neg \text{self}.ocupado(n+1)$
solicitarSalir(n,p)
POST: $\text{self}.peso = \text{self}^{pre}.peso \oplus \{n \mapsto \text{self}^{pre}.peso(n) - p\}$
$\wedge\ (n < N\_NAVES - 1 \Rightarrow \text{self}.ocupado = \text{self}^{pre}.ocupado \oplus \{n+1 \mapsto \text{Cierto}\})$
$\wedge\ (n = N\_NAVES - 1 \Rightarrow \text{self}.ocupado = \text{self}^{pre}.ocupado)$

Figura 3: Especificación del controlador de movimiento entre naves.

- `LogisticaMonitor.java` y `LogisticaCSP.java`: programas principales que crean el recurso compartido y lanzan las tareas de control para cada una de las dos opciones.

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 auxiliares que estén disponibles para asignaturas previas (p.ej. Algoritmos y Estructuras de Datos) para la definición de estructuras de datos auxiliares.

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, similares a las que tenéis en el programa de simulación que os entregamos.