Concurrencia
Prácticas 1 y 2

Grado en Ingeniería Informática/Grado en Matemáticas e Informática Convocatoria de junio 2016/2017
UNIVERSIDAD POLITÉCNICA DE MADRID

Normas

- La fecha límite (recomendada) de entrega de la práctica 1 es el viernes **26 de mayo de 2017** a las 23:59:59. Los resultados de someter a pruebas las entregas anteriores a dicha fecha se publicarán el día **1 de junio**.
- La fecha límite (recomendada) de entrega de la práctica 2 es el viernes **2 de junio de 2017** a las 23:59:59 y los resultados se publicarán el día **8 de junio**.
- Las prácticas con problemas podrán ser reentregadas hasta el **14 de junio de 2017** a las 23:59:59.
- Os recordamos que **todas** las prácticas entregadas pasan por un proceso automático de detección de copias. Los involucrados en la copia de una práctica serán sancionados de acuerdo con la normativa de la Universidad Politécnica de Madrid.

1. RoboFab

En una planta de fabricación de automóviles, los robots se encargan de llevar piezas a la cadena de montaje desde un almacén central. Para ello, depositan las piezas en unos contenedores que se desplazan sobre una cinta transportadora (ver figura).

La idea es que en los contenedores pueden ir piezas de distintos tamaños y pesos, siempre que no se exceda un peso máximo. Varios robots pueden tomar y soltar piezas simultáneamente sobre un mismo contenedor, siempre que este se encuentre en reposo. Cuando un robot quiere soltar una pieza que excedería el peso máximo del contenedor, debe esperar a que la cinta avance y un nuevo contenedor esté disponible. Cuando todos los robots tienen piezas que no entran en el contenedor actual, la cinta avanza para ofrecer un contenedor vacío a los robots.

Los robots aceptan órdenes para ir a tomar piezas y para soltar estas en la zona de descarga. Estas operaciones forman parte de una librería `Robots.java` que no tenéis que implementar:

```
class Robots {
    static final int MAX_ROBOTS = 5;
    static final int MIN_P_PIEZA = 1000;
    static final int MAX_P_PIEZA = 5000;
    // Recoge una pieza e informa de su peso
    // 1 <= idRobot <= MAX_ROBOTS
    // MIN_P_PIEZA <= resultado <= MAX_P_PIEZA
    static int recoger (int idRobot) {
        ...
    }
    // Mueve hasta el punto de descarga y
    // suelta la pieza
    static void soltar (int idRobot) {
        ...
    }
}
```

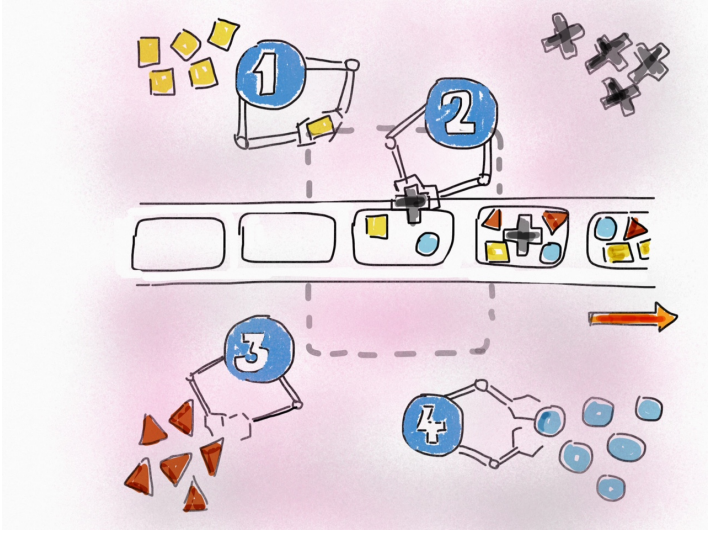

Figura 1: El almacén de piezas y sus incansables robots.

De igual manera, se dispone de una librería para controlar el desplazamiento de la cinta con los contenedores, con una operación para hacer avanzar la cinta:

```
class Cinta {
   static final int MAX_P_CONTENEDOR = 20000;
    // Avanza la cinta hasta que se dispone de un contenedor vacio
    static void avance () {
        ...
    }
}
```

1.1. Diseño

Tendremos un proceso para controlar cada robot y otro más para manejar el reemplazo de contenedores. La comunicación y sincronización es responsabilidad de un gestor. La arquitectura del sistema se muestra en la Figura 2. La idea es que cada proceso `ControlRobot(i)` ejecuta en bucle la secuencia:

```
p = recoger(i); notificarPeso(i,p); permisoSoltar(i); soltar(i);
```

Por su parte, el proceso `ControlCinta` ejecuta en bucle la secuencia:

```
solicitarAvance(); avance(); contenedorNuevo();
```

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 `RoboFabMonitor.java` que implementará la interfaz `RoboFab`. (ver Sec. 3).

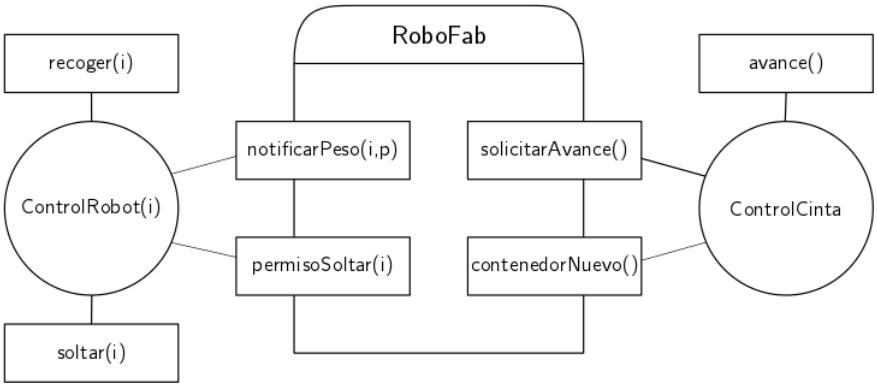

Figura 2: Estructura de procesos

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 `RoboFabCSP.java` que implementará la interfaz `RoboFab`. (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`.

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

- `Robots.java` y `Cinta.java`: las librerías de manejo del hardware.
- `RoboFab.java`: define la interfaz común a las distintas implementaciones del recurso compartido.
- `PlantaMonitor.java` y `PlantaCSP.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**.

El programa de recepción de prácticas podrá rechazar entregas que:

- Tengan errores de compilación.
- Utilicen otras librerías aparte de las estándar de Java.
- No estén suficientemente comentadas. Alrededor de un tercio de las líneas deben ser comentarios **significativos**. (No, no aceptamos comentarios ficticios para rellenar.)
- No sigan la metodología de construcción de código a partir de recursos vista en clase.
- No superen unas pruebas mínimas de ejecución, similares a las que tenéis en el programa de simulación que os entregamos.

C-TAD RoboFab
OPERACIONES
ACCIÓN notificarPeso: *idRobot[i]* × *Peso[i]*
ACCIÓN permisoSoltar: *idRobot[i]*
ACCIÓN solicitarAvance:
ACCIÓN contenedorNuevo:

SEMÁNTICA
DOMINIO:
TIPO: $RoboFab = (peso: \mathbb{N} \times pendientes: idRobot \rightarrow Peso)$
$Peso = MIN\_P\_PIEZA \,..\, MAX\_P\_PIEZA$
$idRobot = 0 \,..\, NUM\_ROBOTS - 1$
INICIAL: $self = (0, pend) \land \forall\, i \in idRobot \bullet pend(i) = 0$
INVARIANTE: $self.peso \leq MAX\_P\_CONTENEDOR$

CPRE: Cierto
notificarPeso(i,p)
POST: $self^{pre} = (p, pends) \land self = (p, pends \oplus \{i \mapsto p\})$

CPRE: $self.peso + self.pendientes(i) \leq MAX\_P\_CONTENEDOR$
permisoSoltar(i)
POST: $self^{pre} = (p, pends) \land self = (p + pends(i), pends \oplus \{i \mapsto 0\})$

CPRE: $self = (p, pends) \land \forall\, i \in idRobot \bullet pends(i) + p > MAX\_P\_CONTENEDOR$
solicitarAvance()
POST: $self = self^{pre}$

CPRE: Cierto
contenedorNuevo()
POST: $self^{pre} = (p, pends) \land self = (0, pends)$

Figura 3: Especificación del controlador del almacén de piezas.