Concurrencia
Prácticas 1 y 2

Grado en Ingeniería Informática UPM/ Grado en Matemáticas e Informática

Convocatoria de Segundo Semestre 2014/2015

Abril de 2015

Normas

- La fecha límite (recomendada) de entrega de la práctica 1 es el martes **19 de Mayo** a las 23:59:59. Los resultados de someter a pruebas las entregas anteriores a dicha fecha se publicarán el día **25 de Mayo**. Se abrirá un nuevo plazo de entrega de la práctica 1 hasta el viernes **29 de Mayo** a las 23:59:59 y los resultados de las entregas hasta esa fecha se publicarán el día **1 de junio**.
- La fecha límite (recomendada) de entrega de la práctica 2 es el sábado **6 de junio** a las 23:59:59 y los resultados se publicacarán el día **8 de junio**.
- Las prácticas (1 y 2) podrán ser entregadas hasta el **17 de junio** a las 23:59:59.
- La práctica se realizará por parejas. El alta de las parejas se realizará medianta la URL `http://lml.ls.fi.upm.es/~entrega/grupocc/` antes del día **10 de mayo** a las 23:59:59.
- 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. Controles de Calidad de una Fábrica

El trabajo consiste en la implementación de un sistema de control automático de calidad de los pedidos de una fábrica. El procesado de los pedidos se realiza a través de unos robots que se encargan de ir aplicando diferentes controles de calidad a cada uno de los pedidos. Cada robot se encarga de aplicar su control y los pedidos deben pasar por todos los controles de calidad para poder ser vendidos. Para garantizar la calidad de los controles, los robots deben pasar por mantenimiento cada cierto número de pedidos procesados. Así mismo, la fábrica dispone de un almacén en el que se colocan los diferentes pedidos que han pasado todos los controles de calidad.

1.1. Diseño

En el sistema informático que hemos diseñado, y que debéis completar, tanto los pedidos, como los robots y los mecánicos son modelados como threads de Java. Así mismo, el almacén también es implementado como un thread adicional. Todos estos threads interactúan con el sistema *Control_Fábrica*, que se encarga de que la concurrencia entre las diferentes operaciones de la fábrica se

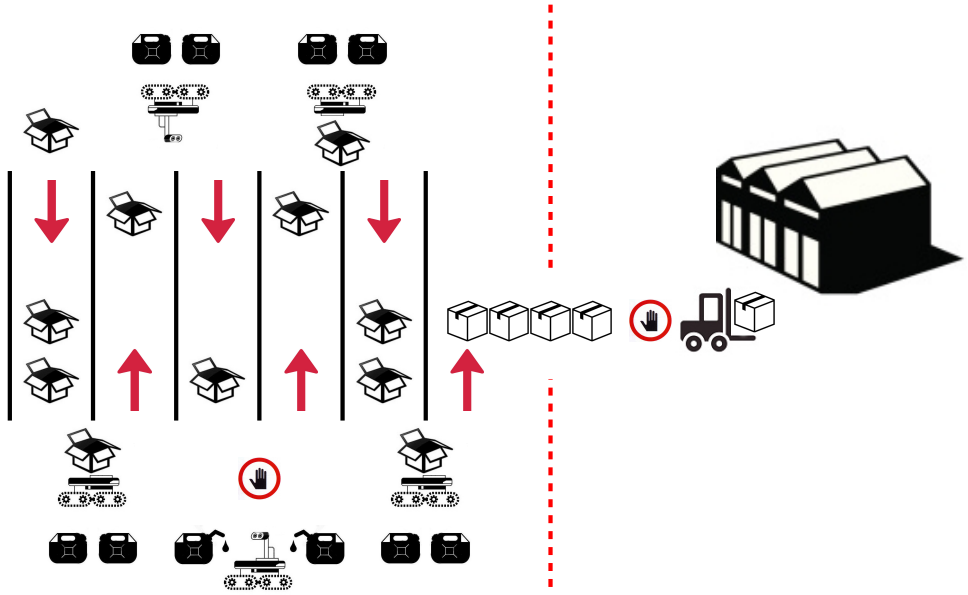
realice correctamente. El sistema de control de la fábrica se ha diseñado como un único recurso compartido, *Control_Fábrica*, que tiene diferentes operaciones que permiten: (1) indicar que un pedido se encuentra preparado para un cierto control, (2) esperar a pasar el control, (3) procesar un pedido por parte de un robot, (4) verificar un robot, y (5) notificar que se ha completado el mantenimiento del robot. El funcionamiento esperado para el recurso es el siguiente:

- La fábrica tiene un total de *N_PEDIDOS*.
- La fábrica tiene un total de *N_ROBOTS*.
- La fábrica tiene tantos mecánicos como robots, es decir, *N_ROBOTS*.
- Cada pedido tiene un identificador *pid* en el rango 0..*N_PEDIDOS* y cada robot un identificador *rid* en el rango 0..*N_ROBOTS*. Los mecánicos son identificados por el id del robot al que se encargan de verificar.
- Los pedidos van notificando que ya se encuentran preparados para ser procesados por un determinado robot.
- Cada robot dispone de una *cola* de pedidos pendientes de procesar. El procesamiento de los pedidos realizará en estricto orden de llegada.
- El pedido esperará a ser procesado por el robot para poder avanzar al siguiente robot.
- Los robots deberán ser verificados por el mecánico cada cierto número de pedidos. El número de pedidos que cada robot puede procesar sin pasar por mantenimiento se encuentra en la secuencia de constantes *N_PROCS*, que tiene longitud *N_ROBOTS* y es accedida mediante el identificador de cada robot, es decir, el valor *N_PROCS*(*rid*), donde $0 \leq rid < N\_ROBOTS$, es el número de pedidos que el robot *rid* puede procesar sin pasar por mantenimiento.
- Mientras un robot está siendo verificado este no podrá procesar pedidos.
- Cuando el proceso de verificación termina, el robot continuará procesando pedidos normalmente.

Así mismo, el sistema dispone de un almacén donde se colocan los pedidos que ya han pasado todos los controles. La implementación del recurso compartido que controla el almacén os lo proporcionamos nosotos y os permitirá comprobar si todos los pedidos han llegado correctamente al almacén. Dicho recurso dispone de dos operaciones, (1) notificar que un pedido ya se encuentra preparado para almacenar, y (2) colocar el pedido en el almacén.

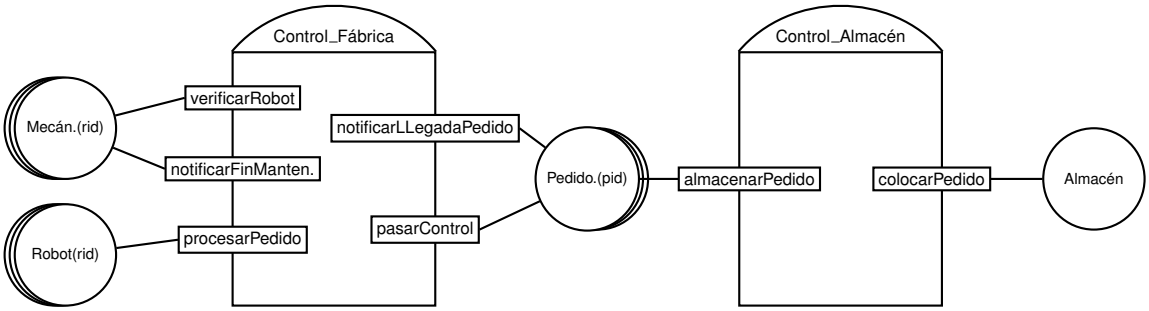
- Cuando un pedido ya ha pasado por todos los robots, será almacenado en el almacén.
- El almacén dispone de una *cola* de pedidos para colocar en estricto orden de llegada los diferentes pedidos pendientes de almacenar.

El siguiente gráfico representa las ideas presentadas:



Control_Fábrica *Control_Almacén*

Para poder probar el funcionamiento del recurso *Control_Fábrica*, que tendréis que implementar, se ha realizado una arquitectura que queda reflejada en siguiente grafo de procesos y recursos:



Las interacciones con el recurso compartido se muestran en el siguiente códido (simplificado), que es ejecutado por cada uno de los procesos que participan en el sistema:

```
// Clase Pedido
public void run() {
  for (i=0; i < N_ROBOTS; i++) {
    ...
    cFabr.notificarLlegadaPedido(pid, i);
    ...
    cFabr.pasarControl(pid);
    ...
  }
  cAlm.almacenarPedido(pid);
}
```

```
// Clase Robot

public void run() {
  while (nProcesos < N_PEDIDOS) {
    ...
    int pid= cFabr.procesarPedido(rid);
    ...
    nProcesos ++;
  }
}
```

```
// Clase Mecanico
public void run() {
  while (...) {
    ...
    nProcs=cFabr.verificarRobot(rid);
    ....
    cFabr.notificarFinMantenimiento(rid);
    ...
  }
}
```

```
  //Clase Almacen
  public void run() {
    nAlmac = 0;
    while (nAlmac < N_PEDIDOS) {
      ...
      int pid = cAlm.colocarPedido();
      ...
      nAlmacenados ++;
    }
  }
```

La Figura 1 detalla la especificación del recurso compartido *Control_Fábrica*.

2. Prácticas

2.1. Primera práctica

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

2.2. Segunda práctica

La entrega consistirá en una implementación del recurso compartido *Control_Fabrica* en Java mediante paso de mensajes síncrono, usando la librería JCSP. La implementación deberá estar contenida en un fichero llamado `ControlFabricaCSP.java` que, al igual que en la práctica 1, implementará la interfaz `ControlFabricaInterface`.

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 la solución con monitores 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.ls.fi.upm.es/teaching/concurrencia` varias unidades de compilación:

- `ControlFabricaMonitor.java`, que contiene las cabeceras de los métodos a implementar y la cabecera del constructor de dicha clase. Deberéis implementar tanto los métodos como el constructor.
- `Fabrica.java`: programa principal que crea los recursos compartidos y lanza los procesos que modelan el comportamiento de los pedidos, robots, mecánicos y almacén para cada una de las dos prácticas.

C-TAD Control_Fábrica

OPERACIONES
ACCIÓN notificarLlegadaPedido: *PID[e]* × *RID[e]*
ACCIÓN pasarControl: *PID[e]*
ACCIÓN procesarPedido: *RID[e]* × *PID[s]*
ACCIÓN verificarRobot: *RID[e]* × ℕ*[s]*
ACCIÓN notificarFinMantenimiento: *RID[e]*

SEMÁNTICA
DOMINIO:
TIPO: *Control_Fábrica= (pedidosEnEspera : PID → boolean* ×
colaRobot : RID → Secuencia(PID) ×
numProcesados : RID → ℕ ×
enMantenimiento : RID → boolean)
PID= 0 .. N_PEDIDOS - 1
RID= 0 .. N_ROBOTS - 1 N_PROCS = . . .

INICIAL: $\forall\, pid \in PID \bullet \text{self}.pedidosEnEspera(pid) = falso \wedge$
$\forall\, rid \in RID \bullet \text{self}.colaRobot(rid) = \langle\rangle \wedge$
$\forall\, rid \in RID \bullet \text{self}.numProcesados(rid) = 0 \wedge$
$\forall\, rid \in RID \bullet \text{self}.enMantenimiento(rid) = falso$

INVARIANTE: $\sum_{rid=0}^{N\_ROBOTS} Longitud(\text{self}.colaRobot(rid)) \leq N\_PEDIDOS \wedge$

$\sum_{rid=0}^{N\_ROBOTS} \text{self}.numProcesados(rid) \leq N\_PEDIDOS * N\_ROBOTS$

CPRE: *cierto*
notificarLlegadaPedido(pid,rid)
POST: $\text{self}^{pre} = (pe, cr, np, em) \wedge l = Longitud(cr(rid)) \wedge$
$\text{self} = (pe \oplus \{pid \rightarrow cierto\}, cr \oplus \{rid \rightarrow cr(rid)(0..l-1) + \langle pid \rangle\}, np, em)$

CPRE: $\neg\text{self}.pedidosEnEspera(pid)$
pasarControl(pid)
POST: $\text{self} = \text{self}^{pre}$

CPRE: $Longitud(\text{self}.colaRobot(rid)) > 0 \wedge \neg\text{self}.enMantenimiento(rid)$
procesarPedido(rid,spid)
POST: $\text{self}^{pre} = (pe, cr, np, em) \wedge l = Longitud(cr) \wedge spid = cr(rid)(0) \wedge$
$\text{self} = (pe \oplus \{pid \rightarrow falso\},$
$cr \oplus \{rid \rightarrow cr(rid)(1..l-1),$
$np \oplus \{rid \rightarrow np(rid) + 1\},$
$em \oplus \{rid \rightarrow (np(rid) + 1 \, mod N\_PROCS(rid)) = 0 \Rightarrow cierto)\})$

CPRE: $\text{self}.enMantenimiento(rid)$
verificarRobot(rid, snumprocs)
POST: $snumprocs = \text{self}^{pre}.numProcesados(rid) \wedge \text{self} = \text{self}^{pre}$

CPRE: cierto
notificarFinMantenimiento(rid)
POST: $\text{self} = \text{self}^{pre} \setminus \text{self}.enMantenimiento(rid) = falso$

Figura 1: Especificación del recurso *Control_Fábrica*

- `ControlFabricaInterface.java`: define la interfaz común a las distintas implementaciones del recurso compartido *Control_Fábrica*.
- `ControlAlmacenInterface.java` y `ControlAlmacenMonitor.java`: definen la interfaz y la implementación del recurso *Control_Almacen*. No es necesario implementar nada en este recurso.
- Código de los threads que modelan los pedidos, robots, mecánicos y almacén: `Pedido.java`, `Robot.java`, `Mecanico.java` y `Almacen.java`.
- La clase `TipoFabrica` permite arrancar diferentes tipos de fábrica estableciendo diferentes valores para *N_PEDIDOS*, *N_ROBOTS* y *N_PROCS*. Esto se realiza en la `Fabrica` mediante la llamada `TipoFabrica.setTipoFabrica(TIPO_FABRICA.TIPO_X)`. Exiten 10 configuraciones predefinidas para que podáis realizar pruebas.
- El método estático `TipoFabrica.N_PROCS(int rid)` se utilizará en la implementación del recurso `ControlFabricaMonitor` para acceder a los diferentes valores de *N_PROCS*.
- Tipo fábrica también permite configurar el "ritmo" de trabajo de la fábrica, es decir, lento, normal, rápido... Exiten 5 velocidades diferentes para la fábrica. Esto se realiza en la clase `Fabrica` mediante la llamada la método `TipoFabrica.setVelocidadFabrica(TIPO_VELOCIDAD.MUY_RAPIDO)`.

Por supuesto, durante el desarrollo podéis cambiar el código de estas clases 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 aparte de las estándar de Java y las mencionadas 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.