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 2022–2023

Calendario de entregas y revisiones

19 de mayo 15:00:00	Fecha límite de entrega **opcional** práctica 1
26 de mayo	Informe de resultados de entrega **opcional** práctica 1
26 de mayo 23:59:59	Fecha límite de entrega **opcional** práctica 2
2 de junio	Informe de resultados de entrega **opcional** práctica 2
5 de junio	Tras estas entregas opcionales revisaremos los sistemas de entrega y publicaremos nuevas pruebas en el sistema de entrega para ambas prácticas
9 de junio 23:59:59	Fecha límite de entregas **obligatorias** práctica 1 y práctica 2

Normas

- Los informes con los resultados incluirán dos tipos de comentarios: **comentarios generales** sobre problemas detectados en muchas prácticas y **comentarios específicos** de tu práctica. Deberás **tener en cuenta todos los comentarios**, los generales y los específicos. A veces tu práctica no recibirá comentarios específicos pero puede estar afectada por los comentarios generales (que normalmente tendrán que ver con orientaciones metodológicas a la hora de programar los recursos usando monitores o JCSP).
- Es **obligatorio reentregar** práctica 1 y práctica 2 en el periodo del 5 de junio al 9 de junio aunque tus entregas opcionales no hubieran recibido un informe negativo.
- Os recordamos que **todas** las prácticas entregadas pasan por un **proceso de detección de copias**.

1. Almacén

Un almacén de la conocida multinacional Orinoco ofrece ventas online a sus clientes. El almacén contiene varios tipos de productos vendidos a precio fijo. Un cliente puede comprar múltiples unidades de un tipo de producto a la vez. En esta práctica haremos la simplificación de asumir que un cliente siempre tiene dinero suficiente para comprar un producto.

El almacén permite la venta de un producto cuando hay suficientes unidades guardadas en el almacén, o cuando se sabe que más unidades están en camino desde las fábricas (que producen los productos) al almacén.

Tras la venta de un producto, si hay un número suficiente de unidades guardadas en el almacén, se procede a la entrega de los productos al cliente. Si un cliente no está satisfecho con los productos entregados, el cliente puede devolverlos.

Vuestra tarea es implementar la lógica del almacén que maneja las compras, las entregas, y las peticiones de reabastecer producto desde las fábricas.

1.1. Diseño

Los clientes pueden invocar tres tipos de operaciones al controlador del almacén (`ControlAlmacen`):

- `comprar(idCliente,idProducto,cantidad)` — el cliente *idCliente* compra *cantidad* unidades del producto tipo *idProducto*,
- `entregar(idCliente,idProducto,cantidad)` — el cliente pide la entrega del producto comprado, y
- `devolver(idCliente,idProducto,cantidad)` — el cliente devuelve el producto comprado.

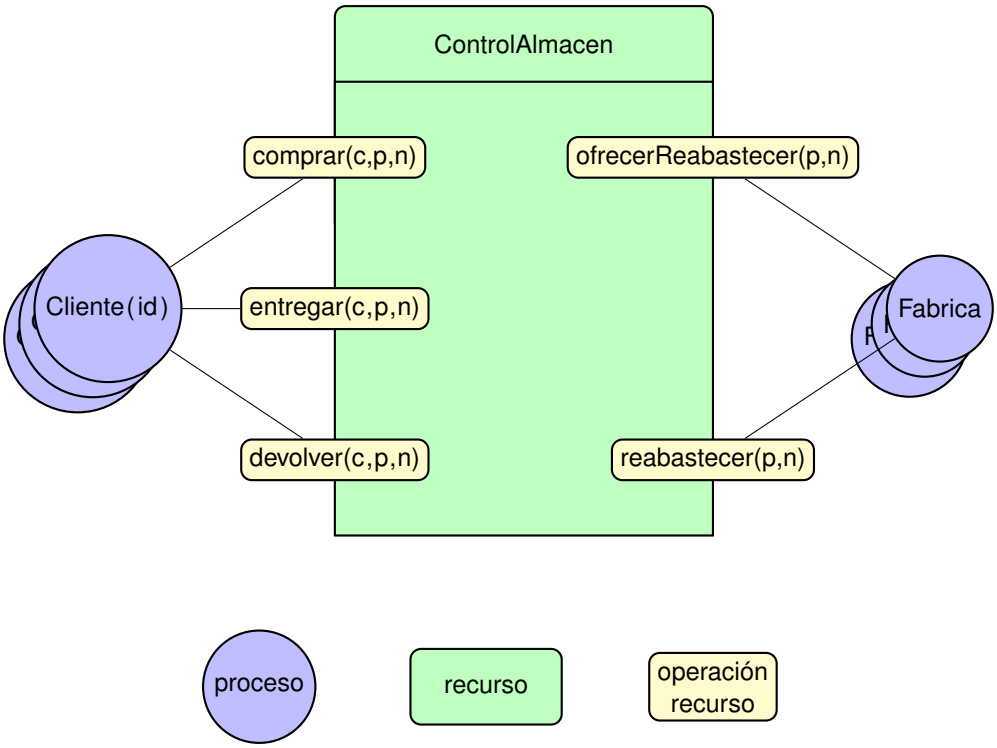

Figura 1: Grafo de procesos/recursos.

Asimismo, las fábricas ofrecen y entregan productos al almacén mediante las operaciones

- `ofrecerReabastecer(idProducto,cantidad)` – una fábrica ofrece *cantidad* ejemplares del producto tipo *idProducto* y
- `reabastecer(idProducto,cantidad)` — la fábrica entrega los ejemplares del producto en el almacén.

Notad que un mismo cliente puede realizar peticiones de distintos tipos de productos sin tener que esperar que se le entregue el primer producto para pedir el segundo, siempre y cuando los productos sean de tipos distintos. La arquitectura del sistema se muestra en la figura 1.

La idea es que el estado interno del recurso compartido sea suficientemente rico como para determinar cuándo se puede permitir que un cliente compre un producto (es decir, cuando hay suficientes unidades en el almacén o en camino desde la fábrica), y cuándo hay que pedir más productos a fábrica (cuando el número de unidades es inferior a un límite definido para este producto). Más concretamente, la gestión de cada tipo de producto en el almacén viene dada por 4 valores:

- *disponibles*: cuántas unidades de producto están presentes actualmente en el almacén,
- *enCamino*: cuántas unidades está previsto que lleguen de fábrica,
- *comprados*: cuántas unidades han sido compradas, pero aún no enviadas a los clientes,
- *minDisponibles*: cuántas unidades de este tipo de producto es deseable siempre tener disponibles (no vendidas) en el almacén. Esta es una constante que puede tener valores distintos para cada tipo de producto.

La especificación del gestor se encuentra en la figura 2 y la interfaz a respetar en su implementación es la siguiente:

```
package cc.controlAlmacen;

public interface ControlAlmacen {
  public boolean comprar(String clientId, String productoId, int cantidad);
  public void entregar(String clientId, String productoId, int cantidad);
```

```
  public void devolver(String clientId, String productoId, int cantidad);

  public void ofrecerReabastecer(String productoId, int cantidad);
  public void reabastecer(String productoId, int cantidad);
}
```

Como podéis ver, los tipos c_{id} y p_{id} de la especificación formal se han representado como `String`.

Prioridad. Cuando hay dos o más entregas posibles *del mismo producto*, es decir hay al menos dos llamadas *entregar(c1,p,n1)* y *entregar(c2,p,n)* esperando, es **obligatorio** dar prioridad a la llamada que lleva más tiempo esperando. Incluso si no hay suficientes unidades de un producto para cumplir una petición de entrega e_1, pero sí para una posterior entrega e_2 (para el mismo tipo de producto p), la entrega de e_2 tiene que esperar por la entrega de e_1.

2. Prácticas

2.1. Primera práctica

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

2.2. Segunda práctica

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

2.3. Constructores

Ambos constructores ControlAlmacenMonitor y ControlAlmacenJCSP reciben como parámetro un map Map<**String**,Integer> productoId que asocia a cada tipo de producto conocido (el **String**) su correspondiente cantidad mínima de unidades en almacén (el entero) — es decir, su *minDisponible*.

2.4. Precondiciones

Si no se cumple una precondición vuestras implementaciones de ControlAlmacen deben lanzar una excepción IllegalArgumentException.

C-TAD ControlAlmacen
ACCIÓN Comprar: c_{id}[e] × p_{id}[e] × ℕ[e] × 𝔹[s]
ACCIÓN Entregar: c_{id}[e] × p_{id}[e] × ℕ[e]
ACCIÓN Devolver: c_{id}[e] × p_{id}[e] × ℕ[e]
ACCIÓN OfrecerReabastecer: p_{id}[e] × ℕ[e]
ACCIÓN Reabastecer: p_{id}[e] × ℕ[e]

SEMÁNTICA
DOMINIO
TIPO ControlAlmacen = p_{id} → (disponibles: ℕ × enCamino: ℕ × comprados: ℕ × minDisponibles: ℕ)

INICIAL self = { p_1 ↦ (0, 0, 0, min_p_1), ..., p_n ↦ (0, 0, 0, min_p_n) }

PRE: n > 0 ∧ p ∈ **dom self**
CPRE: **cierto**
Comprar(c,p,n)
POST: **self**^{pre}(p) = (d,e,c,m) ∧
result = d + e ≥ n + c ∧
result ⇒ **self** = **self**^{pre} ⊕ {p ↦ (d,e,c+n,m)} ∧
¬**result** ⇒ **self** = **self**^{pre}

PRE: n > 0 ∧ p ∈ **dom self**
CPRE: **self**(p).disponibles ≥ n
Entregar(c,p,n)
POST: **self**^{pre}(p) = (d,e,c,m) ∧
self = **self**^{pre} ⊕ {p ↦ (d−n,e,c−n,m)}

PRE: n > 0 ∧ p ∈ **dom self**
CPRE: **cierto**
Devolver(c,p,n)
POST: **self**^{pre}(p) = (d,e,c,m) ∧
self = **self**^{pre} ⊕ {p ↦ (d+n,e,c,m)}

PRE: n > 0 ∧ p ∈ **dom self**
CPRE: **self**(p).disponibles + **self**(p).enCamino − **self**(p).comprados < **self**(p).minDisponibles
OfrecerReabastecer(p,n)
POST: **self**^{pre}(p) = (d,e,c,m) ∧
self = **self**^{pre} ⊕ {p ↦ (d,e+n,c,m)}

PRE: n > 0 ∧ p ∈ domain(**self**)
CPRE: **cierto**
Reabastecer(p,n)
POST: **self**^{pre}(p) = (d,e,c,m) ∧
self = **self**^{pre} ⊕ {p ↦ (d+n,e−n,c,m)}

Figura 2: Especificación formal del recurso compartido.

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:

- `ControlAlmacen.java`: interfaz común a las implementaciones del recurso compartido.
- `ControlAlmacenMonitor.java`: esqueleto para la práctica 1.
- `ControlAlmacenCSP.java`: esqueleto para la práctica 2, que incluirá la mayor parte del código *vernacular* de JCSP (se publicará más tarde).
- `Plata.java`: programa simulador que crea todas las intancias necesarias de las APIs, recursos compartidos y pone en marcha los procesos.[1]

  Nota: el simulador no hace ninguna comprobación de que la implementación del recurso compartido sea correcta, aunque podrás detectar errores que sean muy evidentes. Te **recomendamos** que implementes tú mismo algunas pruebas *secuenciales* sencillas.
- `Cliente.java`, `Fabrica.java`: Ejemplos de código de procesos usados por el simulador `Plata.java`.
- `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 estándar de Java y las bibliotecas auxiliares que están en el código de apoyo (incluida `aedlib-2.10.3.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.

[1] Explora el código fuente para entender el diseño de forma concreta así como para adaptar la solución a monitores o paso de mensajes (busca la instanciación del recurso `ControlAlmacenMonitor` o `ControlAlmacenCSP`).