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 2021–2022

Calendario de entregas y revisiones

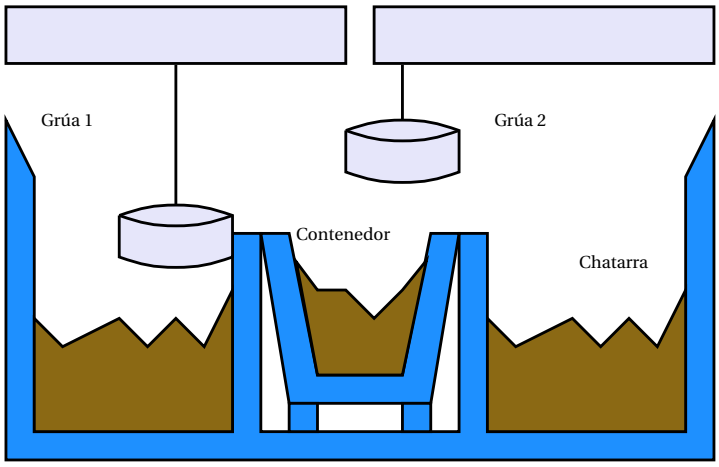
20 de mayo 23:59:59	Fecha límite de entrega **opcional** práctica 1
26 de mayo	Informe de resultados de entrega **opcional** práctica 1
27 de mayo 23:59:59	Fecha límite de entrega **opcional** práctica 2
3 de junio	Informe de resultados de entrega **opcional** práctica 2
4 de junio	Tras estas entregas opcionales revisaremos los sistemas de entrega y publicaremos nuevas pruebas en el sistema de entrega para ambas prácticas
10 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 4 de junio al 10 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. Planta de reciclado

En una planta de reciclado se recuperan ciertos metales de entre la chatarra mediante una serie de grúas equipadas con fuertes electroimanes. Estas grúas se encargan luego de depositar los metales en un contenedor hasta que está más o menos lleno. La siguiente figura muestra un esquema de la planta con dos gruas:



Las gruas son accionadas a través de una clase cuya interfaz mostramos a continuación:

```
class ApiGruas {
  public final int MAX_GRUAS;
  public final int MIN_P_GRUA;
  public final int MAX_P_GRUA;

  public Gruas(int max_gruas, int min_p_grua, int max_p_grua) {
    MAX_GRUAS = max_gruas;
    MIN_P_GRUA = min_p_grua;
    MAX_P_GRUA = max_p_grua;
  }

  /**
   * Provoca que la grua idGrua recoja metal y cuando finalice informa
   * de su peso.
   */
  public int recoger (int idGrua) {
    ...
  }

  /**
   * Mueve la grua idGrua hasta el punto de descarga y desactiva el
   * electroiman.
   */
  public void soltar (int idGrua) {
    ...
  }
}
```

De igual manera, se dispone de una clase para controlar el desplazamiento del contenedor:

```
class ApiContenedor {
  public final int MAX_P_CONTENEDOR;

  public Contenedor(int max_p_contenedor) {
    MAX_P_CONTENEDOR = max_p_contenedor;
  }

  /**
   * Sustituye el contenedor actual con otro vacio.
   */
  public void sustituir () {
    ...
  }
}
```

Queremos implementar un sistema concurrente para controlar las grúas y el contenedor de manera que no se exceda la capacidad de los contenedores, y sustituir contenedores llenos por otros vacíos, asegurándonos de que nunca se intenta depositar metal en la zona del contenedor mientras éste está siendo reemplazado. Para simplificar el problema supondremos que hay espacio suficiente para que unas grúas no se estorben físicamente con otras.

1.1. Diseño

Tendremos un proceso para controlar cada grúa 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 1.

La idea es que el estado interno del recurso compartido sea suficientemente rico como para determinar cuándo hay que solicitar la sustitución del contenedor y sincronizar a los procesos que controlan las grúas con esa circunstancia.

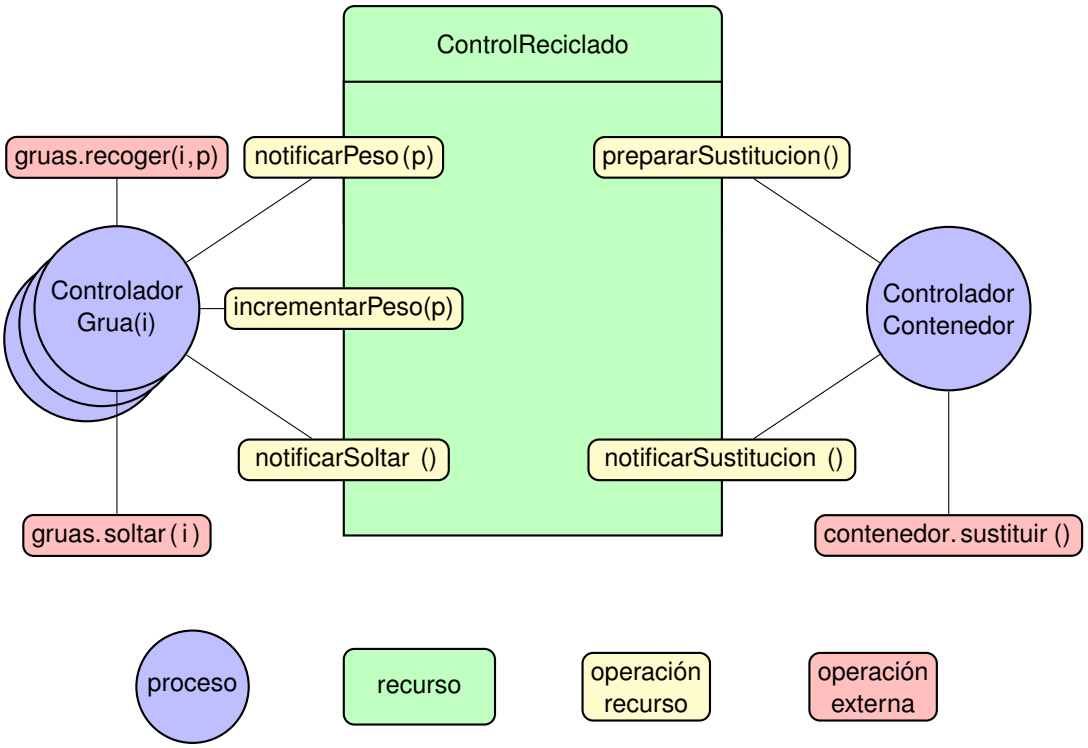


Figura 1: Grafo de procesos/recursos.

Más concretamente, se dispondrá de un *estado de reposición* que puede tomar uno de tres valores: *listo*, que quiere decir que el contenedor admite más carga; *sustituible*, que indica que al menos una de las grúas lleva más carga de la que cabe en el contenedor, por lo que dicha grúa solicita un contenedor nuevo, y *sustituyendo*, estado en el cual no se debe depositar carga ya que el contenedor puede estar siendo reemplazado. Obsérvese que incluso en el estado *sustituible* es posible que una grúa distinta a la que ha solicitado la sustitución del contenedor deposite una cantidad menor en el contenedor.

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

```
package cc.controlReciclado;

public interface ControlReciclado {
  public void notificarPeso(int p) throws IllegalArgumentException;
  public void incrementarPeso(int p) throws IllegalArgumentException;
  public void notificarSoltar();
  public void prepararSustitucion();
  public void notificarSustitucion();
}
```

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 `ControlRecicladoMonitor.java` que implementará la interfaz ControlReciclado.

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 `ControlRecicladoCSP.java` que implementará la interfaz ControlReciclado.

C-TAD ControlReciclado
ACCIÓN NotificiarPeso: $\mathbb{N}$[e]
ACCIÓN IncrementarPeso: $\mathbb{N}$[e]
ACCIÓN NotificarSoltar:
ACCIÓN PrepararSustitución:
ACCIÓN NotificarSustitución:

SEMÁNTICA
DOMINIO
TIPO ControlReciclado = (peso: $\mathbb{N}$ × estado: Estado × accediendo: $\mathbb{N}$)
Estado = listo | sustituible | sustituyendo

INICIAL self = (0, listo , 0)
INVARIANTE self.peso ≤ MAX_P_CONTENEDOR

PRE: p > 0 ∧ p ≤ MAX_P_GRUA
CPRE: **self**.estado ≠ sustituyendo
NotificarPeso(p)
POST: **self**.peso = **self**^{pre}.peso ∧ **self**.accediendo = **self**^{pre}.accediendo
∧ **self**^{pre}.peso + p > MAX_P_CONTENEDOR ⇒ **self**.estado = sustituible
∧ **self**^{pre}.peso + p ≤ MAX_P_CONTENEDOR ⇒ **self**.estado = listo

PRE: p > 0 ∧ p ≤ MAX_P_GRUA
CPRE: **self**.peso + p ≤ MAX_P_CONTENEDOR ∧ **self**.estado ≠ sustituyendo
IncrementarPeso(p)
POST: **self**^{pre} = (peso, e, a) ∧ **self** = (peso + p, e, a + 1)

CPRE: **cierto**
NotificarSoltar ()
POST: **self**^{pre} = (p, e, a) ∧ **self** = (p, e, a − 1)

CPRE: **self** = (_, sustituible , 0)
PrepararSustitución()
POST: **self** = (_, sustituyendo, 0)

CPRE: **cierto**
NotificarSustituci ón()
POST: **self** = (0, listo , 0)

Figura 2: Especificación formal del recurso compartido.

3. Información general

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

`http://vps142.cesvima.upm.es`

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

- `ControlReciclado.java`: interfaz común a las implementaciones del recurso compartido.
- `ControlRecicladoMonitor.java`: esqueleto para la práctica 1.
- `ControlRecicladoCSP.java`: esqueleto para la práctica 2, que incluirá la mayor parte del código *vernacular* de JCSP (se publicará más tarde).
- `ApiGruas.java`: API de las gruas.
- `ApiContenedor.java`: API del contenedor.
- `ControladorGrua.java`: procesos que controlan las gruas interaccionando con el recurso compartido.
- `ControladorContenedor.java`: proceso que controla el contenedor interaccionando con el recurso compartido.
- `PlantaReciclaje.java`: programa concurrente que crea todas las intancias necesarias de los APIs, recursos compartidos y pone en marcha los procesos.[1]

  Nota: el simulador no hace ningún 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.
- `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-2.9.0.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 `ControlRecicladoMonitor` o `ControlRecicladoCSP`).