Concurrencia
Prácticas 1 y 2

Grado en Ingeniería Informática UPM Convocatoria de Segundo semestre 2012-2013

Normas

- La fecha límite de entrega de la práctica 1 es el viernes **24 de mayo de 2013** a las 23:59:59.
- La fecha límite de entrega de la práctica 2 es el viernes **31 de mayo de 2013** a las 23:59:59.
- A partir del **25 de mayo de 2013** publicaremos resultados de someter a pruebas las entregas de la práctica 1, y a partir del **1 de junio de 2013** los de la práctica 2. Las prácticas con problemas podrán ser reentregadas hasta el **7 de junio de 2013** 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. Observer

El patrón de diseño *Observer* (capítulo 5 del libro *Design Patterns* de Gamma, Helm, Johnson y Vlissides) es uno de los esquemas de diseño más aplicados en programación (sistemas reactivos, interfaces de usuario, etc.). Su funcionamiento se basa en un conjunto de eventos *emitidos* por unos *sujetos*, a los que se subscriben los *observadores*. Cuando uno de esos eventos ocurre, todos los observadores subscritos son *notificados* para que actúen en consecuencia.

Hemos adaptado el patrón a esta asignatura de forma que tendremos procesos que emiten eventos y procesos que se subscriben a la a determinados eventos para poder escuchar hasta que alguno sea emitido y reaccionar.

1.1. Diseño

El sistema de gestión de subscripciones y eventos se ha diseñado como un único recurso compartido con operaciones de subscripción, desubscripción y emisión de eventos. Para poder probar el funcionamiento de dicho recurso, recurso que tendréis que implementar, se ha realizado una arquitectura que queda reflejada en el grafo de procesos y recursos en la figura 1.

Los procesos emisores están parametrizados por el identificador de evento que van a emitir (*eid*). Los procesos observadores están paremetrizados por un identificador de procesos diferente (*pid*). La figura 2 muestra el código de los procesos.

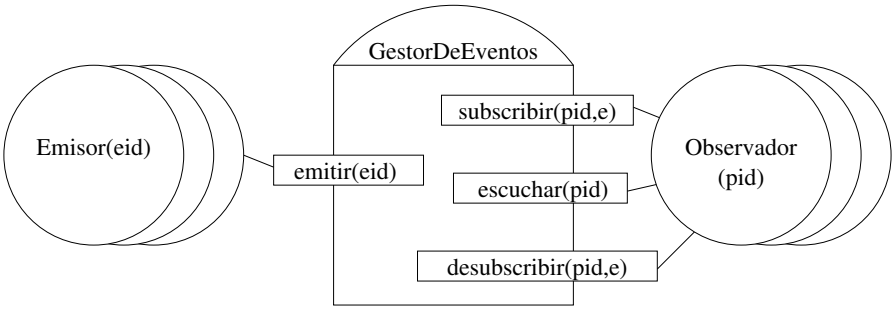

Figura 1: Grafo de procesos/recursos.

```
public class Emisor extends Thread {
  private GestorDeEventos ge;
  private int eid;

  public Emisor(GestorDeEventos ge,
                int eid) {
    this.ge = ge;
    this.eid = eid;
  }

  public void run() {
    // Continuamente
    // emite el evento eid
    while (true) {
      ge.emitir(eid);
    }
  }
}
```

```
public class Observador extends Thread
{
  private GestorDeEventos ge;
  private int pid;

  public Observador(GestorDeEventos ge,
                    int pid) {
    this.ge = ge;
    this.pid = pid;
  }

  public void run() {
    while (true) {
      int n = 2, i, eid = 0;
      // Se subscribe a n eventos
      for (i = 0; i < n; i++) {
        eid = (pid + i) % N_EVENTOS;
        ge.subscribir(pid, eid);
      }
      // Escucha la llegada de algún
      // evento y se desubscribe una
      // vez escuchado hasta quedarse
      // sin subscripciones
      for (i = 0; i < n; i++) {
        eid = ge.escuchar(pid);
        ge.desubscribir(pid,eid);
      }
    }
  }
}
```

Figura 2: Código de los procesos.

Terminamos el diseño con la especificación formal del recurso que gestiona subscripciones, emisiones y escuchas de eventos:

C-TAD Gestor_De_Eventos
OPERACIONES
ACCIÓN Emitir: *EID[e]*
ACCIÓN Subscribir: *PID[e]* × *EID[e]*
ACCIÓN Desubscribir: *PID[e]* × *EID[e]*
ACCIÓN Escuchar: *PID[e]* × *EID[s]*

SEMÁNTICA
DOMINIO:
TIPO: *Gestor_De_Eventos*= *(subscritos : EID* → Conjunto*(PID)* × *porescuchar : EID* → Conjunto*(PID))*
EID= 0 .. N_EVENTOS
PID= 0 .. N_OBSERVADORES

INICIAL: $\forall\, eid \in EID \bullet \text{self}.subscritos(eid) = \emptyset \wedge \text{self}.porescuchar(eid) = \emptyset$

CPRE: Cierto
Emitir(eid)
POST: $\text{self}.porescuchar(eid) = \text{self}^{pre}.subscritos(eid) \wedge$
$\forall\, e \in EID \bullet \ (\text{self}.subscritos(e) = \text{self}^{pre}.subscritos(e) \wedge$
$(e \neq eid \Rightarrow \text{self}.porescuchar(e) = \text{self}^{pre}.porescuchar(e)))$

CPRE: Cierto
Subscribir(pid,eid)
POST: $\text{self}.subscritos(eid) = \text{self}^{pre}.subscritos(eid) \cup \{pid\} \wedge$
$\forall\, e \in EID \bullet \ (\text{self}.porescuchar(e) = \text{self}^{pre}.porescuchar(e) \wedge$
$(e \neq eid \Rightarrow \text{self}.subscritos(e) = \text{self}^{pre}.subscritos(e)))$

PRE: $pid \in \text{self}.subcritos(eid)$
CPRE: Cierto
Desubscribir(pid,eid)
POST: $\text{self}.subscritos(eid) = \text{self}^{pre}.subscritos(eid) \setminus \{pid\} \wedge$
$\text{self}.porescuchar(eid) = \text{self}^{pre}.porescuchar(eid) \setminus \{pid\} \wedge$
$\forall\, e \in EID \bullet e \neq eid \Rightarrow \ (\text{self}.porescuchar(e) = \text{self}^{pre}.porescuchar(e) \wedge$
$\text{self}.subscritos(e) = \text{self}^{pre}.subscritos(e))$

CPRE: $\exists\, e \in EID \bullet pid \in \text{self}.porescuchar(e)$
Escuchar(pid,eid)
POST: $pid \in \text{self}^{pre}.porescuchar(eid) \wedge$
$\text{self}.porescuchar(eid) = \text{self}^{pre}.porescuchar(eid) \setminus \{pid\} \wedge$
$\forall\, e \in EID \bullet \ (\text{self}.subscritos(e) = \text{self}^{pre}.subscritos(e) \wedge$
$(e \neq eid \Rightarrow \text{self}.porescuchar(e) = \text{self}^{pre}.porescuchar(e)))$

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 `GestorDeEventosMonitor.java` que implementará la interfaz `GestorDeEventos`. (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 `GestorDeEventosCSP.java` que implementará la interfaz `GestorDeEventos`. (ver sec. 3).

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 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.ls.fi.upm.es/teaching/concurrencia` varias unidades de compilación:

- `PatronObserverMonitor.java` y `PatronObserverCSP.java`: programas principales que crean el recurso compartido y lanzan los procesos emisores y observadores para cada una de las dos opciones.
- `GestorDeEventos.java`: define la interfaz común a las distintas implementaciones del recurso compartido.
- Código de los procesos emisores y observadores: `Emisor.java` y `Observador.java`.

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.