UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación de Sistemas y Concurrencia
Control 25/5/2012

APELLIDOS _____ NOMBRE _____

DNI _____ ORDENADOR _____ GRUPO _____

1.- SEMÁFOROS.

a) Sean **n abejas** de miel y **un oso** goloso que comparte un tarro de miel. El tarro está inicialmente vacío: tiene capacidad para **H** porciones de miel. El oso sólo come del tarro cuando está lleno y, en ese caso, se traga de una sentada todas las porciones de una vez. El resto del tiempo el oso está descansando. Cada abeja repetidamente recoge una porción de miel y la pone en el tarro; si una abeja llega al tarro y pone la última ración, despierta al oso para que se tome toda la miel del tarro. Implementa este sistema utilizando **semáforos binarios**, suponiendo que hay una **cantidad indefinida de miel** que las abejas pueden recoger, y que el oso está descansando mientras no puede comer. Para ello debes implementar:

- Una clase `Tarro` que proporciona los métodos `void nuevaPorcion(int)`, `void comoMiel()`, usados, respectivamente, por las abejas y el oso para dejar porciones de miel, y para beberse la miel del tarro
- Dos clases `Oso`, y `Abeja` que implementan el comportamiento del oso y las abejas.
- Una case `Principal` que pone en marcha todo el sistema.

b) Implementa el sistema **para que termine**. Supón que cada abeja deposita un número finito de porciones de miel en el tarro. Para simplificar el problema, supón que la capacidad del tarro H es múltiplo del número total de porciones depositadas por las abejas.

2.- MONITORES

a) Supón que **un proceso productor** y **C** procesos consumidores **cons[1..C]** se comunican a través de una variable simple **ms** (simulando un buffer con una única componente). El productor deja todos los datos que produce en **ms**, y los consumidores los recogen de **ms**. Cada dato depositado por el productor debe ser consumido por los **C** consumidores antes de que el productor pueda dejar otro dato en ms. Recíprocamente, un consumidor que ha leído un dato no puede volver a leer hasta que el productor no hay dejado en ms un nuevo dato. Resuelve este problema utilizando métodos sincronizados o locks, suponiendo que el productor produce datos de forma indefinida. Para ello debes implementar:

- Una clase `Mensaje`, que contiene la variable ms, y proporciona los métodos `void nuevoMs()`, `int leoMs(int)`, utilizados, respectivamente,

por el productor y los consumidores para dejar un nuevo mensaje y leerlo de ms. Para simplificar suponemos que los mensajes son de tipo entero.

- Las clases `Productor` y `Consumidor` que implementan el comportamiento de estos procesos.
- Una clase `principal` que pone en marcha el sistema.
-

b) Implementa el sistema para que termine. Para ello, supón que el productor produce un número finito de datos. Si lo ves necesario, añade un método a la clase `Mensaje`, `boolean hayMensaje(int id)`, que devuelve a **true**, si el consumidor id todavía tiene que leer el mensaje almacenado, y **false**, en otro caso.

Notas para hacer el control:

1.- Crea un proyecto **`prControl`**
2.- Crea dos paquetes en **`prControl`** denominados **`miel`** y **`prodcons`**, y deja los fuentes de cada uno de los ejercicios en su paquete correspondiente.
3.- **Pon tu nombre en todos tus fuentes**.
4.- Cuando termines, haz un fichero comprimido (.zip, .rar, .jar) **con los fuentes** de tus ejercicios y súbelos a la tarea del campus virtual creada para este fin.
5.- En el control, no pueden utilizarse ni apuntes, ni ejercicios de clase. Sólo está permitido consultar la documentación de java.