UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación de Sistemas y Concurrencia

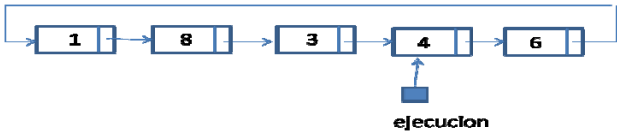
Examen final 21/6/2012

APELLIDOS_____NOMBRE_____

DNI _____ ORDENADOR _____GRUPO _____

Ejercicio 1. Lenguaje C (2 puntos)

Se desea implementar una lista enlazada circular para representar la lista de procesos que están disponibles para ejecución. El puntero externo apuntará al proceso que le corresponde la ejecución (ver figura).



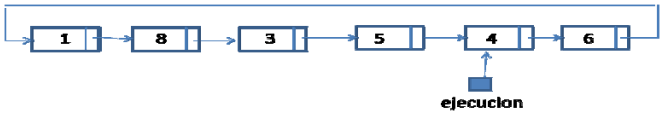
Definir la estructura de datos e implementar las siguientes operaciones:

```
void Crear (LProc *lista)
```

Crea una lista de procesos vacía.

```
void AñadirProceso (LProc *lista, int idproc)
```

Añade el proceso con identificador idproc a la lista de procesos disponibles para ejecución. Este proceso se añade como nodo anterior al nodo al que apunta ejecución. Si la lista está vacía el puntero externo apuntará a l único nodo. Dada la figura anterior, si queremos añadir el proceso 5, la lista quedaría como aparece en la figura:

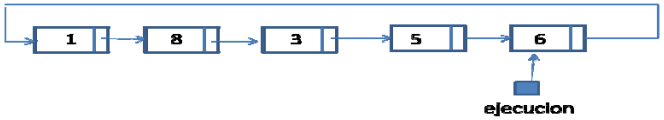



```
void MostrarLista( LProc lista)
```

Muestra la lista de los procesos que están disponibles para la ejecución.

```
void EjecutarProceso(LProc *lista)
```

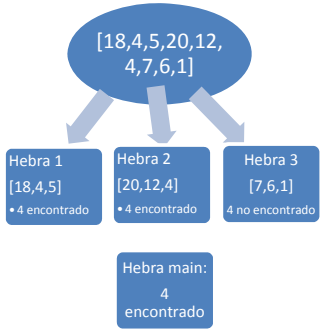
Simula la ejecución del proceso apuntado por ejecución, eliminándolo de la lista de procesos. Así, partiendo de la figura anterior, la lista quedaría :



Ejercicio 2. Hebras en Java (1 punto)

Desarrollar un programa en Java que permita hacer búsquedas de un elemento en un array de tipo int. Para realizar la búsqueda suponer que se dispone de un array de N posiciones y M hebras donde N y M podrán ser leídos desde teclado o almacenados como constantes. Los valores contenidos en el array son inicializados aleatoriamente antes de la creación de las hebras.

La solución desarrollada dividirá el array N en M fragmentos y realizará la búsqueda de un elemento X (elegido aleatoriamente) de manera que cada una de las hebras comprobará si el elemento se encuentra en el fragmento de array que le corresponda (se puede suponer para simplificar que N es múltiplo de M). Posteriormente, una vez hayan terminado todas las hebras de realizar su búsqueda local, en la hebra principal del programa se procederá a consultar a estas hebras si el elemento buscado estaba presente o no en su fragmento.



La figura de la derecha ilustra el proceso para un array de 9 posiciones y 3 hebras donde se está buscando el número 4.

Ejercicios 3 y 4. Semáforos y monitores (5 puntos)

Supón que hay tres personas que necesitan algún método para seleccionar una entre ellas. Una forma de hacerlo es que cada una tire una moneda. Si la tirada de alguna persona sale de forma diferente a la de las otras dos, esa persona gana. En otro caso, es decir, si las tres monedas salen igual (cara o cruz), hay empate y el juego debe repetirse de nuevo.

Así la secuencia de acciones que deben ocurrir en el sistema es:

Los tres jugadores tiran una moneda →los tres jugadores miran quién ha ganado

- → Si hay ganador, el juego termina
- → Si no hay ganador, se empieza de nuevo

3. Diseña un programa que simule el juego de la moneda utilizando **semáforos binarios**

4. Diseña un programa que simule el juego de la moneda utilizando **métodos sincronizados o locks**.

En ambos casos debes completar las 3 clases siguientes, que se dan parcialmente implementadas:

A. Clase `Mesa`: Es la clase del objeto mesa en el que los jugadores tiran sus monedas. una vez que los tres han tirado, comprueban si hay un ganador, y en ese caso, quien es. Para ello, la clase proporciona dos métodos:

- `void nuevaJugada(int id,int res)`, usado por el jugador `id` para dejar el resultado (`res`) de su tirada (0:cara, 1:cruz) en la mesa.

- `int ganador(int id)`, usado por el jugador `id` para ver si hay un ganador en la jugada actual. El método devuelve el identificador del ganador (0,1 o 2) si hay alguno, o -1, si hay empate.

B. Clase `Jugador`: es la clase de cada uno de los objetos personas que juegan. Cada jugador llama sucesivamente a los métodos `nuevaJugada` y `ganador` hasta que haya un ganador en una de las tiradas.

C. Clase `Principal`: la que pone en marcha todo el sistema.

Ejercicio 5. Paso de mensajes (1 punto)

Se quiere realizar el análisis de cadenas de caracteres en Java mediante paso de mensajes. El sistema analiza la cadena de caracteres y cuenta el número de apariciones de las diferentes vocales del alfabeto. Modelarlo de acuerdo con las siguientes indicaciones:

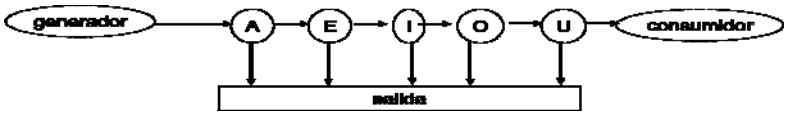
- Existe una hebra **generadora** que es la que va suministrando la cadena **carácter a carácter** a un pipeline de hebras conectadas de manera secuencial (como se muestra en la figura). La longitud de la cadena es aleatoria (valor generado en la hebra generadora), por lo que una vez transmitida por completo al pipeline de hebras, se transmitirá un carácter especial de terminación (ej: (char)0) para que las hebras sepan que no hay más caracteres por recibir.

  Los caracteres de la cadena son generados aleatoriamente utilizando la clase *Random* y convirtiendo el resultado al tipo *char*. El siguiente fragmento de código muestra cómo obtener una letra aleatoria en el rango 'A'..'Z'

```
Random r=new Random();
char letra;
letra=(char)(r.nextInt(25)+'A');
```

- En el pipeline de hebras existe una hebra para cada vocal encargada de contar las apariciones de dicha vocal en la cadena.
- Una vez terminada la transmisión de la cadena al pipeline de hebras, cada hebra de este pipeline transmitirá su resultado a una hebra salida que indicará cuántas veces aparece cada vocal en la cadena transmitida.
- Se puede tener una hebra adicional **consumidor** con el único objetivo de que las hebras del pipeline tengan la misma estructura de canales de entrada/salida.

  La siguiente figura muestra una representación de la red de hebras indicada.



Por ejemplo, para la cadena: 'HOLA JAVA' una traza de mensajes podría ser:

HOLA JAVA[1]
A: 3 veces; E: 0 veces ; I: 0 veces; O: 1 vez; U: 0 veces

[1] La cadena original puede mostrarse carácter a carácter conforme se va creando en la hebra generadora

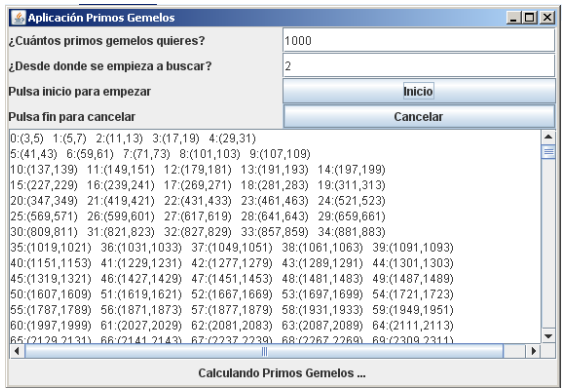
Ejercicio 6. Programación orientada a eventos (1 punto)

Se dice que dos números a y b (a < b) son *primos gemelos*, si a y b son primos y b = a + 2. Por ejemplo, los siguientes pares de números son primos gemelos (3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73), … No se sabe si hay un número infinito de pares de primos gemelos, aunque se conjetura que es así. En cualquier caso, se han encontrado pares de este tipo muy, muy grandes.

El objetivo de esta práctica es construir una GUI calcular los n pares (a,b) de primos gemelos que se encuentran después de un número dado m. Para ello debes utilizar el paquete pPrimosGemelos en el que las clases que debes usar se encuentran parcialmente implementadas. El paquete contiene las siguiente clases:

A. Clase `PrimosGemelos` que puede almacenar una pareja de primos gemelos, junto con la posición en la lista que se está construyendo. Esta clase está implementada.

B. Clase `Panel`, que construye una GUI con un aspecto similar al que se muestra en el diagrama. Esta clase está implementada.



C. Clase `Worker`: Esta clase está parcialmente implementada. Debe calcular los `n` primeros pares de primos gemelos a partir de un número `m`. Los valores `n`, y `m`, que deben pasarse como parámetros al constructor de la clase, son los números leídos desde los `JTextFields numPrimos`, `numInicio` del GUI. Un objeto worker debe ir publicando en el área de texto de la GUI los pares de primos amigos a medida que los va encontrando.

D. Clase `Controlador:` Esta clase está parcialmente implementada. Debe gestionar los eventos de los botones inicio y fin de la GUI. Cuando se pulsa inicio se debe empezar a buscar los primos gemelos con los parámetros leídos de la GUI. Si se pulsa fin, se debe cancelar la tarea e informar del hecho en el mensaje inferior de la GUI. Para simplificar esta aplicación, suponemos que todos los datos que se leen de la GUI son correctos, por lo que no hay que gestionar las posibles excepciones.

E. Clase `Principal:` Esta clase está parcialmente implementada. Debe construir una hebra dispatcher desde la que se cree la GUI.