UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación de Sistemas y Concurrencia

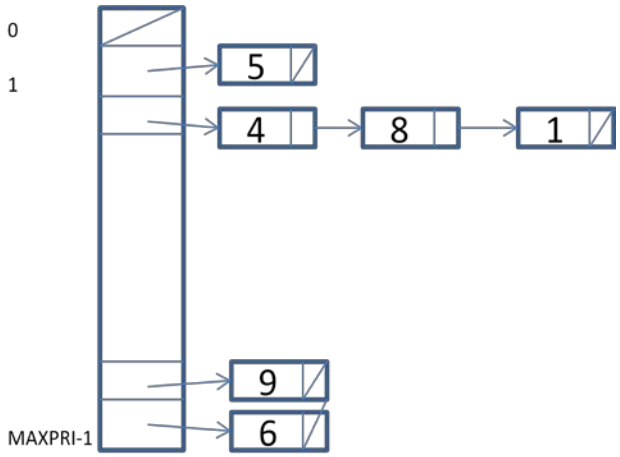
Examen final 4/9/2012

APELLIDOS_____ NOMBRE_____

DNI _____ ORDENADOR _____ GRUPO _____

Ejercicio 1. Lenguaje C (2 puntos)

Se desea implementar un sistema de colas de prioridad como la que aparece en la figura para planificar la ejecución de los procesos del sistema. Un array de MAXPRI representa las prioridades del sistema, donde 0 es la más prioritaria y MAXPRI-1 es la menor prioridad. Cada posición **i** del array contendrá una lista enlazada dinámica donde el primer elemento de ésta es el proceso de prioridad i que debe ejecutarse (siempre que no existan procesos disponibles en las **i-1** posiciones anteriores). Implementar las siguientes operaciones:



Crear. Inicializa el array.

AñadirProceso. Dada una prioridad y un identificador de proceso, lo añade al final de la lista que le corresponde.

EjecutarProceso. Elimina de la lista el proceso más prioritario que le corresponde ejecutarse. Si no existen procesos por ejecutar se indicará con un mensaje de aviso.

Buscar. Dado un identificador de proceso devuelve la prioridad de éste. Si el id del proceso no existe se devolverá -1.

Mostrar. Recorre la estructura para mostrar los procesos existentes que están disponibles para ejecución ordenados por prioridad.

Ejercicio 2. Hebras en Java (1 punto)

Desarrollar un programa en Java que permita hacer sumas de matrices de manera concurrente. Suponer que se dispone de dos arrays conteniendo los datos de las matrices a sumar, siendo estas matrices de tamaño NxN, donde N podrá ser leído desde teclado o almacenado como

constante. Los valores contenidos en el array son inicializados aleatoriamente antes de la creación de las hebras.

La solución desarrollada creará tantas hebras como filas tengan las matrices (es decir, N hebras). Cada una de las hebras recibirá como parámetro una fila completa de cada matriz de manera que en el método `run` de las hebras se realizará la suma de los elementos de la fila.

Desde la hebra principal del programa se esperará a que todas las hebras realicen sus respectivas sumas y se procederá a recoger el resultado y a mostrarlo por pantalla.

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

Supón que tienes una clase `Caja<T>` que utilizan un par de procesos para intercambiarse objetos de tipo T. La clase tiene un método *T intercambiar(int id,T obj)* que es llamado por los dos procesos para intercambiarse sus objetos. El primer proceso que llama tiene que esperar, y cuando el segundo llama al método, los objetos se intercambian y cada proceso continúa su ejecución.

Implementa este sistema, en primer lugar, para que los procesos se intercambien un único número entero. A continuación, supón que cada proceso tiene un array de enteros local con los números escogidos de forma aleatoria. Diseña el código de los procesos para que interaccionen repetidamente, a través del objeto de la clase `Caja`, hasta que uno de los procesos se queda con los números menores de los dos arrays, y el otro proceso con los mayores.

3.- Resuelve este problema con semáforos **binarios**

4.- Resuelve este problema con métodos sincronizados o locks.

Nota: Pueden serte útil los métodos estáticos de la clase Arrays (que está en `java.util`) siguientes:

- `Arrays.sort(int[])`
- `Arrays.toString(int[])`

Ejercicio 5. Paso de mensajes (1 punto)

Desarrollar un programa en Java que permita realizar mediante paso de mensajes la suma de matrices.

Suponed que se dispone de una hebra **sumadora** de dos arrays con los datos de las matrices a sumar. Estas matrices son de tamaño NxN, donde N podrá ser leído desde teclado o almacenado como constante y los valores de sus elementos pueden ser aleatorios.

Se crearán N hebras **trabajadoras** de manera que cada hebra se encargará de **solicitar** una fila de cada matriz a la hebra **sumadora**. Una vez recibidas las filas a sumar, las hebras **trabajadoras** proceden a realizar la suma elemento a elemento de las filas recibidas, y una vez finalizada la suma envían el resultado a una hebra **contenedora**.

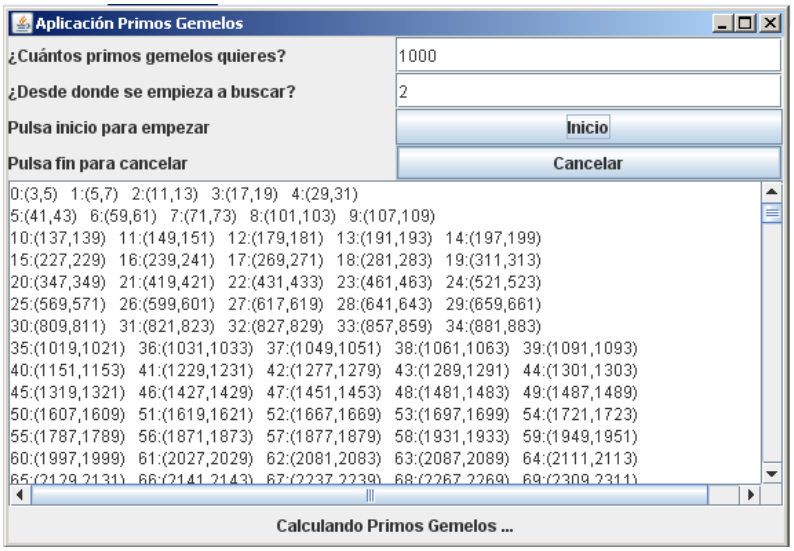
Cuando la hebra **contenedora** tenga el resultado final, lo mostrará por pantalla.

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.

Construir una GUI para 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 siguientes 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.