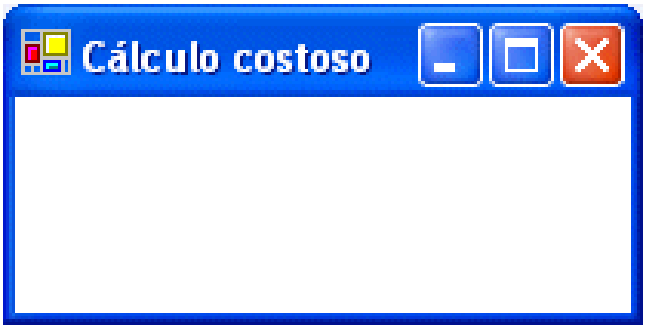
Hebras e interfaces de usuario

Cuando ejecutamos una aplicación con una interfaz gráfica de usuario, automáticamente se crea una hebra que se encarga de procesar todos los eventos asociados a la interfaz de usuario.

Cuando nuestra aplicación ha de realizar alguna operación costosa en tiempo de ejecución, esta operación hemos de realizarla en una hebra aparte. Si no lo hacemos así, podemos encontrarnos con algo como lo siguiente:



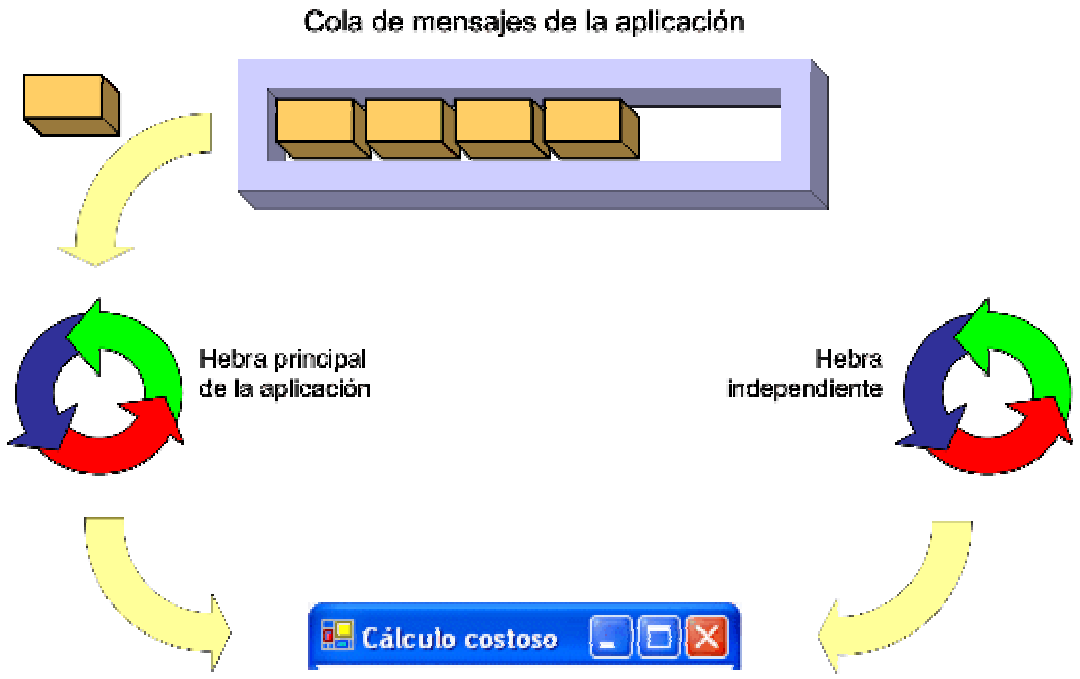
Si mantenemos ocupada la hebra que se encarga de gestionar los eventos de la interfaz de usuario, ésta dejará de responder al usuario (la ventana deja de refrescarse y no podemos controlar la ejecución de la aplicación).

Por tanto, crearemos una hebra independiente cada vez que necesitemos realizar tareas largas (de más de un par de segundos):

Al usar una hebra auxiliar [*worker thread*], nuestra aplicación se comportará como cabría esperar…

… siempre que tengamos cuidado a la hora de utilizar recursos compartidos entre las distintas hebras.

Por ejemplo, si nuestras hebras auxiliares también acceden a los componentes Swing, estamos usando un recurso compartido:



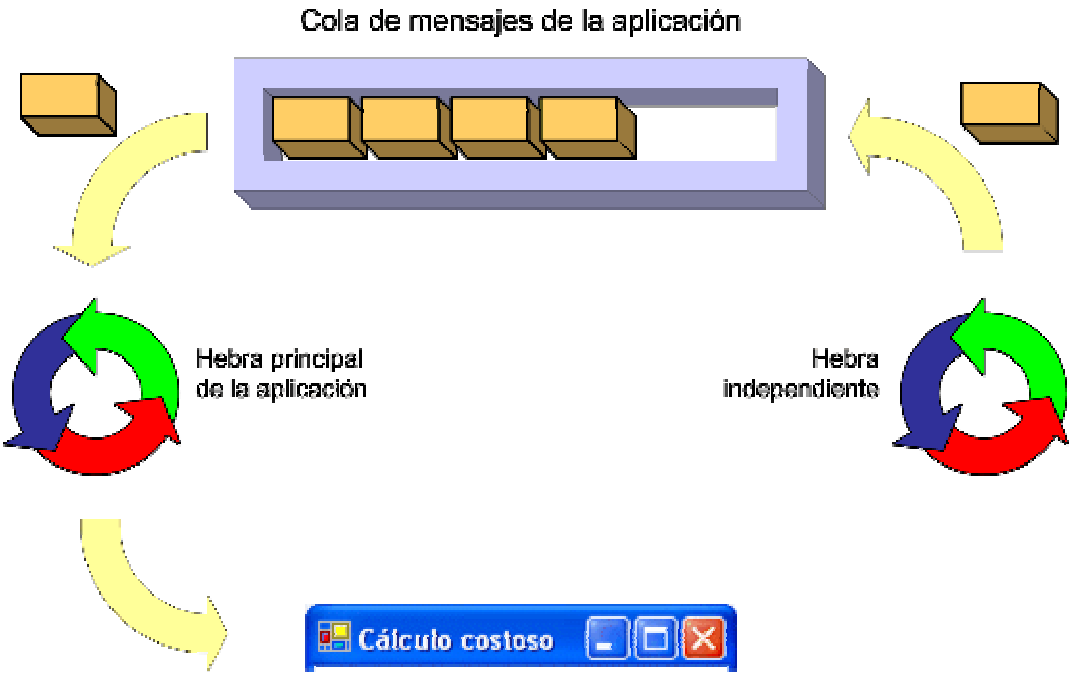
Cuando esto sucede, pueden darse situaciones desagradables si no controlamos el acceso a los recursos compartidos.

La solución más sencilla consiste en eliminar el recurso compartido, de tal forma que sólo una de las hebras pueda acceder a lo que antes era un recurso compartido: se elimina el acceso concurrente a un recurso desde distintas hebras.

Por convención, la hebra responsable de gestionar los eventos de la interfaz de usuario será **siempre** responsable de manipular todos los componentes y controles de nuestra interfaz.

`SwingUtilities.invokeLater()` **SWING**

`== EventQueue.invokeLater()` AWT



Para hacer que la hebra encargada de los eventos se encargue también de ejecutar un fragmento de código, se utiliza el método `SwingUtilities.invokeLater()`.

El método `invokeLater()` recibe como parámetro un objeto que implemente la interfaz `Runnable`.

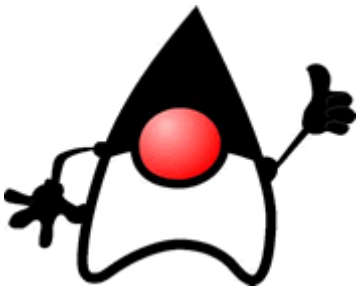
Por tanto, deberemos encapsular en el método `run()` de un objeto `Runnable` lo que queramos hacer sobre los componentes de la interfaz desde una hebra independiente.

> Una variante de `invokeLater()` es el método `SwingUtilities.invokeAndWait()`, que detiene la ejecución de la hebra auxiliar hasta que la hebra principal de la interfaz haya ejecutado el código correspondiente al objeto `Runnable` que se le pasa como parámetro a `invokeAndWait()`.

javax.swing.Timer

Cuando lo que queremos es realizar una operación de forma periódica, no es necesario que creemos explícitamente una hebra.

Podemos utilizar la clase `javax.swing.Timer`, que asociaremos a un `ActionListener` encargado de realizar la operación:



```
// Animación a X fps (frames por segundo)

public class Animator
       extends JFrame
       implements ActionListener
{
  …
  Timer timer;

  public Animator (int fps)
  {
    timer = new Timer(1000/fps, this);
    …
  }

  public void startAnimation()
  {
    timer.start();
  }

  public void stopAnimation()
  {
    timer.stop();
  }

  public void actionPerformed (ActionEvent e)
  {
    // Mostrar el siguiente frame
    …
    repaint();
  }
  …
}
```