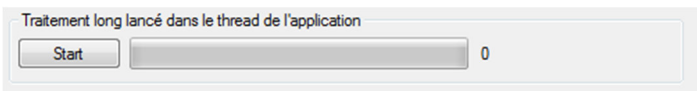
Développement en couches

Module optionnel
Multi-Threading

Le problème

- Chaque application .Net est associée à un thread principal.
  - Chaque action demandée est placée dans la file d'attente associée au thread principal.
  - Chaque action est traitée séquentiellement au sein du thread principal.
- Le multithreading résout les problèmes de débit et de sensibilité.
  - Cela consiste à créer des threads actifs prenant en charge les traitements lourds, laissant ainsi le thread principal piloter les éléments de l'interface utilisateur.
- Attention toutefois, le multithreading peut introduire des problèmes de partage de ressources.



La classe Thread

System.Threading, espace de noms
Thread, classe

- Créer un thread actif

```
// Instanciation du thread, on spécifie dans le
// délégué ThreadStart le nom de la méthode qui
// sera exécutée lorsque l'on appelle la méthode
// Start() de notre thread.
Thread monThread = new Thread(new ThreadStart(traiterAction));
```

- Démarrer un thread

```
// Lancement du thread
monThread.Start();
```

- Stopper un thread

```
private void btStopThread_Click(object sender, EventArgs e)
{
    if (monThread != null && monThread.IsAlive)
        // Détruit notre thread
        monThread.Abort();
}
```

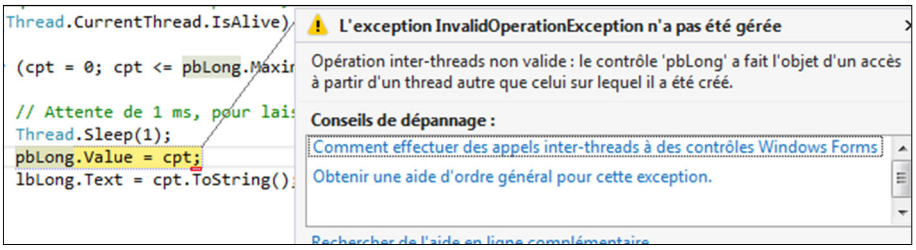

La classe Thread

- Prise en charge du traitement

```
private void traiterAction()
{
    int cpt = 0;
    // Tant que le thread n'est pas tué, on travaille
    while (Thread.CurrentThread.IsAlive)
    {
        for (cpt = 0; cpt <= pbLong.Maximum; cpt++)
        {
            // Attente de 1 ms, pour laisser le temps à l'IHM de se rafraichir
            Thread.Sleep(1);
            pbLong.Value = cpt;
            lbLong.Text = cpt.ToString();
        }
    }
}
```

La classe Thread

- Attention, les ressources ne sont pas **SafeThreading**



Echanger des données avec un Thread

- Créer une classe de gestion du thread qui encapsule :
  - le traitement associé au thread ;
  - les propriétés reçues et utilisées par le traitement ;
  - un délégué permettant au traitement de retourner des données.

```
class GestionThread
{

    public int compteur {get; set;}
    public MultiThreadProject.Form1.DelegateRefreshData Callback { get; set; }

    /// <summary>
    /// méthode appelée lors du démarrage du Thread
    /// respecte la signature du délégué ThreadStart
    /// </summary>
    public void traiterAction()
    {
        int cpt = 0;
        // Tant que le thread n'est pas tué, on travaille
        while (Thread.CurrentThread.IsAlive)
        {
            for (cpt = 0; cpt <= compteur; cpt++)
            {
                // Attente de 1 ms, pour laisser le temps à l'IHM de se rafraichir
                Thread.Sleep(1);

                //Retourner la valeur du compteur au travers de la propriété
                //Callback et appel du délégué (Form1.refreshData)
                if (Callback != null)
                    Callback(cpt);
            }
        }
    }
}
```

Echanger des données avec un Thread

- Signature du délégué

```
//délégué utilisé par le gestionnaire de Thread pour demander le
//rafraichissement du formulaire
public delegate void DelegateRefreshData(int returnData);
```

- Instancier le gestionnaire

```
//
//Instancier la gestionnaire de Thread
//
customThread = new GestionThread()
{
    compteur = pbThread.Maximum,
    Callback = new DelegateRefreshData(this.refreshData)
};
```


Echanger des données avec un Thread

- Instancier et le lancer le Thread

```
// Instanciation du thread, on spécifie dans le
// délégué ThreadStart le nom de la méthode qui
// sera exécutée lorsque l'on appelle la méthode
// Start() de notre thread.
monThread = new Thread(new ThreadStart(customThread.traiterAction));

// Lancement du thread
monThread.Start();
```

Echanger des données avec un Thread

- Mettre à jour l'IHM à partir des données renvoyées par le Thread.

```
/// <summary>
/// méthode appelée par le thread actif demandant le rafraichissement des données
/// </summary>
/// <param name="data"></param>
private void refreshData(int data)
{
    //
    // updateUI s'execute dans le Thread principale.
    //
    this.Invoke(new DelegateRefreshData(updateUI), new Object[] { data });
}

private void updateUI(int data)
{
    pbThread.Value = data;
    lbThread.Text = data.ToString();
}
```


Les fonctions asynchrones

- RI 143
- Disponible depuis la version 5 du langage C#

```
private async void traiterAction()
{
    await Task.Run(() =>
        {
            int cpt = 0;

            for (cpt = 0; cpt <= pbAsync.Maximum; cpt++)
            {
                refreshDataAsync(cpt);
            }
        });

}
```

Les fonctions asynchrones

- Rafraichir le formulaire

```
/// <summary>
/// méthode appelée par le thread actif demandant le rafraichissement des données
/// </summary>
/// <param name="data"></param>
private void refreshDataAsync(int data)
{
    //
    // updateUI s'execute dans le Thread principale.
    //
    this.Invoke(new DelegateRefreshData(updateUIAsync), new Object[] { data });
}

private void updateUIAsync(int data)
{
    pbAsync.Value = data;
    lbAsync.Text = data.ToString();
}
```