



# Threading



- Threads
  - L'unité de base pour laquelle le processeur alloue du temps de calcul
  - Permet une activité multiple qui donne l'apparence simultanée
- Avantages du multi-threading
  - L'application peut travailler en arrière plan sans bloquer l'UI
  - Distinguer les tâches avec des priorités différentes
  - Communiquer avec un réseau, un serveur web et une base de données
- Désavantages potentiels du multi-threading
  - Diminution des performances due à l'accroissement des opérations de gestion des threads, par exemple la commutation entre threads
  - Contrôle de l'exécution du code est complexe avec beaucoup de threads. Il est plus difficile de trouver et corriger un bug.



- Espace de nom du threading :
  - **System.Threading**
- Les différentes classes utiles
  - Thread
  - Timer
  - Monitor
  - Mutex
  - Interlocked
  - Semaphore
  - ThreadPool



## Démarrer un thread

- Créer une nouvelle instance d'un objet Thread
  - L'appel du constructeur nécessite un delegate **ThreadStart** avec un seul paramètre.
  - **ThreadStart** référence la méthode qui sera exécuter par le nouveau Thread
- Le thread ne s'exécute pas temps que la méthode Thread.Start n'est pas appelée.

```
Thread t = new  
    Thread(new ThreadStart(MaClass.UnemethodeStatique));  
t.Start();
```



## Les propriétés et paramètres d'un thread

- Utiliser les propriétés `Thread.Name` et `Thread.Priority` pour lire ou écrire le nom et la priorité du thread.
- Définir si le thread s'exécute en arrière-plan ou en avant-plan en utilisant `Thread.IsBackground`.
  - En arrière-plan le thread n'empêchera pas un processus de s'arrêter.

```
t.Name = "My Background Thread";  
t.Priority = ThreadPriority.AboveNormal;  
t.IsBackground = true;
```

- Encapsuler les paramètres du thread dans un objet



- **Thread.Sleep** cause le blocage du thread

```
Thread.Sleep(3000); // bloc pendant 3 secondes
```

- les méthodes **Suspend** et **Resume** ne sont généralement pas utilisées
  - elles peuvent provoquer des problèmes de deadlocks.
- **Thread.Join** attend la fin d'un autre thread

```
t.Start();  
t.Join(); // Wait for the thread to exit
```

- la méthode **Thread.WaitHandle** attend sur un ou plusieurs événements.

```
WaitHandle.WaitAll(waitEvents);
```

- la propriété **Thread.ThreadState** est un bit-mask de l'état du thread



- La méthode `Thread.Interrupt`
  - Réveille un thread dont l'état est **WaitSleepJoin** et provoque une **ThreadInterruptedException**
- la méthode `Thread.Abort` arrête définitivement un thread
  - Arrête un thread dans n'importe quel état et provoque une **ThreadAbortException**
  - **ThreadAbortException** est une exception spéciale car si elle est attrapée dans un bloc catch, on peut annuler Abort en appelant **ResetAbort**.
  - L'arrêt immédiat n'est pas garanti, Immediate abort not guaranteed, Utiliser **Thread.Join** pour bloquer un thread à la fin.