Thread Ninja

*don't forget `using CielaSpike;`

HOW TO: Start an async co-routine

- Option 1: Similar to `StartCoroutine()`

```
// this keyword is necessary to call an extension method.
this.StartCoroutineAsync(YourRoutine());
// or
yield return this.StartCoroutineAsync(YourRoutine());
```

- Option 2: Create a task manually

```
var task = new Task(YourRoutine());
// pass it to StartCoroutine to launch it.
StartCoroutine(task);
// or do them together.
Task task2;
this.StartCoroutineAsync(YourRoutine(), out task2);
```

HOW TO: Do async jobs

- Every task or async co-routine starts in background

```
IEnumerator YourAsyncRoutine()
{
  // now on background;
  Destory(gameObject); // Error! => access Unity API on background thread.
}
```

- Access Unity API in background

```
IEnumerator YourAsyncRoutine()
{
  // now on background;
  yield return Ninja.JumpToUnity; //=> here's our ninja, following code runs on main thread.
  Destory(gameObject); // OK!
  yield return Ninja.JumpBack; //=> go back to background.
  // continue doing some heavy computing without blocking the main thread.
  // ...
}
```

HOW TO: Wait a task to complete

```
Task task;
this.StartCoroutineAsync(YourRoutine(), out task);
// do sth else.
// wait task to complete.
yield return StartCoroutine(task.Wait());
// task has done.
if (task.State == TaskState.Error)
{ // ... }
```

HOW TO: Cancel an async co-routine

```
Task task;
this.StartCoroutineAsync(YourRoutine(), out task);
// ...
// when you want to cancel it.
task.Cancel();
Debug.Log(task.State); //=> Cancelled
```

HOW TO: Handle error

- A task or an async co-routine will catch & log exceptions even on background thread.

```
Task task;
yield return this.StartCoroutineAsync(YourRoutineThrowsException(), out task);
if (task.State == TaskState.Error)
{
    var ex = task.Exception;
    // check the exception object.
    // ...
}
```