ИСКЛЮЧЕНИЯ В ЗАДАЧАХ

Генерация исключения *NullReferenceException*

```
Task task = Task.Run (() => { throw null; });
try
{
  task.Wait();
}
catch (AggregateException aex)
{
  if (aex.InnerException is NullReferenceException)
    Console.WriteLine ("Null!");
  else
    throw;
}
```

Одновременная генерация двух и более исключений

```
Task t1 = new Task(() =>
{
  throw new OutOfMemoryException();
});
Task t2 = new Task(() =>
{
  throw new DivideByZeroException();
});
t1.Start(); t2.Start();
try
{
  Task.WaitAll(t1, t2);
}
catch (AggregateException aex)
{
  foreach (Exception ex in aex.InnerExceptions)
    Console.WriteLine(ex.Message);
}
```

Flatten

```
Task[] tasks = new Task[N];
// Объявляем и запускаем задачи
..
// Обработка исключений
try
{
  Task.WaitAll(tasks);
}
catch (AggregateException ae)
{
  foreach (Exception e in ae.Flatten().InnerExceptions)
    Console.WriteLine("Message:{ 0}", e.Message);
}
```

Handle

```
public void Handle (Func<Exception, bool> predicate)
```

Если предикат возвращает ***true***, то считается, что исключение «обработано». После того, как делегат запустится на всех исключениях, произойдет следующее:

- если все исключения были «обработаны» (делегат возвратил ***true***), то исключение не генерируется повторно;
- если были исключения, для которых делегат возвратил ***false*** («необработанные»), то строится новый объект ***AggregateException***, содержащий такие исключения, и затем он генерируется повторно.

Исключения и автономные задачи

Необработанные исключения в автономных задачах называются необнаруженными исключениями, и среда CLR будет повторно генерировать исключение в потоке финализаторов, когда объект задачи покидает область видимости и обрабатывается сборщиком мусора). Подписаться на необнаруженные исключения на глобальном уровне можно через статическое событие ***TaskScheduler.UnobservedTaskException***