# Managing Resources

## The Dispose Pattern

1. Open the solution **Jobs** from the **Start** folder in this lab.
2. General note: you’ll have to run this app from the command line.
3. Implement the constructor of the **Job** class, that accepts a string called “**name”**:
   1. Call **NativeJob.CreateJobObject** passing **IntPtr.Zero** and the **name** argument. Store the returning handle in **\_hJob**.
   2. If the handle is zero (**IntPtr.Zero**), throw an **InvalidOperationException**.
   3. Create the **\_processes** object.
4. Implement the **IDisposable** interface on the **Job** class. Use the following guidelines:
   1. Make use of the Dispose pattern, i.e. create a **Dispose(bool)** method as discussed in the course.
   2. Make sure calling **Dispose** multiple times is harmless, while calling anything substantial if the object is disposed throws a **ObjectDisposedException**.
5. Implement the **Kill** method by calling **NativeJob.TerminateJobObject**.
6. In the Main method, create a Job object, and assign some processes to it (Use **Process.Start** to create some simple processes, such as “notepad” or “calc”).
7. Call **Console.ReadLine** and after the user hits <enter> kill all processes in the job using the **Kill** instance method.
8. Part B:
   1. Let’s assume the Native Job is holding a lot of native memory
   2. Add a call in the Job ctor to GC.AddMemoryPressure(sizeInByte) and display a message that the Job was created
   3. Add a call in the finalizer to GC.RemoveMemoryPressure(sizeInBytes) and display a message that the Job was released
   4. Create a loop in your main method that creates 20 Job objects
   5. See what happens when you run the application with different “sizeInBytes”. Try 0 MB and 10 MB