**Creating Practical Asynchronous Solutions**

**Problem 1: Downloading Files Asynchronously**

**Problem Description:**

In the code provided, some of the keywords related to asynchronous programming in C# have been removed. Your task is to fill in the blanks (\_\_\_\_\_) with the correct async or await keywords to make the code function properly.

* Use async where you need to indicate that a method is asynchronous.
* Use await where you need to wait for an asynchronous operation to complete.

**Remember that:**

* Methods that return Task or Task<T> should be marked with the async keyword.
* Operations that need to be awaited must have the await keyword before them.

Once you have completed the exercise, the code should download two files concurrently and print messages indicating the start and completion of each download.

**Code:**

**public** **class** **Problem1**

{

**public** **async** Task<**string**> DownloadFileAsync(**string** fileName)

{

Console.WriteLine($"Starting download of {fileName}...");

**await** Task.Delay(**3000**); // Simulate a 3-second download time

Console.WriteLine($"Completed download of {fileName}.");

**return** $"{fileName} content";

}

**public** **async** Task **DownloadFilesAsync**()

{

// Start downloading "File1.txt" and "File2.txt" concurrently

**var** downloadTask1 = DownloadFileAsync("File1.txt");

**var** downloadTask2 = DownloadFileAsync("File2.txt");

// Wait for both downloads to complete

**await** Task.WhenAll(downloadTask1, downloadTask2);

Console.WriteLine("All downloads completed.");

}

}

**Problem 2: Processing Data Chunks Asynchronously**

**Task:**

In the code provided, some of the asynchronous programming keywords have been removed. Your task is to correctly fill in the blanks (\_\_\_\_\_) using either async or await based on the context.

* **Use async when defining a method that will perform asynchronous operations and return a Task.**
* **Use await where the code needs to pause and wait for an asynchronous operation to complete before continuing.**

Once you have filled in the blanks, the code should asynchronously process chunks of data concurrently, and display messages when each chunk starts and completes processing.

**Code:**

**namespace** **CreatingPracticalAsyncSol**

{

**public** **class** **Problem2**

{

**public** **async** Task **ProcessDataChunkAsync**(**int** chunkNumber)

{

Console.WriteLine($"Processing chunk {chunkNumber}...");

**await** Task.Delay(**1000**); // Simulate processing time

Console.WriteLine($"Completed processing of chunk {chunkNumber}.");

}

**public** **async** Task **ProcessLargeDatasetAsync**(**int** numberOfChunks)

{

**var** tasks = **new** List<Task>();

// Start processing each chunk concurrently

**for** (**int** i = **1**; i <= numberOfChunks; i++)

{

tasks.Add(ProcessDataChunkAsync(i));

}

// Wait for all tasks to complete

**await** Task.WhenAll(tasks);

Console.WriteLine("All data chunks processed.");

}

}

}

**Program.cs:**

**using** **CreatingPracticalAsyncSol**;

**class** **Program**

{

**public** **static** **async** Task **Main**(**string**[] args)

{

Console.WriteLine("=== Running Program 1 ===");

**var** p1 = **new** Problem1();

**await** p1.DownloadFilesAsync();

Console.WriteLine("\n=== Running Program 2 ===");

**var** p2 = **new** Problem2();

**await** p2.ProcessLargeDatasetAsync(**5**); // Example with 5 chunks

Console.WriteLine("\n=== All Programs Finished ===");

}

}