**Concurrency: Multithreading Parallel and Async Code**

**Creating Images from a Fibonacci Sequence**

**Solution:**

1. In your **Workshop5**, run the dotnet command to add the **System.Drawing.Common** reference:

>dotnet add package System.Drawing.Common

1. Include **System.IO** to create a temporary folder and **System. Globalization** to parse input strings into numbers:

using System;

using System.Drawing.Imaging;

using System.Globalization;

using System.IO;

using System.Threading;

using System.Threading.Tasks;

namespace Workshop5

{

public static class FibonacciConsole

{

1. Add the standard static **Main** entry point (you won't need the **async** keyword as you are not awaiting at this point):

public static void Main()

{

1. Use **Path.GetTempPath()** to find the name of the **Temp** folder and add **Fibonacci** as a subfolder name:
2. var tempImagePath = Path.GetTempPath() + "Fibonacci\\";

If that temporary folder doesn't exist, create the folder (this will work on all systems, not just Windows):

if (!Directory.Exists(tempImagePath))

{

Console.WriteLine($"Creating temp folder:

{tempImagePath}");

Directory.CreateDirectory(tempImagePath);

}

else {

Console.WriteLine($"Using temp folder:

{tempImagePath}");

}

1. Use **CancellationTokenSource** to signal that the calculation and image export task should stop.
2. Declare it but leave it as null at this stage:

CancellationTokenSource tokenSource = null;

1. Start a **do**-loop:

do {

Console.Write("Phi (eg 1.0 to 6.0) (x=quit, enter=cancel):");

var input = Console.ReadLine();

You want the user to be able to enter a value for **phi** as many times as they want to. Prompt for a value between **1.0** and **6.0**, as these produce interesting patterns, and show the quit and cancel message.

1. If the user enters a blank value, use that as an opportunity to cancel any other pending calculations:

if (string.IsNullOrEmpty(input))

{

Console.WriteLine("Cancelling previous tasks");

tokenSource?.Cancel();

continue;

}

This allows the other tasks to complete in time. Notice the null check before calling **Cancel**. If this is the first time the loop has run, the token will be null.

1. Check to see if **x** was entered and if so, **break** out of the **do**-loop to end:

if (input == "x")

{

break;

}

1. Use **double.TryParse** to convert the user's input into a double number:

if (!double.TryParse(input, NumberStyles.Any, CultureInfo.CurrentCulture, out double phi))

{

continue;

}

1. Prompt for the number of images to create, using **int.TryParse** to convert the **Console.ReadLine()** string into an integer, and tell the user you are about to start the process:

Console.Write("Image Count (eg 1000):");

input = Console.ReadLine();

if (!int.TryParse(input, NumberStyles.Any,

CultureInfo.CurrentCulture, out int imageCount))

{  
continue;

}

Console.WriteLine($"Creating {imageCount}

images...");

1. You can't reuse a token once it has been cancelled, so create a new token source.
2. Attach an event handler to the **cancel** token for extra information. This token can then be passed to **Task.Ru** and to the **GenerateImageSequences** method:

tokenSource = new CancellationTokenSource();

tokenSource.Token.Register(

() => Console.WriteLine("Cancelled!"));

var token = tokenSource.Token;

Task.Run(() => GenerateImageSequences(tempImagePath,

phi, imageCount, token), token);

}

while (true);

}

18. Add the **GenerateImageSequences** method:

private static async Task GenerateImageSequences(string

tempImagePath, double phi, int imageCount, CancellationToken token)

{

const double PhiIncrement = 0.015D;

const int Points = 3000;

const int ImageSize = 800;

const int PointSize = 5;;

const string FileExtension = ".png";

var fileFormat = ImageFormat.Png;

The **GenerateImageSequences** method does the work of creating multiple sequences and exporting their resulting image. Here, you have defined constants for some of the parameters. It may have been too much to prompt the user to enter these values repeatedly. Note that you need the **async** keyword as this contains **awaitable** code.

1. Use a **for** loop to iterate through the number of images requested.
2. In each loop, await a call to **FibonacciSequence.Calculate** passing in the points per image and the current value of **phi**:

for (var i = 0; i < imageCount; i++)

{

phi += PhiIncrement;

var sequence = await Task.Run(

() => FibonacciSequence.Calculate(Points, phi),

token);

You are returned a list of **Fibonacci** items. You saw earlier that loop-based code can check a token's **IsCancellationRequested** status.

1. If that is the case, use the **break** statement to stop any more image files from being created. The user may have requested a cancellation:

if (token.IsCancellationRequested)

{

break;

}

1. Now add another awaitable block using **Task.Run**:

var imagePath = $"{tempImagePath}Fibonacci\_{Points}\_{phi:N3} {FileExtension}";

await Task.Run( () => ImageGenerator.ExportSequence(sequence, imagePath,

fileFormat, ImageSize, ImageSize, PointSize), token);

}

}

}

}

This will call **ImageGenerator.ExportSequence**, passing in a unique filename based on the number of points, the current value of **phi**, and required image details.

23. Run the console app to produce this output:

Using temp folder: C:Temp\Fibonacci\

Phi (eg 1.0 to 6.0) (x=quit, enter=cancel):1

Image Count (eg 1000):1000

Creating 1000 images...

Phi (eg 1.0 to 6.0) (x=quit, enter=cancel):

20:36:19 [04] Saved Fibonacci\_3000\_1.015.png

20:36:19 [06] Saved Fibonacci\_3000\_1.030.png

20:36:19 [05] Saved Fibonacci\_3000\_1.045.png

20:36:20 [08] Saved Fibonacci\_3000\_1.060.png

20:36:20 [08] Saved Fibonacci\_3000\_1.075.png

20:36:20 [06] Saved Fibonacci\_3000\_1.090.png

20:36:20 [05] Saved Fibonacci\_3000\_1.105.png

20:36:20 [05] Saved Fibonacci\_3000\_1.120.png

20:36:20 [05] Saved Fibonacci\_3000\_1.135.png

20:36:20 [05] Saved Fibonacci\_3000\_1.150.png

20:36:20 [07] Saved Fibonacci\_3000\_1.165.png

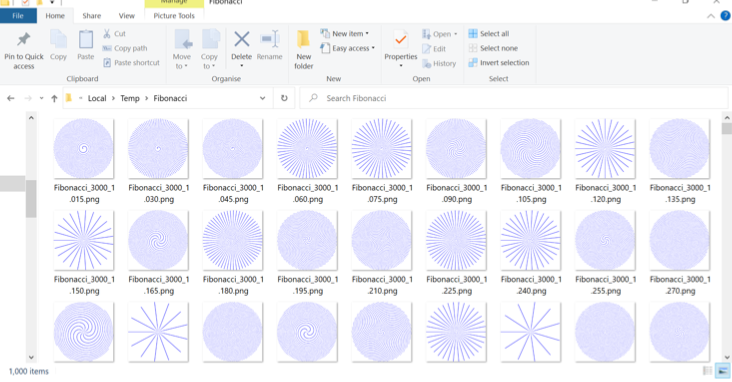
20:36:20 [07] Saved Fibonacci\_3000\_1.180.png

20:36:20 [07] Saved Fibonacci\_3000\_1.195.png

Cancelling previous tasks

Cancelled!

Try entering different values for **phi** to see how that affects the style of images created. From the output, you can see that various threads have been used  
to create the images and many threads are used more than once. By pressing **Enter**, the process is stopped:



**Windows 10 Explorer image folder contents (a subset of images produced)**

In this activity, you created a console application that allowed various inputs to be passed to a sequence calculator. Once you had entered your parameters, the app will start the time-consuming task of creating 1,000 images. You used the **FibonacciSequence.cs** and **ImageGenerator.cs** files and created a console app that will generate the .png files.