I4GUI Opgave CalcPi

Kommentarer til løsningsforslag

Lab1

Der er funktionen CalcPi som gør det tunge arbejde, så den flytter jeg til en tråd. For at få en mere overskuelig struktur opretter jeg en klasse BackgroundThread, som får til opgave at starte CalcPi i en tråd. Da CalcPi nu ligger i en "underklasse" til MainWindow, så synes jeg ikke at den skal kalde metoder i MainWindow, så jeg opretter 2 events, som bruges til at kommunikere med MainWindow (MainWindow subscriber til disse events). Bemærk også klasse PiUpdateEventArgs, som bruges til at sende data med evUpdate event.

Eventhandlerne kaldes fra baggrundstråden og må derfor ikke ændre på GUI-kontrollerne direkte, men skal via dispatcheren få koden afviklet på GUI-tråden. Dette gøres med lidt forskellig syntaks, for at demonstrerer de forskellige muligheder. Personligt foretrækker jeg den korte version:

```
Dispatcher.BeginInvoke(new Action(()=>
{
    // Show result
    tblkResults.Text = worker.Pi;
    btnCalculate.IsEnabled = true;
    sbiStatus.Content = "Ready";
    progressBar.Visibility = Visibility.Hidden;
}));
```

Lab2

Når man benytter BackgroundWorker bliver threadingen pakket ind så man ikke behøver særlig viden om threading (men mindre man bruger shared state inde i DoWork-eventhandleren). CalcPi bruges som DoWork-eventhandler, og ShowProgress og Completed hægtes på eventene ProgressChanged og RunWorkerCompleted. Alle 3 events på BackgrounWorkeren kræver en speciel signatur, så signaturen for de 3 funktioner må tilpasses den signatur eventene kræver.

```
        bgworker = new BackgroundWorker();
        bgworker.DoWork += new DoWorkEventHandler(CalcPi);
        bgworker.ProgressChanged += new ProgressChangedEventHandler(bgworker_ProgressChanged);
bgworker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(bgworker_RunWorkerCompleted);
        bgworker.WorkerReportsProgress = true;
        bgworker.WorkerSupportsCancellation = true;
        bgworker.RunWorkerAsync(digits);
```

Fordelen ved at bruge BackgroundWorker er primært, at vi slipper for at benytte dispatcheren, da det er indbygget i BackgroundWorker. Ulempen er den fastlåste struktur.

Lab3

I lab3 benyttes Parallel.For. Alle tråde laves indirekte af Parallel klassen, og metoden For returnerer ikke før alle tråde er afsluttet, så derfor er selve kaldet til Parallel.For flyttet ud i en tråd som i lab1.

```
StringBuilder pi = new StringBuilder("3", totalDigits + 2);
```

```
int numCalc = (totalDigits + 8) / 9;
string[] results = new string[numCalc];

if (totalDigits > 0)
{
    pi.Append(".");

    Parallel.For(0, numCalc, i =>
        {
            int nineDigits = NineDigitsOfPi.StartingAt(1 + i * 9);
            int digitCount = Math.Min(totalDigits - i * 9, 9);
            string ds = string.Format("{0:D9}", nineDigits);
            results[i] = ds.Substring(0, digitCount);
        });

    for (int i = 0; i < numCalc; i++)
    {
        pi.Append(results[i]);
    }
```

Lab4

Lab 4 er en alternativ løsning af lab3, hvor jeg selv bestemmer antallet af cores og opretter et tilsvarende antal tråde:

```
countCores = Environment.ProcessorCount;
```

Beregner antal decimaler pr. beregingstråd:

```
int digitsPrThread = (int)Math.Ceiling((double)totalDigits / (double)countCores);
```

Opretter det beregnede antal backgrounThread-objekter og starter dem:

```
for (int i = 0; i < countCores; i++)
{
    worker[i] = new BackgroundThread(i);
}

for (int i = 0; i < countCores; i++)
{
    worker[i].evCompleted += new EventHandler<CalcPiEventArgs>(worker_evCompleted);
    start = 0 + i * digitsPrThread;
    stop = Math.Min((i + 1) * digitsPrThread, totalDigits);
    worker[i].CalcPi(start, stop);
}
```

Det største problem i denne opgave er nok at finde ud af hvornår alle tråde er afsluttet, og så få opdateret brugergrænsefladen. Dette gør jeg i evCompleted-eventhandleren, hvor der checkes om alle tråde er completed:

```
bool allThreadsCompleted = true;
lock (this)
    foreach (BackgroundThread bgw in worker)
        if (bgw.IsCompleted == false)
            allThreadsCompleted = false;
if (allThreadsCompleted)
{
    endTime = DateTime.Now;
    this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)delegate
    {
        // Show result
        …
    });
}
```