Aszinkron programozás C#-ban

Aszalos Attila

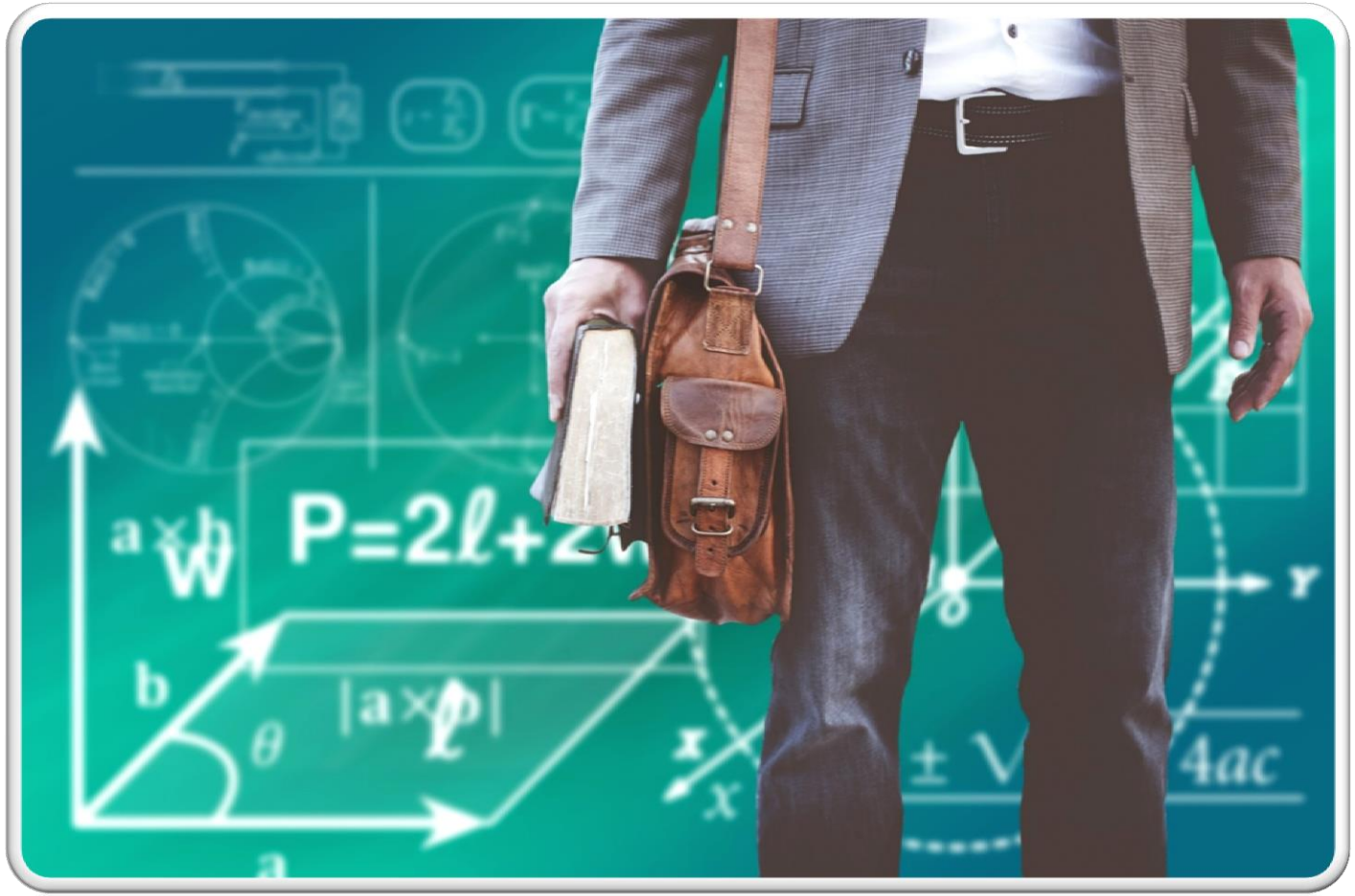
TOPMOTIVE

2019. március 14.

DYSE RO

Tartalom

- Bemutatkozás
- Bevezető
- Az aszinkron programozás alapjai
- Könyváruház applikáció
- Összefoglaló



Bemutatkozás (1)

member of TOPMOTIVE® group
IT solutions for the automotive market
www.dvse.ro

Csapatok

- WPF
- Web
  - MVC
  - Angular
- Mobil
  - Android – Java, Kotlin
  - iOS – Objective C, Swift
- Teszt

Dózsa György utca, 64 – 68 szám
Multinvest Business Center 2, II. emelet
Telefon: +40 265 265487
E-Mail: office@dvse.ro

Bemutatkozás (2)

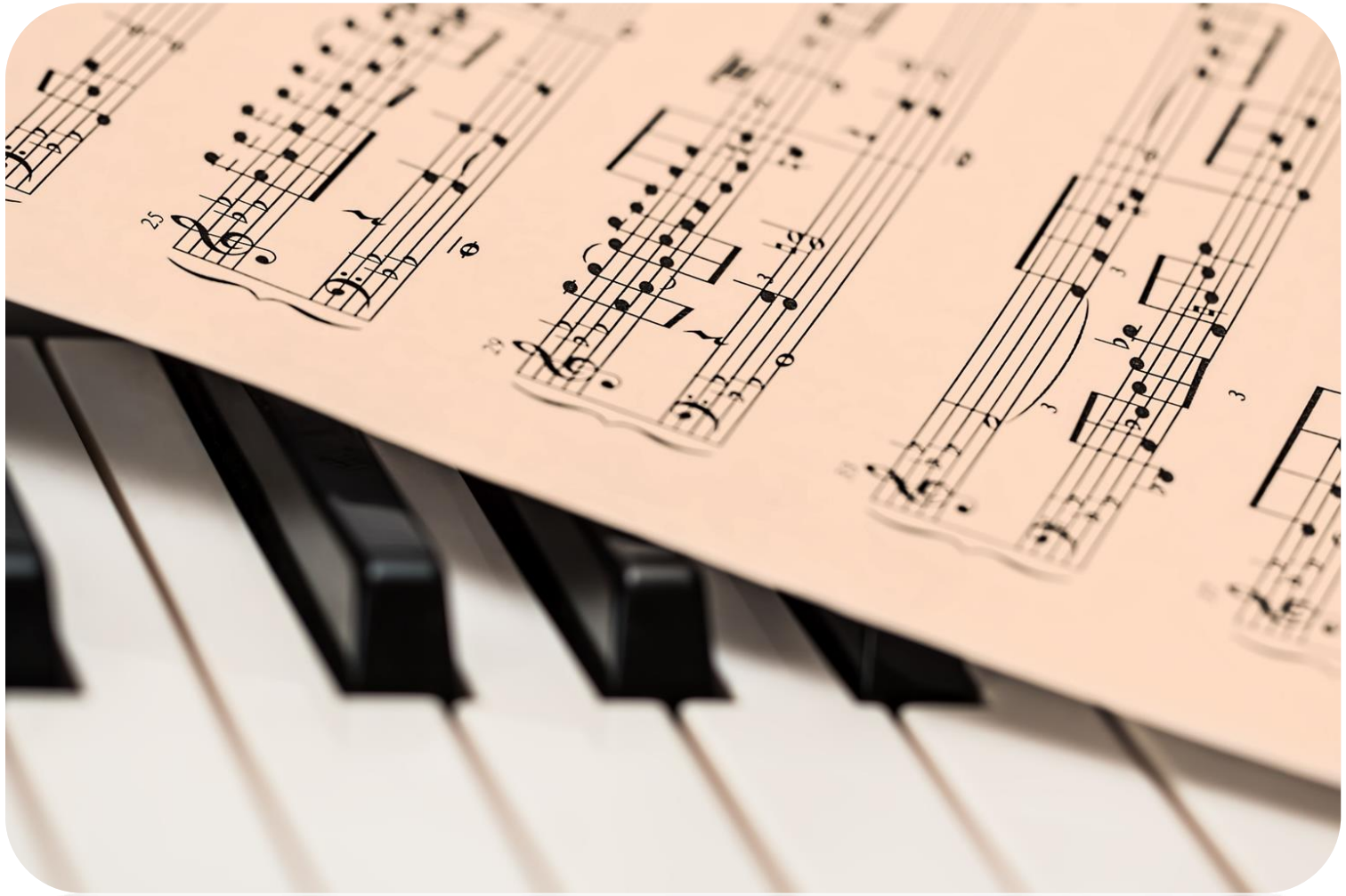
Volt haj…

1998

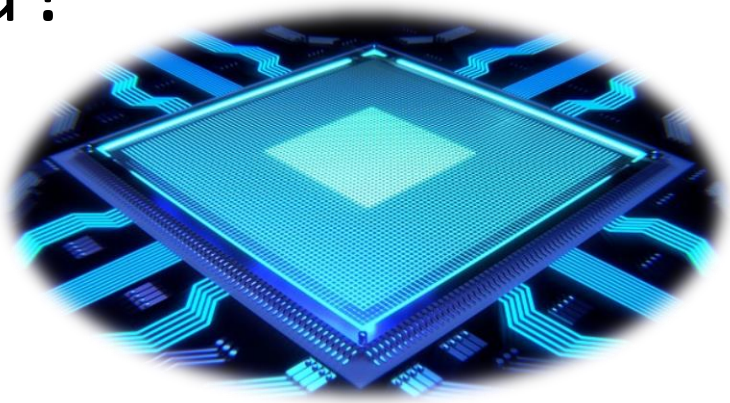
2006

Nincs haj…

Bevezető



Hogyan hajtódik végre a forráskód?



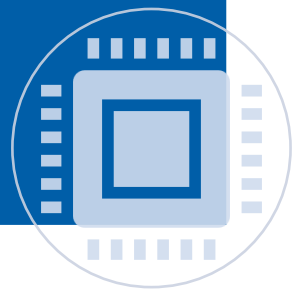
```
private void ExportButton_Click(object sender, RoutedEventArgs e)
{
    this.ExportProgressBar.Visibility = Visibility.Visible;
    List<BookDetails> allDetails = new List<BookDetails>();
    foreach (Book book in SearchResultsItemsControl.ItemsSource.Cast<Book>().Where(p => p.IsChecked))
    {
        BookDetails bookDetails = this.dataService.GetDetails(book);
        allDetails.Add(bookDetails);
    }

    BookDetailsExporter.Export("my_books", allDetails);
    this.ExportProgressBar.Visibility = Visibility.Collapsed;
}
```

Alapvető művelettípusok

- Rendező algoritmusok
- Kereső algoritmusok
- Kriptográfiai algoritmusok
- Backtrack algoritmusok
- Utazó ügynök probléma

Számítási művelet



- Hálózati műveletek
  - Letöltés
  - Feltöltés
- Fájlműveletek
  - Írás
  - Olvasás

Erőforrás hozzáférési művelet

Mérjük le!

Eredmény

Amit tudunk

- A forráskód szekvenciális
- Egyszerre csak 1 művelet hajtódik végre
- Létezik blokkoló hívás

Amit láttunk

- Megfagy az ablak
- Megfagy a „*progress bar*”
- Megfagy a „*hover*” effektus

Az aszinkron programozás alapjai

Szálak – Amit már tanultunk…

Az operációs rendszer több folyamatot is futtat

- Sorban
- Egyidejűleg

A folyamatok osztoznak az erőforrásokon

- Processzor
- Memória

Egy folyamaton belül létezhet egy vagy több szál

- Minding van egy fő szál („*Main Thread*”)
- „Párhuzamosság illúzió” vs „Valódi párhuzamosság”

A .NET keretrendszer fejlődése

.NET 1.0 *13.02.2002*

System.Threading

- Thread
- Timer

.NET 2.0 *07.11.2005*

System.ComponentModel

- BackgroundWorker

.NET 4.0 *12.04.2010*

System.Threading

- Task

Írjuk át!

BackgroundWorker egyszerű használat

```
private void FindPrime_Click(object sender, RoutedEventArgs e)
{
    BackgroundWorker backgroundWorker = new BackgroundWorker();
    backgroundWorker.DoWork += BackgroundWorker_FindNthPrime;
    backgroundWorker.RunWorkerCompleted += BackgroundWorker_FindNthPrimeCompleted;
    backgroundWorker.RunWorkerAsync();
}

private void BackgroundWorker_FindNthPrime(object sender, DoWorkEventArgs e)
{
    int prime = PrimeTool.FindNthPrime(n);
    e.Result = prime;
}

private void BackgroundWorker_FindNthPrimeCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    ResultLabel.Content = $"The {n}th prime is: {e.Result}";
}
```

Konfigurálás

Nem blokkoló hívása a DoWork-nek

Háttér szál

Fő szál

Vigyázat, nem mindegy…

Aszinkron

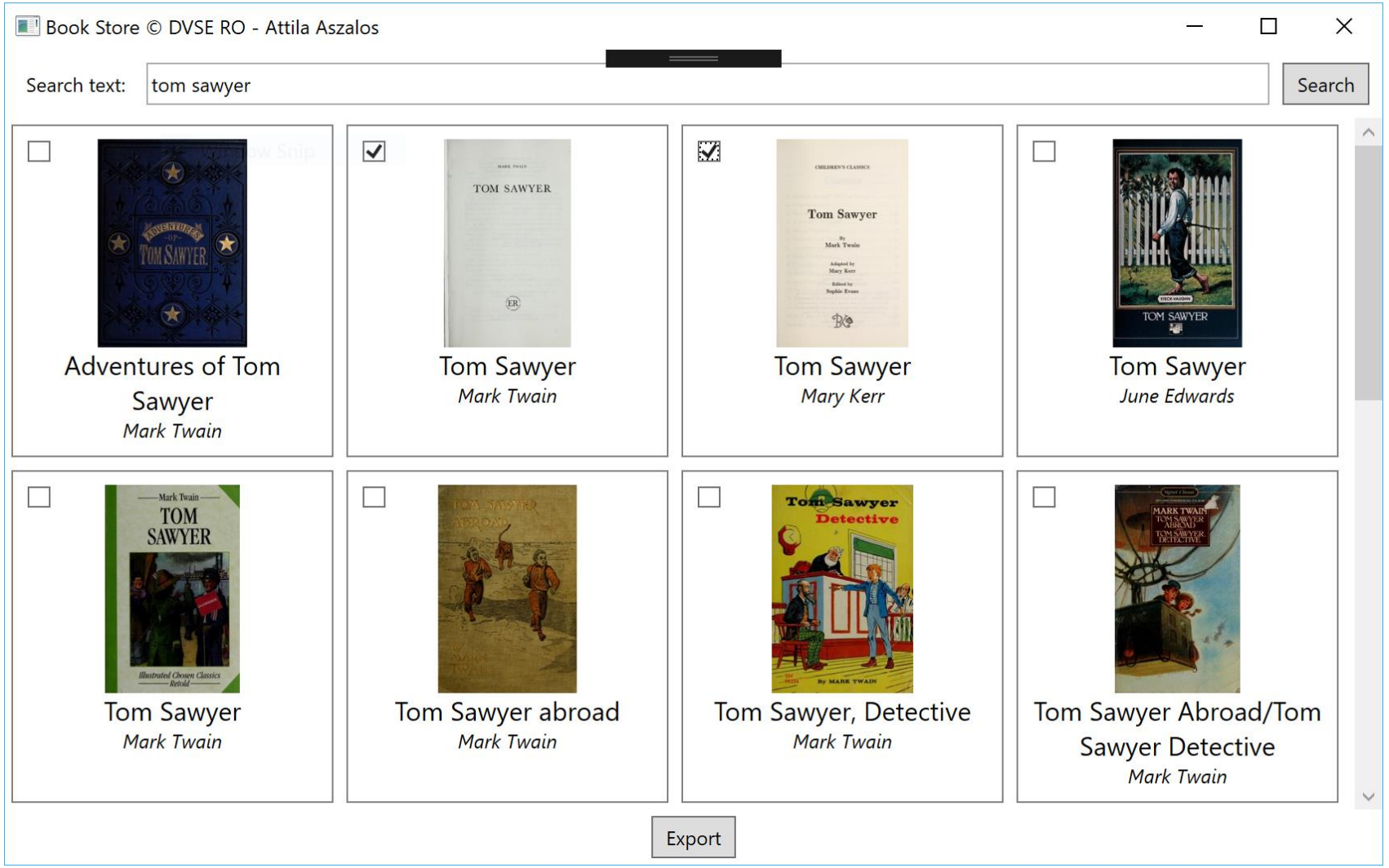
- Futhat háttérszálon
- Futhat a fő szálon is
  - Példa:

```
Debug.WriteLine("Before begin invoke");
Dispatcher.BeginInvoke((Action)(() =>
{
    Debug.WriteLine("In begin invoke");
}));
Debug.WriteLine("After begin invoke");
```

Háttér szál

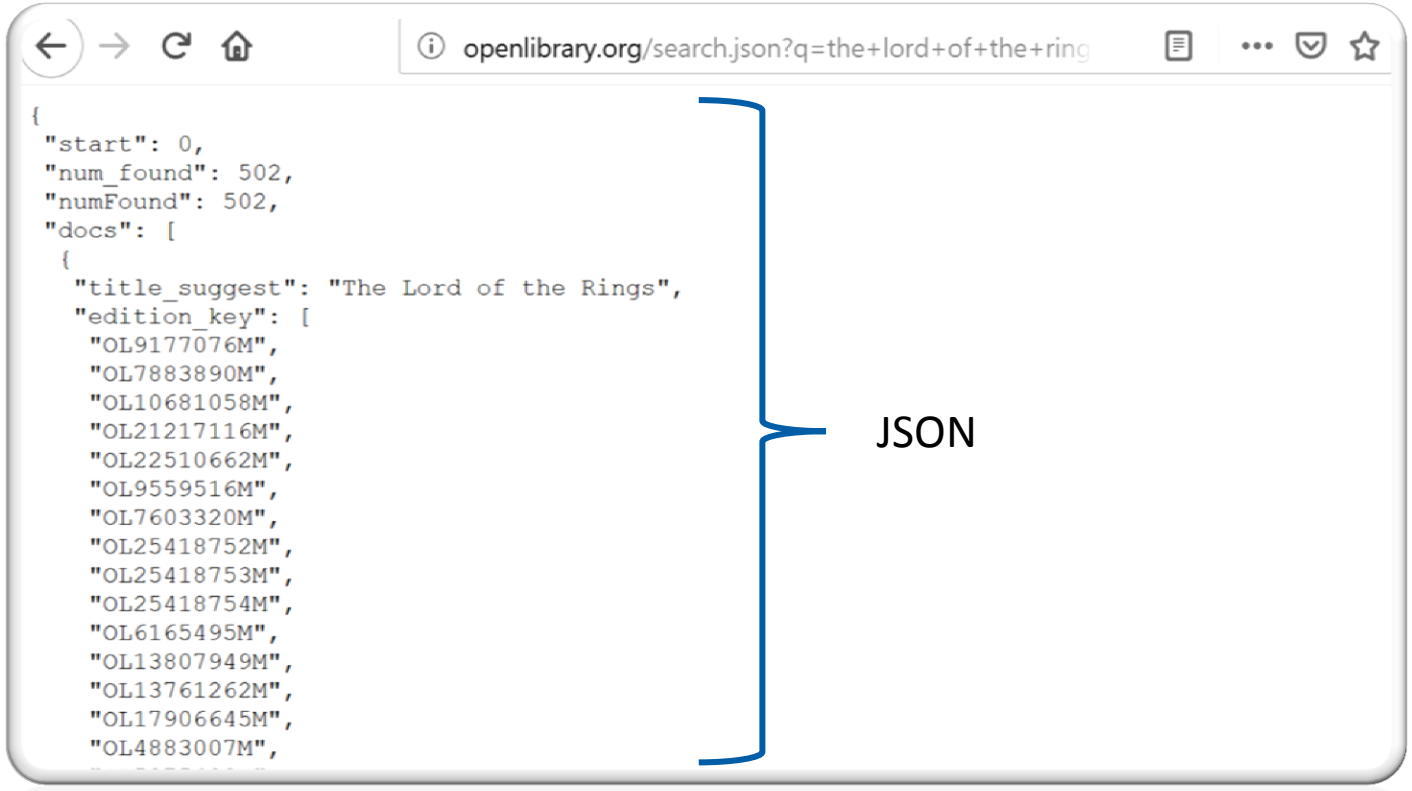
- Mindig aszinkron módon indul el

Könyváruház applikáció



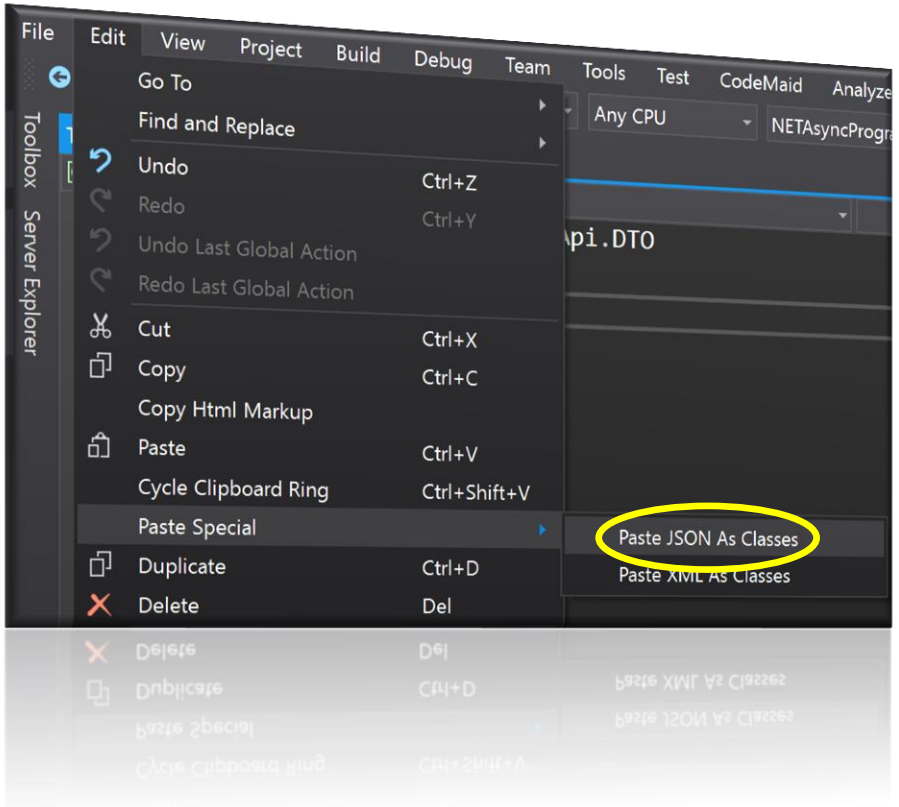
OPEN LIBRARY API

- Fejlesztői dokumentáció: https://openlibrary.org/developers/api
- Példa: http://openlibrary.org/search.json?q=the+lord+of+the+rings



OPEN LIBRARY API használata C#-ból (1)

- DTO-k (Data Transfer Object) létehozása




```
internal class SearchResponse
{
    public int start { get; set; }
    public int num_found { get; set; }
    public int numFound { get; set; }
    public SearchResultItem[] docs { get; set; }
}
internal class SearchResultItem
{
    public string title_suggest { get; set; }
    public string[] publisher { get; set; }
    public string[] isbn { get; set; }
    public string title { get; set; }
    public string[] author_name { get; set; }
    public string key { get; set; }
    public string[] text { get; set; }
    public string[] author_key { get; set; }
    public string type { get; set; }
    public string cover_edition_key { get; set; }
    public string[] subject { get; set; }
    ...
}
```

OPEN LIBRARY API használata C#-ból (2)

- GET HTTP kérés küldése:

```
string address = "http://openlibrary.org/search.json?q=the+lord+of+the+rings";
WebClient client = new WebClient();
string responseText = client.DownloadString(address);
```

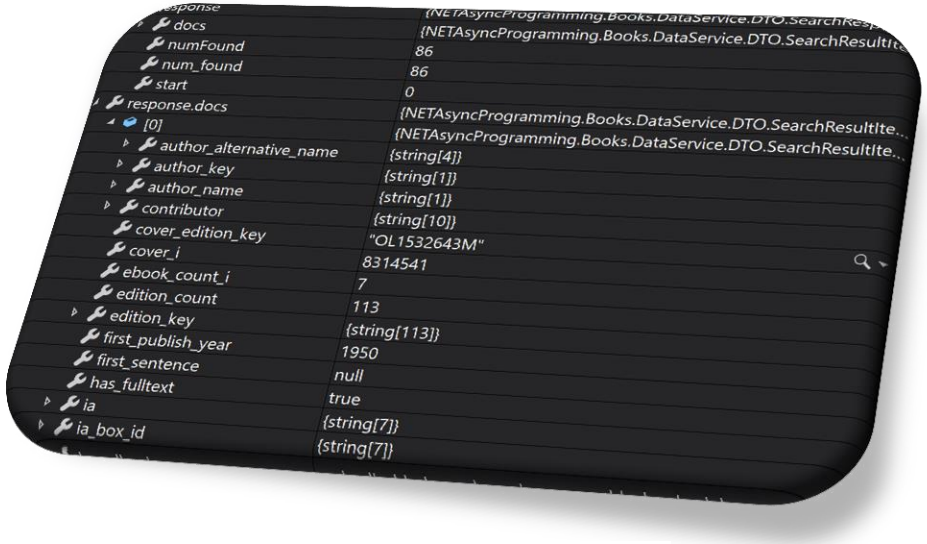
Newtonsoft

```
{
 "start": 0,
 "num_found": 86,
 "numFound": 86,
 "docs": [
  {
   "title_suggest": "The Lord of the Rings",
   "edition_key": [
    "OL9177076M",
    "OL7883890M",
    "OL10681058M",
    "OL21217116M",
```

responseText

Deszerializálás

response

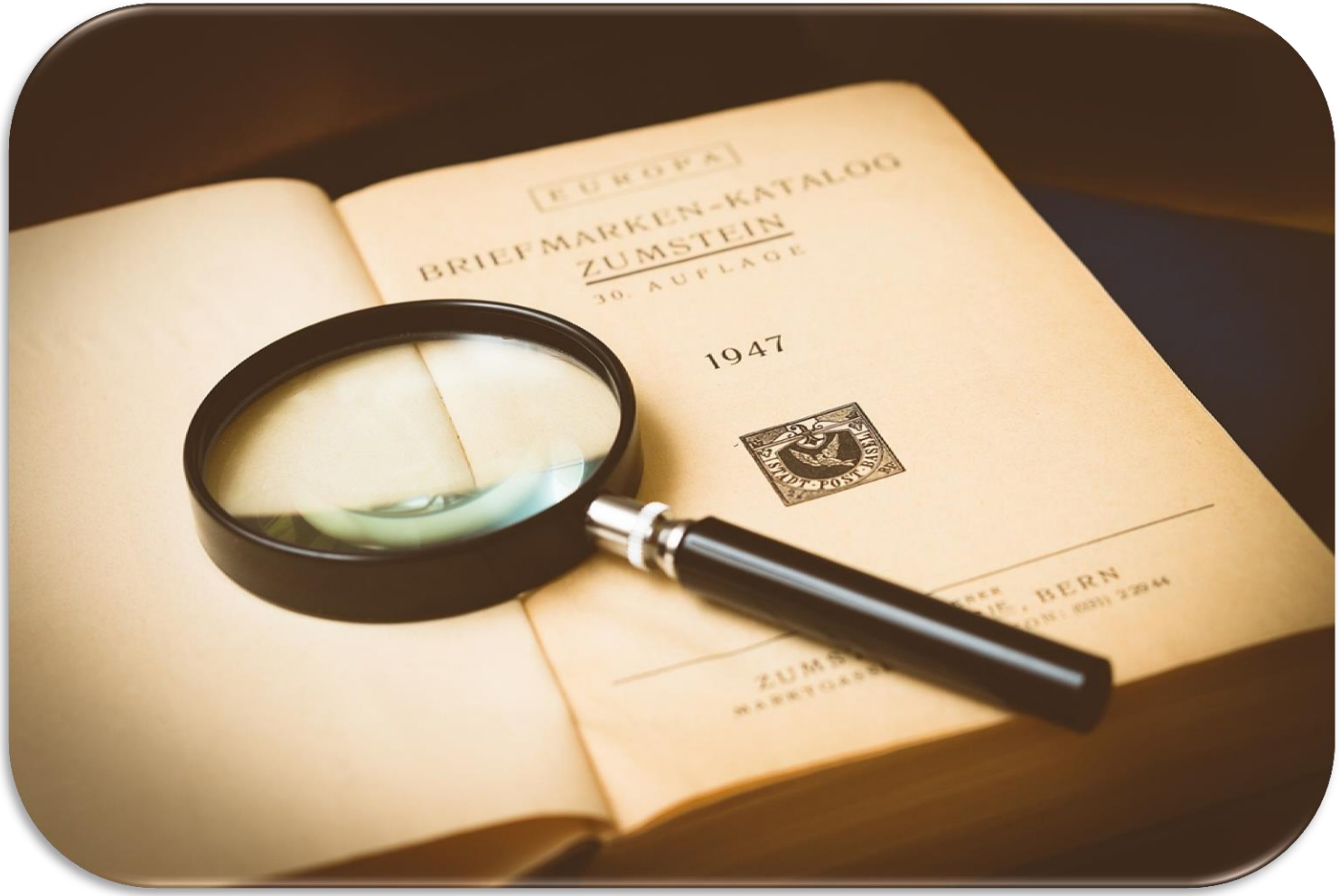



```
SearchResponse response = JsonConvert.DeserializeObject<SearchResponse>(responseText);
```

TOPMOTIVE®

DYSEO

Könyváruház – Demo 01



Könyváruház – Demo 01 – Következtetés

- Hogyan írtuk át a szinkron működést aszinkron működéssé, BackgroundWorker-t használva?
  - ***Callback***:
    - Egy olyan metódus (függvény) ami akkor hívódik meg, amikor egy aszinkron művelet befejeződött
  - Visszatérítési érték átalakul ***callback*** paraméterré
  - Mi is az a „***callback hell***”?

```
public void Search(string searchText, Action<List<Book>> callback)
{
    api.Search(searchText, (SearchResponse response) =>
    {
        SearchResponseMapper.Map(response, (List<Book> models) =>
        {
            callback(models);
        });
    });
}
```

Könyváruház – Demo 01

Könyváruház – Demo 01 – Következtetés

- Hogyan írjuk át a szinkron működést aszinkron működéssé, Task-ot használva?
  - A metódust async-nak jelöltük
  - Az async metódust await-el hívtuk meg
  - Ami nem támogatja a Task API-t Task.Run(...)-al futtattuk

Task – Egyszerűsített használati útmutató

```
public async Task<SearchResponse> SearchAsync(string searchText)
{
    string address = ConstructSearchUrl(searchText);

    HttpClient client = new HttpClient();

    string responseText = await client.GetStringAsync(address);
    SearchResponse response = await Task.Run(() =>
JsonConvert.DeserializeObject<SearchResponse>(responseText));

    return response;
}
```

Task – Vigyázat, nem mindegy...

- Await nélkül:

```
Debug.WriteLine("Before Task.Run");
Task.Run(() => Debug.WriteLine("Task.Run action"));
Debug.WriteLine("After Task.Run");
```

Before Task.Run
After Task.Run
Task.Run action

- Await-el:

```
Debug.WriteLine("Before Task.Run");
await Task.Run(() => Debug.WriteLine("Task.Run action"));
Debug.WriteLine("After Task.Run");
```

Before Task.Run
Task.Run action
After Task.Run

- Task.Run(...)
  - Számítási műveletekre ajánlott használni

Fontosabb Task metódusok és property-k

Használatosak

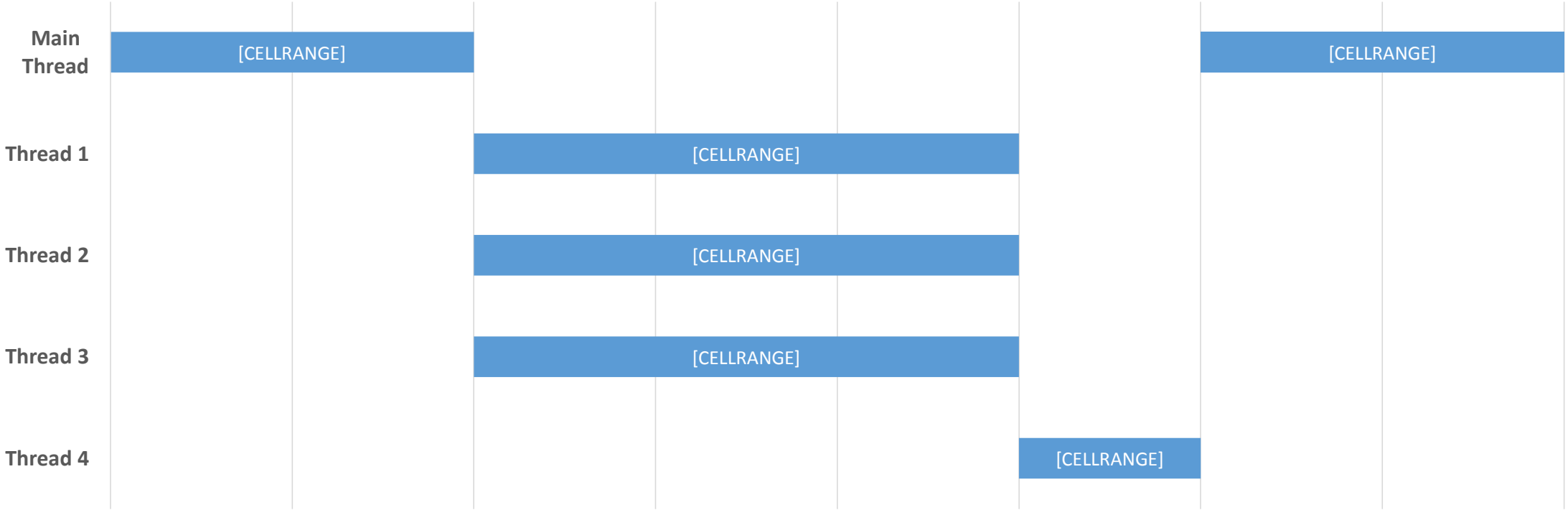
- ContinueWith()
- Exception
- IsCanceled
- IsCompleted
- IsFaulted

Jó tudni róluk

- Result
- Wait()
- Start()
- RunSynchronously()

Könyváruház applikáció

- Export műveletsorrendiség:



Könyváruház – Demo 02

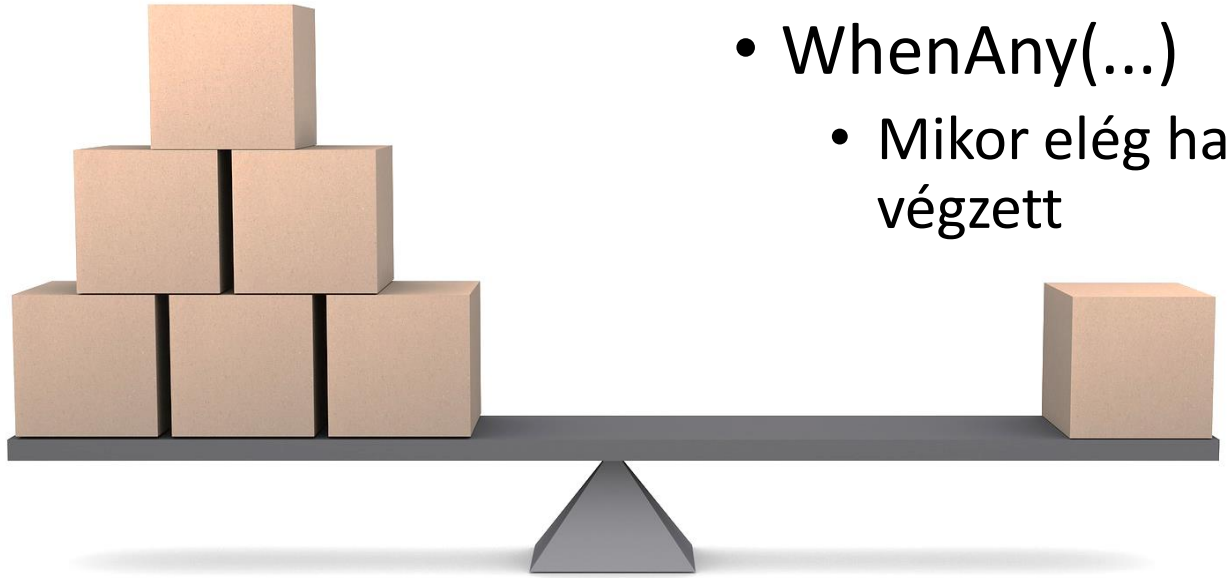
Könyváruház – Demo 02 – Következtetés

BackgroundWorker

- Számláló bevezetése, hogy tudjuk mindenki végzett

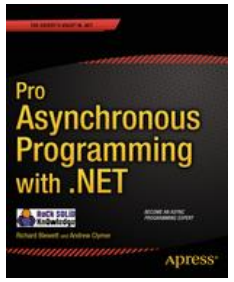
Task

- WhenAll(...) metódus használata
- WhenAny(...)
  - Mikor elég ha csak egy Task végzett



Egyéb információk

- CancellationTokenSource
  - https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-cancellation
- TaskCompletionSource
  - https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcompletionsource-1?view=netframework-4.7.2
- Könyv
  - https://www.apress.com/gp/book/9781430259206



Összefoglaló

- Szinkron / aszinkron metódus hívás
- Számítási művelet / Erőforrás hozzáférési művelet
- UI elemekhez csak UI szál férhet hozzá
- Szinkron forráskód átalakítása aszinkron forráskóddá
  - BackgroundWorker használat
  - Callback fogalma
  - Task használat
- Párhuzamos műveletek végrehajtása
- Task megszakítása

Kérdések...

Köszönöm a figyelmet!