

# Szoftverttechnikák önálló gyakorlat

## 4. önálló feladat

### Többszálú alkalmazások készítése

Thread, ThreadPool, szinkronizáció, ManualResetEvent,  
WaitHandle, Invoke

## Tartalom

TARTALOM .....	2
BEVEZETÉS .....	2
FELADAT 1 – BICIKLI .....	4
FELADAT 2 – RAJTVONAL .....	5
FELADAT 3 – PIHENŐ.....	6
FELADAT 4 – KILÓMÉTERÓRA.....	7
FELADAT 5 – ÚJRAKEZDÉS .....	8
EMLÉKEZTETŐ .....	9
OPCIONÁLIS FELADAT – 2 IMSC PONTÉRT .....	9

## Bevezetés




Az önálló feladat a konkurens/többszálú alkalmazások fejlesztése előadásokon elhangzottakra épít. A feladatok gyakorlati háttéréül a „4. gyakorlat – Többszálú alkalmazások fejlesztése” laborgyakorlat szolgál.

A fentiekre építve, jelen önálló gyakorlat feladatai a feladatléírást követő rövidebb iránymutatás segítségével elvégezhetők.

Az önálló gyakorlat a következő ismeretek elmélyítését célozza

- Windows Forms tervező használata
- Külső szálakból windows-os vezérlők manipulálása
- Közös erőforrás közös szálból történő elérése (lock használata)
- ManualResetEvent, AutoResetEvent

A feladat publikálásának és beadásának alapelvei megegyeznek az előző feladatával, pár kiemelt követelmény:

- A feladathoz tartozó GitHub Classroom hivatkozás: <https://classroom.github.com/a/mSDyVOTt>  
A munkamenet megegyezik az előző házi feladatával: a fenti hivatkozással mindenkinek születik egy privát repója, abban kell dolgozni és a határidőig a feladatot beadni.
-  Az elindulással **ne várd meg a határidő közeledtét**, legalább a saját repó létrehozásáig juss el mielőbb. Így ha bármi elakadás lenne, még időben tudunk segíteni.
-  A repository gyökérmappájában található **neptun.txt** fájlba írd bele a Neptun kódod, csupa nagybetűvel.
-  A beadott megoldások mellé külön indoklást, illetve leírást nem várunk el, ugyanakkor az elfogadás feltétele, hogy a beadott kódban a feladat megoldása szempontjából relevánsabb részek **kommentekkel legyenek ellátva**.

- A munka során a kiindulási repóban levő solutionben/projektben kell dolgozni, új projektet/solutiont ne hozz létre.
- A megoldást a tanszéki portálra **nem** kell feltölteni, de az eredményt itt fogjuk meg hirdetni a kapcsolódó számonkérés alatt.
- Amikor a házi feladatod beadottnak tekinted, célszerű ellenőrizni a GitHub webes felületén a repository-ban a fájlokra való rápillantással, hogy valóban minden változtatást push-oltál-e.

## Feladat 1 – Bicikli

### Feladat

Készítsünk egy Windows Forms alkalmazást. Az alkalmazás felületének bal oldalán egy gomb legyen (ez egy biciklit jelképez), az alkalmazás jobb oldalán egy kék színű panel (ez a célt jelképezi), továbbá legyen egy „start” feliratú gomb felület alján. A gomb megnyomásakor indítsunk egy új háttérzálat, mely a biciklit jelképező gombot 3 és 9 közötti (véletlenszerűen választott) lépésközönként átmozgatja a jobb oldalon található panelig!

### Megoldás

1. Hozzunk létre egy új Windows Forms alkalmazást
2. Adjunk az ablakhoz (ebben a sorrendben) egy panel és két gomb vezérlőt az alábbi tulajdonságokkal:

- a. Panel
  - i. Name: pTarget
  - ii. BackColor: „LightSteelBlue”
- b. Button
  - i. Name: bBike1
  - ii. Text: „b”
  - iii. Font/Name: Webdings
  - iv. Font/Size: 32
- c. Button
  - i. Name: bStart
  - ii. Text: Start

3. Rendezzük be a vezérlőket a következőképpen:



4. A bicikli mozgására definiáljuk az alábbi segédfüggvényeket

```
public void BikeThreadFunction(object param)
{
    Button bike = (Button)param;
    while (bike.Left < pTarget.Left)
    {
        MoveBike(bike);
        Thread.Sleep(100);
    }
}
```

```
delegate void BikeAction(Button bike);
```

```
Random random = new Random();

public void MoveBike(Button bike)
{
    if (InvokeRequired)
    {
        Invoke(new BikeAction(MoveBike), bike);
    }
    else
    {
        bike.Left += random.Next(3, 9);
    }
}
```

Emlékeztető: egy Windows Forms vezérlőhöz/űrlaphoz csak abból a szálból lehet hozzáférni, mely a vezérlőt létrehozta, ugyanis ezek nem szálbiztosak, és kivétel dobásával jelzik, ha mégis „rosszul” próbáljuk őket használni. A probléma elkerülésére az InvokeRequired/Invoke használata nyújt megoldást. A fenti példában azért volt szükség a BikeAction nevű delegate típus definiálására mert az Invoke függvény kötelezően egy általános delegate-et vár el paraméterként, ilyenkor a rövidített szintaktika – a függvénynév megadása delegate példányosítás nélkül – nem használható.

5. Iratkozzunk fel a Start gomb eseménykezelőjére (duplaklikk a Start gombra a designerben), majd teszteljük az alkalmazást.

```
private void bStart_Click(object sender, EventArgs e)
{
    StartBike(bBike1);
}

private void StartBike(Button bBike)
{
    Thread t = new Thread(BikeThreadFunction);
    bBike.Tag = t;
    t.IsBackground = true; // Ne blokkolja a szál a processz megszűnését
    t.Start(bBike);
}
```

A fenti felület tulajdonképpen célja, hogy szálak futását és szinkronizációját (Windows Forms űrlapok/vezérlők vonatkozásában) demonstrálja. A későbbi, immár önállóan megvalósítandó feladatokban további szálakat (és bicikliket) fogunk létrehozni, és a futásukat összehangolni.

## Feladat 2 – Rajtvonal

### Feladat

Valósítsuk meg a rajtvonalat. Egészítsük ki az alkalmazásunkat két további biciklivel, melyek mozgatásáért két további szál fog felelni, illetve egy új panellal (start panel) és egy gombbal (Step1) a következő elrendezésben.



A **Start** gomb megnyomását követően mindhárom bicikli induljon el véletlenszerű tempóban. Amikor egy bicikli a start panelre érkezik, az őt vezérlő szál blokkolva várakozzon. Amikor a **Step1** gombot megnyomjuk, a biciklik folytassák útjukat a célig.

### Megoldás

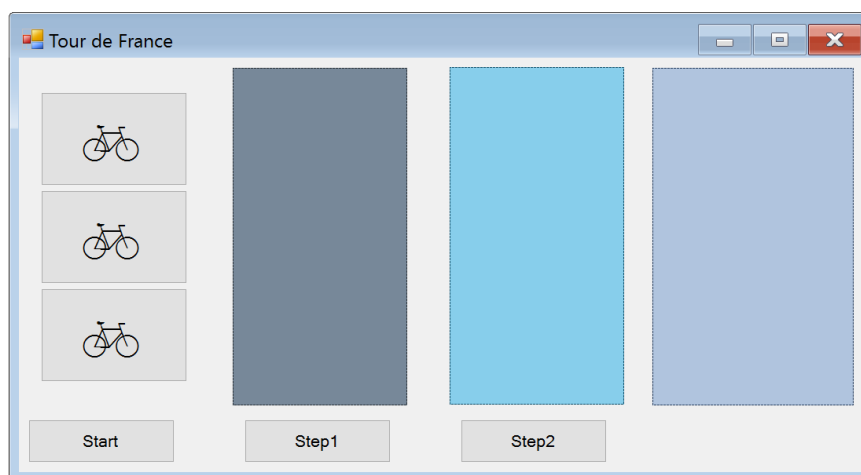
A feladat megoldásához a kapcsolódó gyakorlatban már alkalmazott, illetve az itt korábban megismert elemeket kell alkalmazni és kombinálni. A megoldás lépéseit csak nagy vonalakban adjuk meg, néhány kiegészítő segítséggel:

- Mivel a start-panelt a biciklinél később helyeztük a Form-ra, alapból kitakarja a fölé menő biciklit. Ezen úgy segíthetünk, hogy a tervező nézetben a panelon jobb egérgombbal kattintva kiadjuk a „Send to back” parancsot.
- Az egyszerűbb átláthatóság érdekében fontos, hogy az újabb vezérlőknek is mind beszédes neveket adjunk (pl.: `bBike2`, `bBike3`, `bStep1`)
- Mivel a várakozást követően a versenyzőknek egyszerre kell indulniuk, a várakozás és indítás megvalósítására egy `ManualResetEvent` objektumot célszerű használni.
- A feladat megoldása során gombonként egy szálfüggvényt használj, vagyis a **Step1** gomb megnyomásakor ne új szálakat indítsunk minden gombhoz, hanem oldjuk meg, hogy meglévő szálak várakozzanak, majd a gombnyomást követően folytassák futásukat

## Feladat 3 – Pihenő

### Feladat

Egészítsük ki az alkalmazásunkat egy további panellel (depo panel), mely egy pihenőt jelképez. A pihenőhelyre beérkezve a biciklik megállnak, majd egyesével tovább indulnak. A továbbindításért egy új gomb (**Step2**) felel, melynek minden gombnyomására egy-egy bicikli elindul. A pihenő alatt a bicikliket mozgató szálak blokkolva várakoznak.



### Megoldás

A feladat megoldása analóg az előzőével, ám ezúttal `AutoResetEvent`-et kell használni.

## Feladat 4 – Kilóméteróra

### Feladat

Egészítsük ki a Form-ot egy `long` típusú mezővel. Minden egyes bicikli, minden megtett lépés után növelje meg ezt a számlálót a lépés során megtett pixelek számával. A cél-panel alatt legyen egy gomb, melyet megnyomva a gomb szövege a számláló aktuális értékére változik. Ügyeljen a kölcsönös kizárásra, melyet `lock` utasítás segítségével valósítson meg.

### Megoldás

A megoldás menete:

- Készítsen egy új függvényt, amely a megtett utat számláló változót megnöveli a paraméterben kapott pixel számmal (`void increasePixels(long step)`). Ügyeljen arra, hogy ezt a függvényt bárhol, bármely számból lehessen hívni.
- Készítsen egy másik függvényt, amellyel biztonságosan kiolvasható az aktuális számláló értéke, bármilyen számból is hívják (`long getPixels()`).
- A bicikli mozgásakor hívja meg a lépést hozzáadó `increasePixels` függvényt.
- A célpanel alatt levő gomb kattintásakor kérdezze le az aktuális számláló értéket a `getPixels` függvénnyel, és írja ki a gombra az értéket.

**Lényeges:** *a megoldás csak akkor elfogadható, ha a lock utasítással a kölcsönös kizárás megvalósításra kerül (`increasePixels` és `getPixels` függvények).*

## Feladat 5 – Újrakezdés

### Feladat

Egészítsük ki az alkalmazásunkat úgy, hogy bármelyik biciklit újra tudjuk indítani. Az újra indításhoz elég a biciklit jelképező gombra kattintani. Ilyenkor a bicikli visszakerül a kiinduló pozícióba és újra kezdi a futamot. Az új futam során a bicikli Step1-nél és Step2-nél is ismét meg kell álljon.

### Megoldás

A következőkben megadjuk a feladat megoldásának néhány fontos elemét.

- A feladat megoldásához meg kell tudnunk szakítani az aktuálisan futó szálát, legalábbis `WaitSleepJoin` állapotban (`Thread.Interrupt` metódus). Ehhez minden egyes gombhoz tárolnunk kell az aktuálisan őt vezérlő szál objektumot. Ezt (hasonlóan a 3. gyakorlat 4. feladatához) megtehetjük a vezérlő `Tag` tulajdonságában. A `Tag` tulajdonság beállítására vagy a szál létrehozása után a `StartBike` műveletben, vagy a szálfüggvényben (az aktuális szál `Thread.CurrentThread`-del való lekérdezésével) kerítsünk sort. Példa az utóbbira:

```
public void BikeThreadFunction(object param)
{
    Button bike = (Button)param;
    bike.Tag = Thread.CurrentThread;
    ...
}
```

Ezt az információt aztán kiolvashatjuk a gombnyomás eseménykezelőjében:

```
private void bike_Click(object sender, EventArgs e)
{
    Button bike = (Button)sender;
    Thread thread = (Thread)bike.Tag;

    // Ha még nem indítottuk ez a szálát, ez null.
    if (thread == null)
        return;

    // Megszakítjuk a szál várakozását, ez az adott szálban egy
    // ThreadInterruptedException-t fog kiváltani
    // A függvény leírásáról részleteket az előadás anyagaiban találsz
    thread.Interrupt();

    // Megvárjuk, amíg a szál leáll
    thread.Join();
    ...
}
```

Érdekes észrevenni, hogy a gomb eseménykezelőjében a `sender` paraméterből kiolvasható, hogy konkrétan melyik gombtól származik az esemény. Ezt kihasználva nem szükséges mindhárom gombhoz külön eseménykezelő függvényt írunk, hanem használhatja mindhárom gomb ugyanazt a függvényt. Egy eseményhez a következőképpen tudunk Visual Studioban egy már létező függvényt hozzárendelni: a Properties ablak események oldalán ne duplán kattintsunk az eseményen, hanem kattintsunk egyszer az esemény során, majd



nyissuk le az esemény sorában a jobboldali oszlopban megjelenő legördülőmezőt, és válasszuk ki a listából a megfelelő függvényt.

- A `thread.Interrupt()` hívás a `BikeThreadFunction` függvényen belül egy `ThreadInterruptedException` kivételt fog kiváltani (amikor a szál `WaitSleepJoin` állapotba kerül, vagyis a `Sleep` és `WaitOne` művelethívások során). Fontos, hogy a kivételre fel legyünk készülve, vagyis a függvény teljes törzse try-catch blokkal számítson az ilyen típusú hibára. Például így:

```
try
{
    // Teljes függvénytörzs
    ...
}
catch (ThreadInterruptedException)
{
    // Lenyeljük, de szigorúan kizárólag a ThreadInterruptedException-t.
    // Ha nem kezeljük az Interrupt hatására a szállfüggvényünk
    // és az alkalmazásunk is csúnyán "elszállna".
}
```

A feladat további megoldása önállóan elvégezhető a korábbi ismeretek alapján.

## Emlékeztető

**Lényeges,** hogy a beadott megoldások mellé külön indoklást, illetve leírást nem várunk el, ugyanakkor kérjük, hogy a beadott kódban a feladat megoldása szempontjából relevánsabb részek **kommentekkel legyenek ellátva**. Ezen felül ne felejtse el beadáskor a **neptun.txt**-t is kitölteni.

## Opcionális feladat – 2 IMsc pontért

### Feladat

Tegyük lehetővé a biciklik megállítását. Tegyen ki egy új gombot a Start gomb alá Stop felirattal. A Stop gombra kattintás állítsa meg az összes biciklit, és állítsa le a bicikliket futtató szálakat is.

### Megoldás

A következőkben megadjuk a feladat megoldásának néhány fontos elemét.

- Vegye fel a Stop gombot a felületre és készítse elő a kattintást kezelő függvényt.
- A megállításhoz szükség lesz két jelzésre a bicikliket futtató szál felé. Ez egyik jelzés egy bool típusú változó, amelyet a bicikliket futtató szál ciklusa figyel. Vegye fel ezt a jelzést `stopBikes` néven, és módosítsa a szál függvényt, hogy ha a bool változó jelez, fejezze be a futást.
- A másik jelzés abban az esetben kell, ha a szálak várnak. Ilyenkor nem tudják a bool változót ellenőrizni. Vegyen fel egy új `ManualResetEvent` típusú változót, amely a leállítás esemény fogja jelezni. Ezt az eseményt a bool változóval együtt a biciklire való kattintás eseménykezelőjében állítsa.
- A bicikliket mozgató szál függvényben kommentezze ki (**ne törölje!**) az eddigi várakozást megvalósító kódrészeket, és készítsen egy új megoldást az előbb felvett újraindítást jelző `ManualResetEvent` segítségével. A várakozásokra továbbra is

szükség lesz, azonban várakozni nem csak a start vonalra illetve a pihenőre szükséges, hanem a leállítást *is* észre kell venni.

- Ha leállítás történt, a szál futását be kell fejezni. Ha a leállást jelző esemény megtörtént, térjen vissza a szál függvénye egy `return` utasítással.