



دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ)  
تمرین‌های سری سیزدهم (delegates, Threads and Events)

استاد: سید صالح اعتمادی

مهلت ارسال: ۱۳ خرداد ۱۴۰۰

## فهرست مطالب

۳	۱	مقدمه و آماده‌سازی
	۱.۱	نکات مورد توجه
	۱.۲	آماده‌سازی‌های اولیه
	۱.۲.۱	آماده‌سازی‌های مربوط به git
	۱.۲.۲	آماده‌سازی‌های مربوط به visual studio
۴	۲	پیاده‌سازی تمرین
	۱.۲	مجموعه تست‌های SingleFileWatcher
	۱.۱.۲	Register تست
	۲.۱.۲	Unregister تست
	۳.۱.۲	MultiRegisterUnregister تست
	۴.۱.۲	Dispose تست
	۲.۲	مجموعه تست‌های DirectoryWatcher
	۱.۲.۲	RegisterAddFile تست
	۲.۲.۲	RegisterDeleteFile تست
	۳.۲.۲	UnRegister تست
	۳.۲	مجموعه تست‌های SingleReminder
	۱.۲.۲	SingleReminderThread تست
	۲.۲.۲	SingleReminderThreadPool تست
	۳.۲.۲	SingleReminderTask تست
	۴.۲	مجموعه تست‌های ActionTools
	۱.۴.۲	CallSequential تست
	۲.۴.۲	CallParallel تست
	۳.۴.۲	CallParallelThreadSafe تست
	۴.۴.۲	CallSequentialAsync تست
	۵.۴.۲	CallParallelAsync تست
	۶.۴.۲	CallParallelThreadSafeAsync تست
۷	۳	ارسال تمرین
	۱.۳	مشاهده وضعیت اولیه فایل‌ها
	۲.۳	اضافه کردن فایل‌های تغییر یافته به stage
	۳.۳	کردن تغییرات انجام شده commit
	۴.۳	ارسال تغییرات انجام شده به Remote repository
	۵.۳	ساخت Pull Request
	۶.۳	ارسال Pull Request به بازبیننده

## ۱ مقدمه و آماده‌سازی

### ۱.۱ نکات مورد توجه

- توجه داشته باشید که برای کسب نمره‌ی قبولی درس کسب حداقل نصف نمره‌ی هر سری تمرین الزامی می‌باشد.
- مهلت ارسال پاسخ تمرین تا ساعت ۲۳:۵۹ روز اعلام شده است. توصیه می‌شود نوشتن تمرین را به روزهای پایانی موكول نکنید.
- همکاری و هم‌فکری شما در حل تمرین مانع ندارد، اما پاسخ ارسالی هر کس حتما باید توسط خود او نوشته شده باشد.
- مبنای درس، اعتماد بر پاسخ ارسالی از سوی شماست؛ بنابراین ارسال پاسخ در ریپیازیتوری گیت شما به این معناست که پاسخ آن تمرین، توسط شما نوشته شده است. در صورت تقلب یا اثبات عدم نوشتن پاسخ حتی یک سوال از تمرین، برای هر دو طرف تقلب‌گیرنده و تقلب‌دهنده نمره‌ی مردود برای درس در نظر گرفته خواهد شد.
- توجه داشته باشید که پاسخ‌ها و کدهای مربوط به هر مرحله را بایستی تا قبل از پایان زمان مربوط به آن مرحله، در سایت Azure DevOps (طبق توضیحات کارگاه‌ها و کلاس‌ها) فرستیید. درست کردن Pull request Complete و Pull request و انتقال به شاخه‌ی master پس از تکمیل تمرین فراموش نشود!
- پس از پایان مهلت ارسال تا ۲ روز به ازای هر روز تأخیر ۱۰ درصد از نمره مربوط به تمرین کسر خواهد شد و پس از ۲ روز نمره‌ای به تمرین تعلق نخواهد گرفت.
- برای طرح سوال و پرسش و پاسخ از صفحه درس در Quera استفاده کنید.

### ۲.۱ آماده‌سازی‌های اولیه

قواعد نام‌گذاری تمرین را از جدول ۱ مطالعه کنید.

جدول ۱ : قراردادهای نام‌گذاری تمرین

Naming conventions					
Branch	Directory	Solution	Project	Test Project	Pull Request
fb_A4	A4	A4	A4	A4.Tests	A4

#### ۱.۲.۱ آماده‌سازی‌های مربوط به git

✓ ابتدا به شاخه‌ی master بروید.

```
1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
2 $ git checkout master
3 Switched to branch 'master'
4 Your branch is up to date with 'origin/master'.
```

✓ تغییرات انجام شده در Remote Repository را دریافت کنید.

```
1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (master)
2 $ git pull
3 remote: Azure Repos
4 remote: Found 8 objects to send. (90 ms)
5 Unpacking objects: 100% (8/8), done.
6 From https://9952XXXX.visualstudio.com/AP99002/_git/AP99002
7   e7fd3b5..2cc74de  master      -> origin/master
8 Checking out files: 100% (266/266), done.
9 Updating e7fd3b5..2cc74de
10 Fast-forward
11   .gitattributes          |    63 +
12   A4/A4.sln                |    37 +
13   A4/A4/A4.csproj          |    61 +
14   A4/A4/Program.cs          |    15 +
15
16
17
```

✓ یک شاخه‌ی جدید با نام `fb_A4` بسازید و تغییر شاخه دهید.

```

1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (master)
2 $ git checkout -b fb_A4
3 Switched to a new branch 'fb_A4'
4 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
5 $
```

توصیه می‌شود پس از پیاده‌سازی هر کلاس تغییرات انجام شده را `commit` و `push` کنید.

## ۲.۲.۱ آماده‌سازی‌های مربوط به visual studio

یک پروژه‌ی جدید طبق قراردادهای نام‌گذاری موجود در جدول ۱ در ریشه‌ی ریپازیتوری git خود بسازید.  
ساختار فایل پایه‌ای که در اختیار شما قرار می‌گیرد به صورت زیر است:

```

1 A4
2   ---Project
3   |   ActionTools.cs
4   |   DirectoryWatcher.cs
5   |   ISingleReminder.cs
6   |   Program.cs
7   |   SingleFileWatcher.cs
8   |   SingleReminderTask.cs
9   |   SingleReminderThread.cs
10  |   SingleReminderThreadPool.cs
11
12 \---ProjectTests
13     ActionToolsTests.cs
14     DirectoryWatcherTests.cs
15     SingleFileWatcherTests.cs
16     SingleReminderTests.cs
```

در فایل پایه دو پوشه وجود دارد شما باید فایل‌های (A4) موجود در پوشه‌ی Project را به پروژه‌ی اصلی (A4) و فایل‌های (A4.Tests) موجود در پوشه‌ی ProjectTests را به پروژه‌ی تست (A4.Tests) اضافه کنید.

## ۲ پیاده‌سازی تمرین

### ۱.۲ مجموعه تست‌های `SingleFileWatcher`

هدف این تمرین آشنایی با `event` و `delegate` و طرز استفاده از آنها می‌باشد. در این کلاس لازم است از `Changed` به نام `event` و `Changed` به نام `delegate` استفاده کنید. در کلاس `System.IO.FileSystemWatcher` استفاده کنید.

#### ۱.۱.۲ تست `Register`

برای پاس شدن این تست لازم است که کلاس `SingleFileWatcher` را به گونه‌ای پیاده‌سازی کنید که علاوه بر پیاده‌سازی واسط `IDisposable` را به گونه‌ای پیاده‌سازی کنید که هنگام تغییر فایلی که به سازنده پاس می‌شود داده شده به مت `Register` صدا زده شود. علت پیاده‌سازی واسط `Disposable` در این کلاس عضویت شی‌ای از نوع `FileSystemWatcher` است که خود این واسط را پیاده‌سازی می‌کند. لذا لازم است که استفاده کننده از این شی بداند که وقتی کارش با این شی تمام شد باید مت `Dispose` را صدا بزند. برای جزئیات بیشتر در رابطه با نیازمندی‌های پیاده‌سازی مت تست را مطالعه کنید. <sup>۱۵/۱۴</sup>

#### ۱.۱.۲ تست `Unregister`

برای پاس شدن این تست لازم است مت `Unregister` را بدرستی پیاده‌سازی کنید. بطوريکه بعد از صدا زدن این مت برای یک هنگام تغییر فایل دیگر صدا زده نشود. برای جزئیات بیشتر پیاده‌سازی مت تست را مطالعه کنید. <sup>۱۴/۱۳</sup>

**۳.۱.۲ تست MultiRegisterUnregister**

پیاده‌سازی شما از سازنده و متدهای Register و Unregister باید بگونه‌ای باشد که بیش از یک **delegate** بتوانند در آن واحد Register شده و در صورت نیاز بعداً Unregister شوند. مطالعه این تست و اطمینان از پاس شدن آن به درک مفهوم **Multicast delegate** کمک می‌کند.

**۴.۱.۲ تست Dispose**

این تست برای اطمینان از پیاده‌سازی واسط **IDisposable** طراحی شده. در صورت پیاده‌سازی این واسط این تست کامپایل شده و پاس می‌شود.

**۲.۲ مجموعه تست‌های DirectoryWatcher**

پیاده‌سازی کلاس **DirectoryWatcher** مشابه کلاس **SingleFileWatcher** می‌باشد. با این تفاوت که علاوه بر پایش پوشه بجای فایل منتظر دو نوع تغییر ایجاد و حذف فایل در پوشه بوده و بعد از اطلاع از این تغییر آن را به **delegate** هایی که برای آن تغییر کرده باشد اطلاع می‌دهیم. برای پیاده‌سازی این کلاس لازم است از **event** **Created** و **Deleted** در کلاس **Register** استفاده کنید.

**۱.۲.۲ تست RegisterAddFile**

برای پاس شدن این تست که علاوه بر سازنده کلاس **DirectoryWatcher** متد **Register** داشته باشد بگونه‌ای پیاده‌سازی شود که در صورت ایجاد یک فایل در پوشه پاس شده به سازنده **Deletegate** پاس شده به **Register** برای نوع تغییر ایجاد فایل صدا زده شود. برای جزئیات بیشتر تست را مطالعه کنید.

**۲.۲.۲ تست RegisterDeleteFile**

برای پاس شدن این تست علاوه بر صدا زدن **delegate** مربوطه هنگام ایجاد فایل، در صورت حذف فایلی از پوشه پاس شده به سازنده، **delegate** مربوطه را صدا بزنید. برای جزئیات بیشتر تست را مطالعه کنید.

**۳.۲.۲ تست UnRegister**

در صورت پیاده‌سازی صحیح متد **UnRegister** این تست پاس خواهد شد. برای جزئیات بیشتر تست را مطالعه کنید.

**۳.۲ مجموعه تست‌های SingleReminder**

تا اینجا با استفاده از یک **event** که توسط کلاس **FileSystemWatcher** پیاده‌سازی شده بود آشنا شدیم. حال نوبت آن است که شما یک **event** پیاده‌سازی کنید. برای این کار یک **ISingleReminder** واسط به نام **event** در نظر گرفته‌ایم که شما آن را به سه روش پیاده‌سازی می‌کنید. ابتدا با استفاده از یک **Thread** ساده. سپس با استفاده از **ThreadPool** و نهایتاً با استفاده از **Task**. همه بیاده‌سازی‌ها یک کار را انجام می‌دهند ولی به روش‌های متفاوت. هدف نهایی این است که سازنده هر کدام از این کلاس‌های یک پیام و مدت زمان در سازنده دریافت کنند. سپس با ارائه یک **Reminder** به نام **event** **Register** کردن را فراهم کنند. بعد از صدا زدن متد **Start** تمام کسانی که با **event** این کلاس **Register** کرده‌اند، بعد از زمان مشخص شده، پیام معین را دریافت می‌کنند.

**۱.۳.۲ تست SingleReminderThread**

برای پاس شدن این تست لازم است که کلاس **SingleReminderThread** را طبق توضیح بالا پیاده‌سازی کنید. در این قسمت لازم است هنگام پیاده‌سازی از کلاس **System.Threading.Thread** استفاده کنید. در صورت عدم استفاده مناسب از این کلاس نمره این تمرین صفر لحاظ خواهد شد.

**۲.۳.۲ تست SingleReminderThreadPool**

برای پاس شدن این تست لازم است که کلاس **SingleReminderThreadPool** را طبق توضیح بالا پیاده‌سازی کنید. در این قسمت لازم است هنگام پیاده‌سازی از کلاس **System.Threading.ThreadPool** استفاده کنید. در صورت عدم استفاده مناسب از این کلاس نمره این تمرین صفر لحاظ خواهد شد.

**۳.۳.۲ تست SingleReminderTask**

برای پاس شدن این تست لازم است که کلاس `SingleReminderTask` را طبق توضیح بالا پیاده‌سازی کنید. در این قسمت لازم است هنگام پیاده‌سازی از کلاس `System.Threading.Tasks.Task` استفاده کنید. در صورت عدم استفاده مناسب از این کلاس نمره این تمرین صفر لحاظ خواهد شد. ۶/۱۰

**۴.۲ مجموعه تست‌های ActionTools**

هدف این بخش از تمرین‌ها آشنایی بیشتر شما با کلاس `System.Threading.Tasks.Task` و پیاده‌سازی متدهای `async` و استفاده از کلمه کلیدی `await` می‌باشد. علاوه بر این فهم مساله `Race Condition` و چگونگی حل آن با استفاده از `lock`. برای این قسمت از تمرین لازم است کلاس استاتیک `ActionTools` را تعریف کرده و متدهای منتظر با تست‌ها را پیاده‌سازی کنید.

**۱.۴.۲ تست CallSequential**

در کلاس `ActionTools` متد `CallSequential` را به گونه‌ای پیاده‌سازی کنید که تعدادی `delegate` از نوع `[params Action[] delegate]` به عنوان پارامتر دریافت کند و این ها `delegate` را یکی پس از دیگری صدا بزن و پس از اتمام همگی، پایان بپذیرد. لازم است مقدار برگشتی این متد زمان اجرای آن به میلی ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید. ۵/۱۱

**۲.۴.۲ تست CallParallel**

در کلاس `ActionTools` متد `CallParallel` را به گونه‌ای پیاده‌سازی کنید که تعدادی `delegate` از نوع `[params Action[] delegate]` به عنوان پارامتر دریافت کند و این ها `delegate` را به صورت همزمان با استفاده از کلاس `Task` صدا زده و پس از اتمام همگی، پایان بپذیرد. لازم است مقدار برگشتی این متد، مدت زمان اجرای آن به میلی ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید. ۴/۱۲

**۳.۴.۲ تست CallParallelThreadSafe**

در کلاس `ActionTools` متد `CallParallelThreadSafe` را به گونه‌ای پیاده‌سازی کنید که تعدادی `delegate` از نوع `[params Action[] delegate]` و یک عدد تکرار به عنوان پارامتر دریافت کند و این ها `delegate` را به صورت همزمان و به تعداد تکرار با استفاده از کلاس `Task` صدا ده و پس از اتمام همگی پایان بپذیرد. در این پیاده‌سازی با استفاده از `lock` لازم است اطمینان حاصل کنید که با شروع همه `delegate` ها بصورت همزمان و اجرا به تعداد تکرار مشخص شده، ولی هیچکدام از `delegate` ها بصورت همزمان اجرا نشوند. مثلاً اول برای بار سوم اجرا شود بعد `delegate` دوم برای بار پنجم و به همین ترتیب. ولی هر دو `delegate` در آن واحد در حال اجرا نباشند. لازم است مقدار برگشتی این متد، مدت زمان اجرای آن به میلی ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید. ۳/۱۳

**۴.۴.۲ تست CallSequentialAsync**

در کلاس `ActionTools` متد `CallSequentialAsync` را شبیه متد `CallSequential` پیاده‌سازی کنید، با این تفاوت که لازم است این متد بصورت `async` پیاده‌سازی شود که بلافصله مقداری از نوع `Task long` برگرداند و اتمام پذیرد. در پیاده‌سازی این متد لازم است از کلمه کلیدی `await` استفاده کنید. لازم است مقدار برگشتی این متد، مدت زمان اجرای آن به میلی ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید. برای جزئیات بیشتر متد تست را مطالعه کنید. ۲/۱۴

**۵.۴.۲ تست CallParallelAsync**

در کلاس `ActionTools` متد `CallParallelAsync` را شبیه متد `CallParallel` پیاده‌سازی کنید، با این تفاوت که لازم است این متد بصورت `async` پیاده‌سازی شود که بلافصله مقداری از نوع `Task long` برگرداند و اتمام پذیرد. در پیاده‌سازی این متد لازم است از کلمه کلیدی `await` استفاده کنید. لازم است مقدار برگشتی این متد، مدت زمان اجرای آن به میلی ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید. برای جزئیات بیشتر متد تست را مطالعه کنید. ۱/۱۵

## ۶.۴.۲ تست CallParallelThreadSafeAsync

در کلاس `ActionTools` متد `CallParallelThreadSafe` پیاده‌سازی کنید، با این تفاوت که لازم است این متد بصورت `async` پیاده‌سازی شود که بلافاصله مقداری از نوع `Task<long>` برگرداند و اتمام پذیرد. در پیاده‌سازی این متد لازم است از کلمه کلیدی `await` استفاده کنید. لازم است مقدار برگشته این متد، مدت زمان اجرای آن به میلی‌ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید.<sup>۱۶</sup>

## ۳ ارسال تمرین

در اینجا یکبار دیگر ارسال تمرینات را با هم مرور می‌کنیم:

### ۱.۳ مشاهدهٔ وضعیت اولیهٔ فایل‌ها

ابتدا وضعیت فعلی فایل‌ها را مشاهده کنید:

```

1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
2 $ git status
3 On branch fb_A4
4 Untracked files:
5   (use "git add <file>..." to include in what will be committed)
6
7 A4/
8
9 nothing added to commit but untracked files present (use "git add" to track)
```

همان‌طور که مشاهده می‌کنید فolder `A4` و تمام فایل‌ها و فولدرهای درون آن در وضعیت `Untracked` قرار دارند و همان‌طور که در خط آخر خروجی توضیح داده شده برای `commit` کردن آن‌ها ابتدا باید آن‌ها را با دستور `git add` وارد `stage` کنیم.

### ۲.۳ اضافه کردن فایل‌های تغییر یافته به stage

حال باید فایل‌ها و فولدرهایی را که در `stage` قرار ندارند را وارد `stage` کنیم. برای این کار از دستور `git add` استفاده می‌کنیم.

```

1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
2 $ git add A4/*
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```

1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
2 $ git status
3 On branch fb_A4
4 Changes to be committed:
5   (use "git reset HEAD <file>..." to unstage)
6
7     new file:  A4/A4.sln
8     new file:  A4/A4/A4.csproj
9     new file:  A4/A4/App.config
10    new file:  A4/A4/Program.cs
11    new file:  A4/A4.Tests/A4.Tests.csproj
12
13
14
```

همان‌طور که مشاهده می‌کنید فolder `A4` و تمام فولدرها و فایل‌های درون آن (به جز فایل‌هایی که در `gitignore` معین کرده‌ایم) وارد `stage` شده‌اند.

### ۳.۳ commit کردن تغییرات انجام شده

در گام بعدی باید تغییرات انجام شده را `commit` کنیم. فراموش نکنید که فقط فایل‌هایی را می‌توان `commit` کرد که در `stage` قرار داشته باشد. با انتخاب یک پیام مناسب تغییرات صورت گرفته را `commit` می‌کنیم:

```

1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
2 $ git commit -m "Implement A4"
3 [fb_A4 c1f21df] Implement A4
4 15 files changed, 595 insertions(+)
5 create mode 100644 A4/A4.sln
6 create mode 100644 A4/A4/A4.csproj
7 create mode 100644 A4/A4/Program.cs
8 create mode 100644 A4/A4.Tests/A4.Tests.csproj
9 create mode 100644 A4/A4.Tests/packages.config
10 .
11 .
12 .

```

### ۴.۳ ارسال تغییرات انجام شده به Remote repository

کام بعدی ارسال تغییرات انجام شده به Remote Repository است.

```

1 sauleh@SE-X1Y MINGW64 /c/git/AP99002 (fb_A4)
2 $ git push origin fb_A4
3 Enumerating objects: 25, done.
4 Counting objects: 100% (25/25), done.
5 Delta compression using up to 8 threads
6 Compressing objects: 100% (22/22), done.
7 Writing objects: 100% (25/25), 9.56 KiB | 890.00 KiB/s, done.
8 Total 25 (delta 4), reused 0 (delta 0)
9 remote: Analyzing objects... (25/25) (5 ms)
10 remote: Storing packfile... done (197 ms)
11 remote: Storing index... done (84 ms)
12 To https://9952XXXX.visualstudio.com/AP99002/_git/AP99002
13 * [new branch] fb_A4 -> fb_A4

```

### ۵.۳ ساخت Pull Request

با مراجعه به سایت Azure DevOps یک Pull Request جدید با نام A4 بسازید به طوری که امكان merge کردن شاخهی fb\_A4 را بر روی شاخهی master را بررسی کند. (این کار در صورتی انجام می‌شود که کد شما کامپایل شود و همچنین تست‌های آن پاس شوند) در نهایت با انتخاب گزینهی set auto complete در صفحهی Pull Request مربوطه تعیین کنید که در صورت وجود شرایط merge این کار انجام شود. دقت کنید که گزینهی Delete source branch نباید انتخاب شود.

### ۶.۳ ارسال Pull Request به بازبیننده

در نهایت Pull Request ساخته شده را برای بازبینی، با بازبینندهی خود به اشتراک بگذارید.