دانشکده‌ی مهندسی کامپیوتر

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

علی حیدری
استاد: سید صالح اعتمادی

مهلت ارسال: ۲۱ خرداد ۱۳۹۸

فهرست مطالب

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 اعلام‌شده است. توصیه می‌شود نوشتن تمرین را به روزهای پایانی موکول نکنید.
- هم‌کاری و هم‌فکری شما در حل تمرین مانعی 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 ریپازیتوری گیت شما به این معناست که پاسخ آن تمرین، توسط شما نوشته شده است. در صورت تقلب یا اثبات عدم نوشتار 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) بفرستید. درست کردن `Pull request` و `Complete` کردن `Pull request` و انتقال به شاخه‌ی `master` پس از تکمیل تمرین فراموش نشود!
- پس ازپایان مهلت ارسال تا ۲ روز به ازای هر روز تاخیر 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_A13`	`A13`	`A13`	`A13`	`A13Tests`	`HW13`

۱.۲.۱ آماده‌سازی‌های مربوط به git

✓ ابتدا به شاخه‌ی `master` بروید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A11)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

✓ تغییرات انجام‌شده در `Remote Repository` را دریافت کنید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
$ git pull
remote: Azure Repos
remote: Found 8 objects to send. (90 ms)
Unpacking objects: 100% (8/8), done.
From https://9752XXXX.visualstudio.com/AP97982/_git/AP97982
   e7fd3b5..2cc74de  master                -> origin/master
Checking out files: 100% (266/266), done.
Updating e7fd3b5..2cc74de
Fast-forward
 .gitattributes                                |    63 +
 A13/A13.sln                                   |    37 +
 A13/A13/A13.csproj                            |    61 +
 A13/A13/App.config                            |     6 +
 A13/A13/Program.cs                            |    15 +
 A13/A13/Properties/AssemblyInfo.cs            |    36 +
 .
 .
```

```
.
```

✓ یک شاخه‌ی جدید با نام `fb_A13` بسازید و تغییر شاخه دهید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
$ git checkout -b fb_A13
Switched to a new branch 'fb_A13'
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A13)
$
```

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:

```
A13
+---Project
|       ActionTools.cs
|       DirectoryWatcher.cs
|       ISingleReminder.cs
|       Program.cs
|       SingleFileWatcher.cs
|       SingleReminderTask.cs
|       SingleReminderThread.cs
|       SingleReminderThreadPool.cs
|
\---ProjectTests
        ActionToolsTests.cs
        DirectoryWatcherTests.cs
        SingleFileWatcherTests.cs
        SingleReminderTests.cs
```

در فایل پایه دو پوشه وجود دارد شما باید فایل(های) موجود در پوشه‌ی Project را به پروژه‌ی اصلی (A13) و فایل(های) موجود در پوشه‌ی ProjectTests را به پروژه‌ی تست (A13Tests) اضافه کنید.

۲ پیاده‌سازی تمرین

۱.۲ مجموعه تست‌های `SingleFileWatcher`

هدف این تمرین آشنایی با `delegate` و `Event` و طرز استفاده از آنها می‌باشد. در این کلاس لازم است از `event` به نام `Changed` 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` سازنده و متد `Register` را به گونه‌ای پیاده‌سازی‌کنید که هنگام تغییر فایلی که به سازنده پاس می‌شود `delegate` داده‌شده به متد `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 شئ بداند که وقتی کارش با این شئ تمام شد باید متد `Dispose` را صدا بزند. برای جزئیات بیشتر در رابطه با نیازمندی‌های پیاده‌سازی متد تست را مطالعه کنید. ۱۵/۱

۲.۱.۲ تست `Unregister`

برای پاس شدن این تست لازم است متد `Unregister` را بدرستی پیاده‌سازی کنید. بطوریکه بعد از صدا زدن این متد برای یک `delegate` هنگام تغییر فایل دیگر صدا زده نشود. برای جزئیات بیشتر پیاده‌سازی متد تست را مطالعه کنید. ۱۴/۲

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 بعدا Unregister شوند. مطالعه این تست و اطمینان از پاس شدن آن به درک مفهوم Multicast delegate کمک می‌کند. ۱۳/۳

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. ۱۲/۴

۲.۲ مجموعه تست‌های DirectoryWatcher

پیاده‌سازی کلاس DirectoryWatcher مشابه کلاس SingleFileWatcher می باشد. 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 Register کرده باشند اطلاع می دهیم. برای پیاده‌سازی این کلاس لازم است از هایevent Created و Deleted در کلاس System.IO.FileSystemWatcher استفاده کنید.

۱.۲.۲ تست RegisterAddFile

برای پاس شدن این تست لازم است که علاوه بر سازنده کلاس DirectoryWatcher متد Register بگونه‌ای پیاده‌سازی شود که در صورت ایجاد یک فایل در پوشه پاس شده به سازنده، Deletgate پاس شده به Register برای نوع تغییر ایجاد فایل صدا زده شود. برای جزئیات بیشتر تست را مطالعه کنید. ۱۱/۵

۲.۲.۲ تست RegisterDeleteFile

برای پاس شدن این تست لازم است 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. ۱۰/۶

۳.۲.۲ تست UnRegister

درصورت پیاده‌سازی صحیح متد UnRegister این تست پاس خواهد شد. برای جزئیات بیشتر تست را مطالعه کنید. ۹/۷

۳.۲ مجموعه تست‌های SingleReminder

تا اینجا با استفاده از یک event که توسط کلاس FileSystemWatcher پیاده‌سازی شده بود آشنا شدید. حال نوبت آن است که شما یک event پیاده‌سازی کنید. برای این کار یک واسط به نام ISingleReminder 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 نهایتا با استفاده از Task. همه پیاده‌سازی‌ها 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 event به نام Reminder امکان 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. ۸/۸

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. ۷/۹

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 متناظر 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[]` به عنوان پارامتر دریافت کند و این delegateها را یکی پس از دیگری صدا بزن و پس از اتمام همگی، پایان بپذیرد. لازم است مقدار برگشتی این متد، مدت زمان اجرای آن به میلی‌ثانیه 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[]` به عنوان پارامتر دریافت کند و این delegateها 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 میلی‌ثانیه 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[]` و یک عدد تکرار به عنوان پارامتر دریافت کند و این delegateها 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. مثلا `delegate` 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 میلی‌ثانیه 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 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 برگشتی این متد، مدت زمان اجرای آن به میلی‌ثانیه 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 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 برگشتی این متد، مدت زمان اجرای آن به میلی‌ثانیه 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 `CallParallelThreadSafeAsync` را شبیه 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 برگشتی این متد، مدت زمان اجرای آن به میلی‌ثانیه باشد. برای محاسبه زمان اجرای متد می‌توانید از کلاس `Stopwatch` استفاده کنید. برای جزئیات بیشتر متد تست را مطالعه کنید. ۰/۱۶

۳ ارسال تمرین

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

۱.۳ مشاهده‌ی وضعیت اولیه‌ی فایل‌ها

ابتدا وضعیت فعلی فایل‌ها را مشاهده کنید:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A13)
$ git status
On branch fb_A13
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    A13/

nothing added to commit but untracked files present (use "git add" to track)
```

همان‌طور که مشاهده می‌کنید فولدر `A13` 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.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A13)
$ git add A13/*
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A13)
$ git status
On branch fb_A13
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   A13/A13.sln
    new file:   A13/A13/A13.csproj
    new file:   A13/A13/App.config
    new file:   A13/A13/Program.cs
    new file:   A13/A13/Properties/AssemblyInfo.cs
    new file:   A13/A13Tests/A13Tests.csproj
    new file:   A13/A13Tests/Properties/AssemblyInfo.cs
    new file:   A13/A13Tests/packages.config
    .
    .
    .
```

همانطور که مشاهده می‌کنید فولدر `A13` 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 باشند. با انتخاب یک پیام مناسب تغییرات صورت گرفته را `commit` می‌کنیم:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A13)
$ git commit -m "Implement HW13"
[fb_A13 c1f21df] Implement HW13
 15 files changed, 595 insertions(+)
 create mode 100644 A13/A13.sln
 create mode 100644 A13/A13/A13.csproj
 create mode 100644 A13/A13/App.config
 create mode 100644 A13/A13/Program.cs
 create mode 100644 A13/A13/Properties/AssemblyInfo.cs
 create mode 100644 A13/A13Tests/A13Tests.csproj
 create mode 100644 A13/A13Tests/Properties/AssemblyInfo.cs
 create mode 100644 A13/A13Tests/packages.config
.
.
.
```

۴.۳ ارسال تغییرات انجام شده به Remote repository

گام بعدی ارسال تغییرات انجام شده به `Remote Repository` است.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A13)
$ git push origin fb_A13
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 8 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (25/25), 9.56 KiB | 890.00 KiB/s, done.
Total 25 (delta 4), reused 0 (delta 0)
remote: Analyzing objects... (25/25) (5 ms)
remote: Storing packfile... done (197 ms)
remote: Storing index... done (84 ms)
To https://9752XXXX.visualstudio.com/AP97982/_git/AP97982
 * [new branch]      fb_A13 -> fb_A13
```

۵.۳ ساخت Pull Request

با مراجعه به سایت Azure DevOps یک Pull Request جدید با نام `HW13` بسازید به طوری که امکان `merge` کردن شاخه‌ی `fb_A13` را بر روی شاخه‌ی `master` را بررسی کند. (این کار درصورتی 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.