دانشکده مهندسی کامپیوتر

برنامه‌سازی پیشرفته
تمرین های سری هشتم *

مدرس: سید صالح اعتمادی
مستند: امید میرزاجانی

مهلت ارسال:
شنبه ۲۷ اردیبهشت ۹۹

فهرست مطالب

*با تشکر ویژه از آقای علی حیدری که این تمرین را در ترم دوم سال تحصیلی ۹۷-۹۸ تهیه کردند.

۱ مقدمه

۱.۱ موارد 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 نهایی 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 از پایان 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.
- بعضی از قسمت های تمرین نیاز به پیاده سازی بر روی هر چهار زبان `C#` ، `Python` ، `C++` و `Java` را دارند بعضی هم خیر. بنابراین روبروی هر سوال زبان های مورد نیاز برای پیاده سازی مشخص شده است.

۲ آماده سازی های اولیه

۱.۲ ساخت پروژه ی C#

برای ایجاد پروژه C# کافی است کد زیر را در ترمینال خود اجرا کنید:

```
mkdir A8_cs
cd A8_cs
dotnet new sln
mkdir A8_cs
cd A8_cs
dotnet new console
cd ..
dotnet sln add A8_cs\A8_cs.csproj
mkdir A8_cs.Tests
cd A8_cs.Tests
dotnet new mstest
dotnet add reference ..\A8_cs\A8_cs.csproj
cd ..
dotnet sln add A8_cs.Tests\A8_cs.Tests.csproj
```

۲.۲ قواعد نام گذاری

قواعد نام‌گذاری تمرین را از جدول ۱ مطالعه کنید.

جدول ۱: قراردادهای نام‌گذاری تمرین

Naming conventions		
Branch	Directory	Pull Request
`fb_A8`	A8	A8

* در کل یک دیرکتوری داخل Assignments به نام A۸ بسازید و داخل آن،یک دیرکتوری به نام A۸_cs داشته باشید و فایل های مربوطه را داخل دیرکتوری مربوطه بگذارید.

۳.۲ 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.

۱.۳.۲ 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.

۲.۳.۲ 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.

۳.۳.۲ 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.

۴.۳.۲ تست `Dispose`

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

۴.۲ 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.

۱.۴.۲ 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.

۲.۴.۲ 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.

۳.۴.۲ تست UnRegister

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

۵.۲ 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.

۱.۵.۲ 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.

۲.۵.۲ 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

۳.۵.۲ 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.

۶.۲ 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.

۱.۶.۲ 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 هاdelegate 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.

۲.۶.۲ 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 هاdelegate 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.

۳.۶.۲ 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 هاdelegate 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.

۴.۶.۲ 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.

۵.۶.۲ 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.

۶.۶.۲ 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.

موفق باشید.