دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ)
تمرین‌های سری دوم - Exception Handling

مدرس: سید صالح اعتمادی *

مهلت ارسال: ۲۱ اسفند ۱۴۰۰

فهرست مطالب

*با تشکر از آقای علی حیدری که نسخه ابتدایی این مستند را در بهار ۹۸ تهیه کردند.

۱ مقدمه و آماده‌سازی

۱.۱ نکات مورد توجه

- 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 `main` پس از تکمیل تمرین فراموش نشود!

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_A2`	`A2`	`A2`	`A2`	`A2.Tests`	`A2`

۱.۲.۱ آماده‌سازی‌های مربوط به git

اگر چه در گارگاه git مفاهیم و روش کار با آن آموزش داده شد اما بار دیگر در اینجا کارهایی را که باید در ابتدای تمرین انجام دهید را مرور می‌کنیم.

✓ ابتدا به شاخه‌ی `main` بروید.

```
/c/git/AP00012 (fb_A2)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

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

```
/c/git/AP00012 (main)
$ git pull
remote: Azure Repos
remote: Found 8 objects to send. (90 ms)
Unpacking objects: 100% (8/8), done.
From https://40052XXXX.visualstudio.com/AP00012/_git/AP00012
   e7fd3b5..2cc74de  main                -> origin/main
Checking out files: 100% (266/266), done.
Updating e7fd3b5..2cc74de
Fast-forward
 A2/A2.sln                                    |    37 +
 A2/A2/A2.csproj                              |    61 +
```

```
 A2/A2/Program.cs                                |     15 +
 .
 .
 .
```

✓ یک شاخه‌ی جدید با نام `fb_A2` بسازید و تغییر شاخه دهید.

```
/c/git/AP00012 (main)
$ git checkout -b fb_A2
Switched to a new branch 'fb_A2'
/c/git/AP00012 (fb_A2)
$
```

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

۲.۲.۱ آماده‌سازی‌های مربوط به visual studio

ساختار فایل پایه‌ای که در اختیار شما قرار می‌گیرد به صورت زیر است:

```
+---A2
    \---Project
    |       ExceptionHandler.cs
    |
    \---ProjectTests
            AdvancedExceptionTests.cs
            ConstructorTests.cs
            FileNotFoundExceptionTests.cs
            FormatExceptionTests.cs
            GetMethodTests.cs
            IndexOutOfRangeExceptionTests.cs
            MultipleExceptionTests.cs
            OutOfMemoryExceptionTests.cs
            OverflowExceptionTests.cs
            SetMethodTests.cs
```

فایل(های) موجود در پوشه‌ی `Project` را به پروژه‌ی اصلی و فایل(های) موجود در پوشه‌ی `ProjectTests` را به پروژه‌ی تست (`A2.Tests`) اضافه (`Add`) کنید.

۲ پیاده‌سازی تمرین

انجام این تمرین علاوه بر درک مفهوم Exception نیاز به مقدار قابل توجهی دیباگ کردن و آزمون و خطا دارد. با توجه به پیچیدگی برخی تست‌ها، تعدادی از متدهای لازم پیاده‌سازی شده و بدون هیچ تغییری ۱۹ تا از ۳۷ تست پاس می‌شوند. این تست‌ها برای مطالعه شما گذاشته شده تا با مطالعه و دیباگ آنها بتوانید تست‌های دیگر را پاس کنید. توجه کنید که بعضی از تست‌ها مربوط به کلاس یا متد یکسان هستند و در حین سعی شما برای پاس شدن یک تست، ممکن است تست دیگری خطا بدهد. برای انجام این تمرین قطعا نیاز به دیباگ کردن تست‌ها دارید. لذا اگر در وی‌اس‌کد مشکلی برای دیباگ کردن تست‌ها دارید، حتما از هم‌کلاسی‌ها و اساتید حل تمرین کمک بگیرید. همچنین راه‌حل‌های پاس شدن تست‌ها قطعا یکتا نیستند. لذا شباهت بیش از اندازه کدها نشانه عدم رعایت صداقت در حل تمرین‌ها می‌باشد. همکاری و همفکری اشکالی ندارد. ولی در نهایت سعی کنید کد را از ابتدا خودتان بزنید که از یادگیری مطمئن شوید.

بجز موارد معدودی مثل `InvalidDataException` و `NotImplementedException` که لازم است مستقیم throw بشوند، در بقیه موارد نباید از دستور throw برای ایجاد استثناء استفاده کنید. بلکه باید شرایطی که باعث انداخته شدن آن استثناء می‌شود را فراهم کنید. مثل اینکه اگر رشته حرفی غیر عدد به `int.Parse` بدهید، استثناء `FormatException` را پرتاب می‌کند. در این حالت اگر شما بجای این، مستقیم استثناء `FormatException` را پرتاب کنید، جواب اشتباه است.

۱.۲ `ThrowIfOdd`

متد `ThrowIfOdd` را به گونه‌ای پیاده‌سازی کنید که در صورتی که عدد n ورودی فرد باشد یک استثنا [۱] از نوع `InvalidDataException` پرتاب [۲] کند. 34/2

۲.۲ `ExceptionHandler.ctor`

سازنده‌ی این کلاس را به گونه‌ای پیاده‌سازی کنید که در صورتی که متغیر ورودی `causeExceptionInConstructor` آن `true` باشد استثنائی از نوع `NullReferenceException` رخ دهد. دقت کنید که شما مجاز به ساخت استثنا جدید و پرتاب آن نیستید بلکه باید استثنا در زمان اجرا رخ دهد. 31/5

۳.۲ `ExceptionHandler.Input.Get`

`getter` کلاس `ExceptionHandler` را به گونه‌ای پیاده‌سازی کنید که در صورت `null` بودن `Input` منجر به رخ دادن استثنائی از نوع `NullReferenceException` شود. 28/8

۴.۲ `ExceptionHandler.Input.Set`

`setter` کلاس `ExceptionHandler` را به گونه‌ای پیاده‌سازی کنید که در صورت `null` بودن `value` منجر به رخ دادن استثنائی از نوع `NullReferenceException` شود.

۵.۲ `IndexOutOfRangeExceptionMethod()`

متد `IndexOutOfRangeExceptionMethod()` را به گونه‌ای پیاده‌سازی کنید که منجر به رخ دادن استثنائی از نوع `IndexOutOfRangeException` در آن شود.

در صورتی که مقدار `DoNotThrow` برابر `false` باشد باید استثنا پس از `catch` شدن مجددا پرتاب شود. در غیر این صورت باید مقدار `ErrorMsg` برابر با رشته‌ی `"Caught exception "` به علاوه‌ی نوع استثنا رخ داده شود. مثلا در این جا:

```
"Caught exception IndexOutOfRangeException"
```

۶.۲ `FormatExceptionMethod()`

متد `FormatExceptionMethod` برای مثال پیاده‌سازی شده و شما می‌توانید از این متد به عنوان راهنمایی برای پیاده‌سازی سایر متدها استفاده کنید

۷.۲ `FileNotFoundExeptionMethod()`

متد `FileNotFoundExeptionMethod` را به گونه‌ای پیاده‌سازی کنید که منجر به رخ دادن استثنائی از نوع `FileNotFoundExeption` در آن شود.

در صورتی که مقدار `DoNotThrow` برابر `false` باشد باید استثنا پس از `catch` شدن مجددا پرتاب شود. در غیر این صورت باید مقدار `ErrorMsg` برابر با رشته‌ی `"Caught exception "` به علاوه‌ی نوع استثنا رخ داده شود.

۸.۲ `OutOfMemoryExceptionMethod()`

متد `OutOfMemoryExceptionMethod` را به گونه‌ای پیاده‌سازی کنید که منجر به رخ دادن استثنائی از نوع `OutOfMemoryException` در آن شود.

در صورتی که مقدار `DoNotThrow` برابر `false` باشد باید استثنا پس از `catch` شدن مجددا پرتاب شود. در غیر این صورت باید مقدار `ErrorMsg` برابر با رشته‌ی `"Caught exception "` به علاوه‌ی نوع استثنا رخ داده شود.

[1] Exception
[2] throw

۹.۲ `OverflowExceptionMethod()`

متد `OverflowExceptionMethod` را به گونه‌ای پیاده‌سازی کنید که منجر به رخ دادن استثنائی از نوع `OverflowException` در آن شود.
در صورتی که مقدار `DoNotThrow` برابر `false` باشد باید استثنا پس از `catch` شدن مجددا پرتاب شود. در غیر این صورت باید مقدار `ErrorMsg` برابر با رشته‌ی `"Caught exception "` به علاوه‌ی نوع استثنا رخ داده شود.

۱۰.۲ `MultipleExeptionMethod()`

متد `MultipleExeptionMethod()` را به گونه‌ای پیاده‌سازی کنید که منجر به رخ دادن استثناهائی از نوع `IndexOutOfRangeException` و `OutOfMemoryException` در آن شود.
در صورتی که مقدار `DoNotThrow` برابر `false` باشد باید استثنا پس از `catch` شدن مجددا پرتاب شود. در غیر این صورت باید مقدار `ErrorMsg` برابر با رشته‌ی `"Caught exception "` به علاوه‌ی نوع استثنا رخ داده شود.

۱۱.۲ `FinallyBlockMethod()`

برای پیاده‌سازی این متد علاوه بر تسلط به مفهوم و چگونگی رفتار `try-catch-finally`، لازم است تست‌های زیر را با دقت مطالعه و دیباگ کنید.

- `TestFinallyBlockException`
- `TestFinallyBlockNoExceptionNoReturn`
- `TestFinallyBlockExceptionNoCatch`
- `TestFinallyBlockExceptionNoCatch`

توجه کنید که همانند قسمت‌های قبلی تست‌ها به هیچ وجه نباید هیچ تغییر کنند. هدف از دیباگ کردن تست‌ها فهم رفتار متدهای مربوطه می‌باشد. با توجه به پارامترهای سازنده کلاس `ExceptionHandler` و پارامتر ورودی متد `FinallyBlockMethod` رفتار این متد متفاوت است. با مطالعه این تست‌ها متوجه پارامترهایی که رفتار این متد را تغییر می‌دهند می‌شوید. سپس بدنه متد را به گونه‌ای پیاده‌سازی کنید که تست‌های یکی پس از دیگری پاس شوند. برای کمک به شما مقداری از این متد پیاده‌سازی شده است. چنانچه علاقمند به آزمون سخت‌تری از توانایی خود دارید، بدنه موجود را پاک کرده و از ابتدا پیاده‌سازی کنید. به فیلد `FinallyBlockStringOut` و چگونگی استفاده از آن‌ها در خود متد و تست‌ها نیز دقت کنید. یکی از روش‌های راستی‌آزمایی رفتار این متد استفاده از این فیلد است.

۱۲.۲ `NestedMethods`

هدف از این تست (معما) علاوه بر تمرین و تسلط به مطالعه دقیق کد و دیباگ کردن، راست‌آزمایی تسلط شما به رفتار استثناء و `try-cath` می‌باشد. وقتی یک استثناء پرت می‌شود در درون خود اطلاعات مسیر پرتاب یا افتادن را در فیلد `StackTrace` ذخیره می‌کند. با توجه به محتوای تست `NestedExceptionTest` معلوم می‌شود که متدهایی با نام‌های `MethodA`، `MethodB`، `MethodC`، و `MethodD` باید درست شوند و استثنایی با نوع «مناسب» و از محل «مناسب» به‌گونه‌ای پرتاب شود که متدهای بالا در مسیر آن قرار گیرند.

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:

```
/c/git/AP00012 (fb_A2)
$ git status
On branch fb_A2
Untracked files:
```

```
  (use "git add <file>..." to include in what will be committed)

    A2/

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

همان‌طور که مشاهده می‌کنید فولدر `A2` 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.

```
/c/git/AP00012 (fb_A2)
$ git add A2/*
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
/c/git/AP00012 (fb_A2)
On branch fb_A2
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   A2/A2.sln
    new file:   A2/A2/A2.csproj
    new file:   A2/A2/Program.cs
    new file:   A2/A2.Tests/A2.Tests.csproj
```

همانطور که مشاهده می‌کنید فولدر `A2` 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:

```
/c/git/AP00012 (fb_A2)
$ git commit -m "Implement HW2"
[fb_A2 c1f21df] Implement HW2
 15 files changed, 595 insertions(+)
 create mode 100644 A2/A2.sln
 create mode 100644 A2/A2/A2.csproj
 create mode 100644 A2/A2/Program.cs
 create mode 100644 A2/A2.Tests/A2.Tests.csproj
```

۴.۳ ارسال تغییرات انجام شده به Remote repository

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

```
/c/git/AP00012 (fb_A2)
$ git push origin fb_A2
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://40052XXXX.visualstudio.com/AP00012/_git/AP00012
 * [new branch]      fb_A2 -> fb_A2
```

۵.۳ ساخت Pull Request

در نهایت باید با مراجعه به سایت Azure DevOps یک `Pull Request` جدید با نام `HW2` بسازید به طوری که امکان `merge` کردن شاخه‌ی `fb_A2` را بر روی شاخه‌ی `main` 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.