دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ)
تمرین‌های سری چهاردهم

علی حیدری
استاد: سید صالح اعتمادی

مهلت ارسال: ۲ تیر ۱۳۹۸

فهرست مطالب

[1] Repository

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_A14`	`A14`	`A14`	`A14`	`A14Tests`	`HW14`

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 مخزن `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 +
 A14/A14.sln                                   |    37 +
 A14/A14/A14.csproj                            |    61 +
 A14/A14/App.config                            |     6 +
 A14/A14/Program.cs                            |    15 +
 A14/A14/Properties/AssemblyInfo.cs            |    36 +
 .
 .
```

```
.
```

✓ یک شاخه‌ی جدید با نام `fb_A14` بسازید و تغییر شاخه دهید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
$ git checkout -b fb_A14
Switched to a new branch 'fb_A14'
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A14)
$
```

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:

```
A14
  +---Project
  |       AccumulateState.cs
  |       Calculator.cs
  |       CalculatorSate.cs
  |       ComputeState.cs
  |       ErrorState.cs
  |       IState.cs
  |       PointState.cs
  |       Program.cs
  |       StartState.cs
  |
  \---ProjectTests
          ProgramTests.cs
```

در فایل پایه دو پوشه وجود دارد شما باید فایل(های) موجود در پوشه‌ی Project را به پروژه‌ی اصلی (A14) و فایل(های) موجود در پوشه‌ی ProjectTests را به پروژه‌ی تست (A14Tests) اضافه کنید.

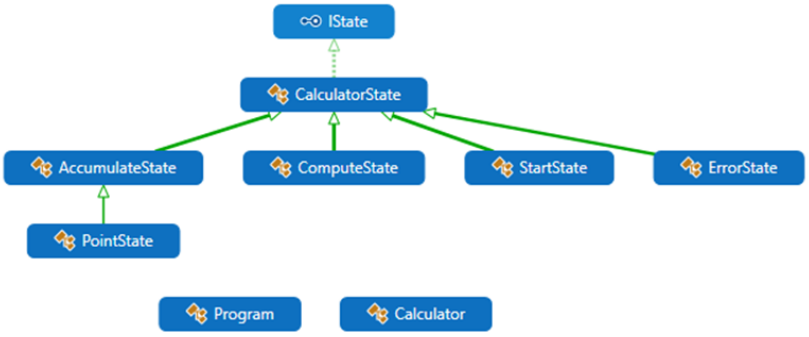
۲ پیاده‌سازی تمرین

هدف از این تمرین آشنایی شما با الگوهای برنامه‌سازی شیءگرا است. بطور خاص الگوی وضعیت[۲] .علاوه بر این مفاهیم زیر نیز مرور می‌شود

- Inheritance
- Abstract Class
- Virtual Method
- Interface
- Polymorphism
- Lambda Expressions
- Writing Testable Code

هدف دیگر این تمرین این است که از کد پیچیده وحشت نکنید. سعی کنید با خواندن کدها هر کلاس را به تنهایی متوجه بشوید که چکار می‌کند، روابط بین کلاس‌ها را هم متوجه بشوید. ممکن است که نتوانید تصور کنید کل مجموعه تمام کلاس‌ها با هم چکار می‌کنند، ولی اگر هر کلاس را متوجه بشوید و ارتباطش با کلاس‌های دیگر را هم متوجه بشوید، می‌توانید هر کلاس را درست پیاده‌سازی کنید. علاوه بر این به اهمیت و فایده یونیت تست پی خواهید برد که چقدر به دیباگ کردن و درست کردن کد کمک می‌کند. برای تمرین می‌توانید ابتدا از یونیت تست‌ها استفاده نکنید و ببینید آیا می‌توانید تمرین را انجام دهید؟

[2] state pattern



شکل ۱

موضوع این تمرین نوشتن یک ماشین حساب بسیار ساده است که چهار عملگر اصلی را انجام می‌دهد. برای نحوه کار ماشین حساب، از یک ماشین حساب ساده توی موبایل یا کامپیوترتان استفاده کنید. این ماشین حساب هم مثل آن‌ها عمل می‌کند. با این تفاوت که فقط از صفحه‌کلید ورودی دریافت می‌کند نه با کلیک یا لمس. به عنوان مثال یک ماشین حساب باز کنید. دکمه صفر را چند بار فشار دهید. آیا صفحه نمایش تغییری می‌کند؟ نقطه (ممیّز) را فشار دهید. چه اتفاقی می‌افتد؟ چند بار دیگر هم فشار دهید. آیا صفحه نمایش تغییری می‌کند؟ حالا چند بار صفر را فشار دهید. آیا صفحه نمایش این دفعه فرقی می‌کند؟ چرا؟ در ابتدا ماشین حساب در وضعیت شروع (StartState) بود. ولی بعد از این‌که نقطه را فشار دادید از وضعیت شروع خارج شد و به وضعیت نقطه (PointState) وارد شد. در اینجا اتفاقات کلیدهای ورودی هستند.
اگر با این دید به کد نگاه کنید، بهتر متوجه می‌شوید. کد را چندین بار مطالعه کنید اما از آن وحشت نکنید! قسمت‌هایی که نیاز به پیاده‌سازی شما دارد با علامت # و عدد کنار آن مانند: #۱ ، #۲ ، ... مشخص شده است. این ماشین حساب باید مثل یک ماشین حساب معمولی کار بکند. اگر مطمئن نیستید در یک حالتی باید چکار کند، یک ماشین حساب ساده بردارید و امتحان کنید. نمودار روابط کلاس ها در شکل ۱ موجود است. مثل تمرین قبل دقت، کنید که تعداد خط‌های کد که شما باید اضافه کنید کم است. از دیباگ کردن برنامه برای فهم این که چطوری کار می‌کند استفاده کنید. لطفا در کامنت‌ها توضیح اضافه در مورد روابط کلاس‌ها و این که هر کلاسی چرا عضو کلاس دیگر است یا از آن ارث میرد یا ... بدهید. اگر فقط روی جاهایی که کد پاک شده تمرکز کنید کارتان سخت خواهد شد. باید همه کلاس‌ها و روابطشان را متوجه بشوید تا بتونید جاهای خالی را پر کنید. بعد از اینکه از درست کار کردن برنامه اطمینان پیدا کردید، عملگر توان را به ماشین حساب اضافه کنید. نشانه عملگر توان کاراکتر ^ است.

۱.۲ تست Zero

بعد از درست کردن پروژه‌ها و اضافه کردن فایل‌ها این تست بدون هیچ تغییری از طرف شما، پاس می‌شود. متن این تست و بخصوص متد `RunTest` را با دقت مطالعه کرده و خط به خط اجرا/دیباگ کنید و از تسلط بر نحوه انجام تست اطمینان حاصل کنید. به طور خلاص هدف این تست این است که بعد از باز کردن ماشین‌حساب هر چند بار دکمه صفر فشار داده شود صفحه نمایش عدد صفر را نشان داده و تغییری نمی‌کند. ۱۳/۱

۲.۲ تست Accumulation

برای پیدا کردن شهود نسبت به هدف این تست برنامه `calc.exe` و یا یک ماشین‌حساب دستی باز کنید. هر چند بار عدد صفر را بزنید، آیا تغییری در صفحه نمایش مشاهده می‌کنید؟ حال، ابتدا دکمه یک را زده و بعد دکمه صفر را به دفعات فشار دهید. آیا متوجه تفاوت رفتار می‌شوید؟ برای پاس شدن این تست لازم است از دیباگر استفاده کرده و تغییرات لازم را در کد ایجاد کرده تا تست پاس بشود. ۱۲/۲

۳.۲ تست AccumulateState

هدف این تست مانند بخش قبل است. با این تفاوت که وقتی ماشین‌حساب اجرا می‌شود، هر عددی غیر از صفر باید رفتاری متفاوت از عدد صفر داشته باشد. ولی همانطور که در تست قبل مشاهده کردید، این رفتار متفاوت فقط در ابتدا است. بعد از اینکه عددی غیر از صفر وارد شد، دیگر عدد صفر با دیگر اعداد تفاوتی نمی‌کند. ۱۱/۳

۴.۲ تست PointState

هدف این تست،آزمون وارد کردن درست اعداد اعشاری در ماشین حساب می‌باشد. ۱۰/۴

۵.۲ تست PointsOnlyState

برای پاس شدن این تست لازم است از دیباگر استفاده کرده و تست را دیباگ کنید. این تست زمانی پاس می‌شود که توی ماشین‌حساب هر چند بار دکمه نقطه فشار داده شود، فقط یک صفر و یک نقطه روی صفحه نمایش نشان داده شود. برای نمونه و مقایسه می‌توانید از یک ماشین‌حساب دستی یا برنامه `calc.exe` استفاده کنید. ۹/۵

۶.۲ تست ExtraPoint

مجددا برای فهمیدن هدف این تست از یک ماشین‌حساب استفاده کنید. آیا بعد از اینکه مثلا عدد یک و یک دهم را وارد کردید فشار دادن دکمه نقطه تغییری در ماشین‌حساب ایجاد می‌کند؟ این نشان‌دهنده این است که بعد از وارد کردن نقطه، ماشین‌حساب به حالتی وارد می‌شود که وارد کردن نقاط بعدی صفحه نمایش را تغییر نمی‌دهد. مجددا با کمک گرفتن از دیباگر تغییرات لازم در برنامه را انجام دهید. ۸/۶

۷.۲ تست StartState

حال که اعداد به درستی به ماشین‌حساب وارد می‌شوند، لازم است که بتوانیم محاسبات بین اعداد را انجام دهیم. برای گام اول،این تست زمانی پاس می‌شود که اگر دکمه بعلاوه فشار داده شد، ماشین‌حساب به حالت `ComputeState` تغییر حالت دهد. ۷/۷

۸.۲ تست Sum

بعد از این مقدمات، لازم است که ماشین حساب یک حساب ساده را بتواند انجام دهد. به این معنی که بعد از فشار دادن دکمه مساوی، نتیجه محاسبه نمایش داده شود. ۶/۸

۹.۲ تست ErrorState

چنانچه بعد از نمایش نتیجه یک محاسبه، دکمه مساوی مجددا فشار داده شود، لازم است حالت ماشین‌حساب به `ErrorState` تغییر پیدا کند و صفحه نمایش تغییری نکند. ۵/۹

۱۰.۲ تست Multiply

حال که عملگر جمع به درستی پیاده‌سازی شد، نوبت عملگر ضرب می‌باشد. تغییرات لازم برای پاس شدن این تست را اعمال کنید. ۴/۱۰

۱۱.۲ تست MultipleSum

چنانچه تست‌های قبل به درستی پیاده‌سازی شده باشند، این تست نیز بدون هیچ تغییری باید پاس بشود. ۳/۱۱

۱۲.۲ تست Divide

این تست درستی اجرای عملگر تقسیم را راست‌آزمایی می‌کند. ۲/۱۲

۱۳.۲ تست StartingPoint

چنانچه تست‌های قبل به درستی پیاده‌سازی شده باشند، این تست نیز بدون هیچ تغییری باید پاس بشود. ۱/۱۳

۱۴.۲ تست Power

حال که چهار عمل اصلی را برای ماشین‌حساب پیاده سازی کردید، نوبت پیاده‌سازی یک عملگر جدید می‌باشد. تغییرات لازم را برای پاس شدن تست توان اعمال کنید. ۰/۱۴

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_A14)
$ git status
On branch fb_A14
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    A14/

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

همان‌طور که مشاهده می‌کنید فولدر `A14` 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_A14)
$ git add A14/*
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A14)
$ git status
On branch fb_A14
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   A14/A14.sln
    new file:   A14/A14/A14.csproj
    new file:   A14/A14/App.config
    new file:   A14/A14/Program.cs
    new file:   A14/A14/Properties/AssemblyInfo.cs
    new file:   A14/A14Tests/A14Tests.csproj
    new file:   A14/A14Tests/Properties/AssemblyInfo.cs
    new file:   A14/A14Tests/packages.config
    .
    .
    .
```

همانطور که مشاهده می‌کنید فولدر `A14` 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_A14)
$ git commit -m "Implement HW14"
[fb_A14 c1f21df] Implement HW14
 15 files changed, 595 insertions(+)
 create mode 100644 A14/A14.sln
 create mode 100644 A14/A14/A14.csproj
 create mode 100644 A14/A14/App.config
 create mode 100644 A14/A14/Program.cs
 create mode 100644 A14/A14/Properties/AssemblyInfo.cs
 create mode 100644 A14/A14Tests/A14Tests.csproj
```

```
 create mode 100644 A14/A14Tests/Properties/AssemblyInfo.cs
 create mode 100644 A14/A14Tests/packages.config
.
.
.
```

۴.۳ ارسال تغییرات انجام شده به مخزن Remote

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

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A14)
$ git push origin fb_A14
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_A14 -> fb_A14
```

۵.۳ ساخت Pull Request

با مراجعه به سایت Azure DevOps یک Pull Request جدید با نام `HW14` بسازید به طوری که امکان `merge` کردن شاخه‌ی `fb_A14` 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.