دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ)
تمرین‌های سری پنجم (کلاس و اشیا)

مدرس: سید صالح اعتمادی
طراحی و تدوین: مبین داریوش، سید حسین ساداتی‌پور، علی حیدری

مهلت ارسال: ۲۰ فروردین ۱۳۹۸

فهرست مطالب

۱ مقدمه و آماده‌سازی

برای یک برنامه‌نویس خوب شدن، لازم است تا موارد گوناگونی را رعایت و تمرین کنید تا به مرور مهارت برنامه‌نویسی خود را بهبود ببخشید و به نتیجه مطلوب برسید، یکی از موارد بسیار مهم و کلیدی آن است که بیش‌تر از آنکه توانایی کدنویسی خود را افزایش دهید، توانایی فکر کردن الگوریتمی و حل چالش را در خود افزایش دهید؛ یک برنامه‌نویس حرفه‌ای پیش از آنکه وارد مرحله کد نویسی شود، مشکل را بررسی کرده و راه حل برطرف‌سازی آن را به دست می‌آورد و پس از آنکه از این مرحله به خوبی گذشت، می‌تواند با دانش کدنویسی خود، برنامه‌ای بنویسد که به خوبی در حل مشکل مفید واقع شود. اما! آنچه در این درس از شما می‌خواهیم پیشرفت تفکر الگوریتمی همراه با پیشرفت مهارت کدنویسی است. سوالات این سری، شاید در نگاه اول برای شما کمی نامفهوم به نظر برسد که احتمالا بخاطر جدید بودن مدل تفکر برای حل مسئله است. برای درک و فهمیدن سوالات تنها راه‌حل ممکن صرف وقت کافی برای انجام این تکلیف و پیدا کردن ارتباط بین اجزای مختلف برنامه می‌باشد. پس خواهشی که از شما داریم آن است که برای رسیدن به اهدافتان، وقت کافی و لازم را در نظر بگیرید.

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 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 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_A3`	A3	A3	A3	A3Tests	HW5

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 `master` بروید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_hw3)
$ 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://9652XXXX.visualstudio.com/AP97982/_git/AP97982
   e7fd3b5..2cc74de  master               -> origin/master
Checking out files: 100% (266/266), done.
Updating e7fd3b5..2cc74de
Fast-forward
 .gitattributes                                |    63 +
 A2/A2.sln                                     |    37 +
 A2/A2/A2.csproj                               |    61 +
 A2/A2/App.config                              |     6 +
 A2/A2/Program.cs                              |    15 +
 A2/A2/Properties/AssemblyInfo.cs              |    36 +
 .
 .
 .
```

✓ یک شاخه‌ی جدید با نام `fb_A3` بسازید و تغییر شاخه دهید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
$ git checkout -b fb_A3
Switched to a new branch 'fb_A3'
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A3)
$
```

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:

```
+---A3
    +---Project
            City.cs
            Customer.cs
            Order.cs
            Product.cs
            Shop.cs

    +---ProjectTests
            GradedTests.cs
            TestData.cs
```

شما باید فایل‌های موجود در پوشه‌ی `Project` را به پروژه‌ی اصلی (`A3`) اضافه (`Add`) کنید. و فایل‌های موجود در پوشه‌ی `ProjectTests` را به پروژه‌ی تست (`A3Tests`) اضافه (`Add`) کنید.

۲ پیاده‌سازی کلاس‌ها

۱.۲ توضیحات کلی

فرض کنید صاحب یک فروشگاه اینترنتی هستید، برای مدیریت این فروشگاه لازم است تا داده‌های مختلفی را سامان‌دهی کنید. برای مثال ممکن است شما خریدارانی داشته باشید، هر کدام از این افراد سبدهای خرید گوناگونی دارند و هر سبد خرید متناسب با کالاهایی که در آن وجود دارد، قیمتی متفاوت با یک‌دیگر دارند. از طرفی شما به عنوان فروشگاه از هر کالا تعداد معینی در دسترس دارید و نمی‌توانید بیش از آنچه که موجود دارید از مشتریان خود سفارش بگیرید. نکته دیگر آن است که برای هر سبد خرید، باید بررسی شود که آیا یک سبد خرید به خریدار تحویل داده شده است یا خیر و در صورتی که تحویل داده نشده است، پیگیری شود تا هر چه سریع‌تر به خریدار آن تحویل گردد.
می‌خواهیم برنامه‌ای برای مدیریت فروشگاه بنویسیم که بتوانیم اطلاعات زیر را با آن به دست آوریم:

۱. قیمت کل هر **سبد خرید**

۲. سفارشات تحویل نشده‌ی هر **مشتری**
۳. **کالایی** که مشتری از آن بیشترین بار سفارش داده
۴. **شهرهایی** که مشتریان در آن زندگی می‌کنند
۵. مشتری(هایی) که در یک شهر خاص زندگی می‌کنند
۶. مشتری(هایی) که بیشترین تعداد سفارش را داشته‌اند

هنر برنامه‌نویسی تبدیل فضای مسئله به فضای راه‌حل با استفاده از امکانات زبان برنامه‌نویسی است. برای این کار در یک زبان شیگرا شما باید ابتدا موجودیت‌های مسئله را شناسایی کنید و سپس مطابق با رفتارهای آن و خواسته‌های مسئله به حل مسئله بپردازید.
خوب به صورت مسئله فکر کنید در این مسئله چه موجودیت‌هایی وجود دارند؟ روابط بین این موجودیت‌ها چگونه است؟هر موجودیت چه رفتارهایی از خود نشان می‌دهد؟

۲.۲ کلاس City

در اینجا کمی تبدیل فضای مسئله به فضای راه‌حل با استفاده از زبان شیگرای سی‌شارپ را تمرین می‌کنیم:
موجودیت شهر دارای ویژگی «نام» است پس کلاس `City` دارای یک `property` از نوع `string` به نام `Name` می‌باشد که بیانگر نام شهر است. ممکن است موجودیتی که در نظر می‌گیریم ویژگی(های) دیگری هم داشته باشد اما ما فقط ویژگی‌هایی را در نظر می‌گیریم که به حل مسئله کمک می‌کند.
گام اول: برای هر `property` ای که در کلاس وجود دارد `getter` و `setter` مناسب بنویسید.
گام دوم: شما باید سازنده (`constructor`) این کلاس را تکمیل کنید تا شیای که از کلاس ساخته می‌شود معتبر باشد یعنی هر `City` ای که ساخته می‌شود لزوما دارای ویژگی نام باشد.
پس از پیاده‌سازی صحیح سازنده‌ی این کلاس تستِ `CityConstructorTest` پاس خواهد شد.[۱۰/۱]
بعد از پیاده‌سازی این کلاس و پاس شدن تست‌های آن کار شما با این کلاس و فایل آن تمام شده است. دیگر نیازی به تغییر این کد نخواهید داشت.

۳.۲ کلاس Product

یکی دیگر از موجودیت‌های مسئله کالاست. هر کالا دارای ویژگی‌های مربوط به خود می‌باشد که ویژگی‌هایی که در این مسئله برای ما اهمیت دارد نام کالا و قیمت آن است. بنابراین برای کلاس `Product` دو `property` به نام‌های `Name` و `Price` در نظر می‌گیریم.
گام اول: برای هر `property` ای که در کلاس وجود دارد `getter` و `setter` مناسب بنویسید.
گام دوم: سازنده‌ی این کلاس را تکمیل کنید به طوری که بتوان با پاس دادن قیمت کالا و نام آن یک شی جدید از نوع `Product` ساخت که لزوما دارای مقادیر پاس شده در سازنده باشد. پس از پیاده‌سازی صحیح سازنده‌ی کلاس، تستِ `ProductConstructorTest` پاس خواهد شد.[۹/۲]
بعد از پیاده‌سازی این کلاس و پاس شدن تست‌های آن کار شما با این کلاس و فایل آن تمام شده است. دیگر نیازی به تغییر این کد نخواهید داشت.

۴.۲ کلاس Order

هر کاربر می‌تواند یک یا چند سبد خرید داشته باشد و هر سبد خرید هم می‌تواند از یک یا چند کالا تشکیل شده باشد. برای موجودیت «سبد خرید» کلاسی به نام `Order` در نظر می‌گیریم. هر سبد خرید هم می‌تواند دو حالت داشته باشد: یا به دست مشتری رسیده و یا نرسیده. با توجه به آنچه گفته شد برای کلاس `Order` دو `property` به نام‌های `Products` و `IsDelivered` انتخاب کرده‌ایم.
گام اول: برای هر `property` ای که در کلاس وجود دارد `getter` و `setter` مناسب بنویسید.
گام دوم: سازنده‌ی این کلاس را تکمیل کنید به طوری که بتوان با پاس دادن لیست کالاها و وضعیت تحویل آن یک شی جدید از نوع `Order` ساخت که لزوما دارای مقادیر پاس شده در سازنده باشد. پس از پیاده‌سازی صحیح سازنده‌ی این کلاس تستِ `OrderConstructorTest` پاس خواهد شد.[۸/۳]
می‌توان در یک بیان ساده گفت که رفتارهای موجودیت‌ها در دنیای واقعی همان متدهای کلاس و اشیا در دنیای برنامه‌نویسی است. موجودیت «سبد خرید» یک رفتار در نظر گرفته‌ایم و آن رفتار این است که قیمت کل سبدخرید را با توجه به کالاهایی که در آن وجود دارد استخراج کنیم.
گام سوم: شما باید متد `CalculateTotalPrice` را به گونه‌ای پیاده‌سازی کنید که قیمت کل سبد خرید را با توجه به کالاهایی که در آن وجود دارد برگرداند. پس از پیاده‌سازی صحیح این متد تستِ `CalculateTotalPriceTest` پاس خواهد شد.[۷/۴]
بعد از پیاده‌سازی این کلاس و پاس شدن تست‌های آن کار شما با این کلاس و فایل آن تمام شده است. دیگر نیازی به تغییر این کد نخواهید داشت.

۵.۲ کلاس Customer

موجودیت دیگری که در مسئله وجود دارد «مشتری» است. هر مشتری ویژگی‌هایی مانند اسم، آدرس، تلفن همراه، شهر محل زندگی، سفارشات و... می‌باشد که ما برای حل این مسئله فقط به سه ویژگیِ «نام»، «شهر محل زندگی» و «سفارش‌ها»ی این موجودیت نیاز داریم بنابراین باید این ویژگی‌ها را در قالب `property` در کلاس بیان کنیم. برای مشتری `property` هایی به نام `Name` ، `City` و `Orders` در نظر گرفته‌ایم.

گام اول: برای هر `property` ای که در کلاس وجود دارد `getter` و `setter` مناسب بنویسید.

گام دوم: شما باید سازنده‌ی این کلاس را تکمیل کنید به طوری که بتوان با پاس دادن نام مشتری، شهر محل زندگی مشتری و سبدهای خرید آن شیای از نوع کلاس `Customer` ساخت که لزوما دارای مقادیر پاس شده در سازنده باشد. پس از پیاده‌سازی صحیح سازنده‌ی این کلاس تستِ `CustomerConstructorTest` پاس خواهد شد.[۶/۵]

همان‌طور که در توضیحات کلاس `Order` گفته شد باید رفتارهای اشیا را نیز بررسی کنیم و اگر در حل صورت مسئله به ما کمک می‌کند آن‌را پیاده‌سازی کنیم. برای هر شیای که از این کلاس ساخته می‌شود دو رفتار در نظر گرفته‌ایم. یک رفتار این است که کالایی مشتری بیشترین تعداد سفارش از آن را داشته است را سوال کنیم. رفتار دیگر این است که سفارشاتی که به او تحویل نشده است را بپرسیم.

گام سوم: متد `MostOrderedProduct` را به گونه‌ای پیاده‌سازی کنید که کالایی که مشتری بیش‌ترین بار از آن سفارش داده است برگرداند. اگر چند کالا داریم که بیش‌ترین بار از آن سفارش داده شده یکی را به دل‌خواه برگردانید. پس از پیاده‌سازی صحیح این متد تستِ `MostOrderedProductTest` پاس خواهد شد.[۶/۵]

گام چهارم: متد `UndeliveredOrders` را به گونه‌ای پیاده‌سازی کنید که کالاهایی که به مشتری تحویل داده نشده است را برگرداند. پس از پیاده‌سازی صحیح این متد تستِ `UndeliveredOrdersTest` پاس خواهد شد.[۴/۷]

بعد از پیاده‌سازی این کلاس و پاس شدن تست‌های آن کار شما با این کلاس و فایل آن تمام شده است. دیگر نیازی به تغییر این کد نخواهید داشت.

۶.۲ کلاس Shop

موجودیت آخری که برای حل مسئله در نظر گرفته‌ایم «فروشگاه» است. هر فروشگاهی ویژگی‌هایی مانند مشتریان، نام، آدرس و تاریخ افتتاح و... دارد که ما برای حل مسئله فقط دو ویژگی نام و مشتریان را در نظر گرفته‌ایم. با توجه به آنچه گفته شد برای کلاس `Shop` دو `property` به نام‌های `Name` و `Customers` در نظر گرفته‌ایم.

گام اول: برای هر `property` ای که در کلاس وجود دارد `getter` و `setter` مناسب بنویسید.

گام دوم: شما باید سازنده‌ی این کلاس را تکمیل کنید به طوری که بتوان با پاس دادن نام فروشگاه و مشتریان آن شیای از نوع کلاس `Shop` ساخت که لزوما دارای مقادیر پاس شده در سازنده باشد. پس از پیاده‌سازی صحیح سازنده‌ی این کلاس تستِ `ShopConstructorTest` پاس خواهد شد.[۳/۸]

همان‌طور که در توضیحات کلاس‌های قبلی گفته شد باید رفتارهای اشیا را نیز بررسی کنیم. برای هر شیای که از این کلاس ساخته می‌شود سه رفتار در نظر گرفته‌ایم. یک رفتار این است که شهرهایی که فروشگاه از آن مشتری دارد را استخراج کنیم. رفتار دیگر این است که مشتریانی که در یک شهر بخصوص زندگی می‌کنند و مشتری فروشگاه هستند را استخراج کنیم. رفتار آخر یافتن مشتری‌ای است که بیشترین تعداد سفارش تحویل نشده دارد.

گام سوم: متد `CitiesCustomersAreFrom` را به گونه‌ای پیاده‌سازی کنید که شهرهایی که مشتریان فروشگاه در آن زندگی می‌کنند را برگرداند. پس از پیاده‌سازی صحیح این متد تستِ `CitiesCustomersAreFromTest` پاس خواهد شد.[۲/۹]

گام چهارم: متد `CustomersFromCity` را به گونه‌ای پیاده‌سازی کنید که با گرفتن یک شی از نوع کلاس `City` مشتریانی که از در آن شهر زندگی می‌کنند را برگرداند. پس از پیاده‌سازی صحیح این متد تستِ `CustomersFromCityTest` پاس خواهد شد.[۱/۱۰]

گام پنجم: متد `CustomersWithMostOrders` را به گونه‌ای پیاده‌سازی کنید که مشتری یا مشتریانی که بیشترین تعداد سفارش از فروشگاه را داشته‌اند را برگرداند. پس از پیاده‌سازی صحیح این متد تستِ `CustomersWithMostOrdersTest` پاس خواهد شد.[۰/۱۱]

بعد از پیاده‌سازی این کلاس و پاس شدن تست‌های آن کار شما با این کلاس و فایل آن تمام شده است. دیگر نیازی به تغییر این کد نخواهید داشت.

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_A3)
$ git status
On branch fb_A3
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    A3/

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

همان‌طور که مشاهده می‌کنید فولدر `A0` 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_A3)
$ git add .
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A3)
On branch fb_A3
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   A3/A3.sln
    new file:   A3/A3/A3.csproj
    new file:   A3/A3/App.config
    new file:   A3/A3/City.cs
    new file:   A3/A3/Customer.cs
    new file:   A3/A3/Order.cs
    new file:   A3/A3/Product.cs
    new file:   A3/A3/Program.cs
    new file:   A3/A3/Properties/AssemblyInfo.cs
    new file:   A3/A3/Shop.cs
    new file:   A3/A3Tests/A3Tests.csproj
    new file:   A3/A3Tests/GradedTests.cs
    new file:   A3/A3Tests/Properties/AssemblyInfo.cs
    new file:   A3/A3Tests/TestData.cs
    new file:   A3/A3Tests/packages.config
```

همانطور که مشاهده می‌کنید فولدر `A3` 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_A3)
$ git commit -m "Implement HW5"
[fb_A3 c1f21df] Implement HW5
 15 files changed, 595 insertions(+)
 create mode 100644 A3/A3.sln
 create mode 100644 A3/A3/A3.csproj
 create mode 100644 A3/A3/App.config
 create mode 100644 A3/A3/City.cs
 create mode 100644 A3/A3/Customer.cs
 create mode 100644 A3/A3/Order.cs
```

```
 create mode 100644 A3/A3/Product.cs
 create mode 100644 A3/A3/Program.cs
 create mode 100644 A3/A3/Properties/AssemblyInfo.cs
 create mode 100644 A3/A3/Shop.cs
 create mode 100644 A3/A3Tests/A3Tests.csproj
 create mode 100644 A3/A3Tests/GradedTests.cs
 create mode 100644 A3/A3Tests/Properties/AssemblyInfo.cs
 create mode 100644 A3/A3Tests/TestData.cs
 create mode 100644 A3/A3Tests/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_A3)
$ git push origin fb_A3
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://9652XXXX.visualstudio.com/AP97982/_git/AP97982
 * [new branch]      fb_A3 -> fb_A3
```

۵.۳ ساخت Pull Request

در نهایت باید با مراجعه به سایت Azure DevOps یک `Pull Request` جدید با نام `HW5` بسازید به طوری که امکان `merge` کردن شاخه‌ی `fb_A3` 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.