دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ)
تمرین‌های سری یازدهم (وراثت ساده)

علی حیدری، محمدمهدی عبداله‌پور
استاد: سید صالح اعتمادی

مهلت ارسال: ۴ 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 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_A11`	`A11`	`A11`	`A11`	`A11Tests`	`HW11`

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_A10)
$ 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 +
 A11/A11.sln                                   |    37 +
 A11/A11/A11.csproj                            |    61 +
 A11/A11/App.config                            |     6 +
 A11/A11/Program.cs                            |    15 +
 A11/A11/Properties/AssemblyInfo.cs            |    36 +
 .
 .
 .
```

✓ یک شاخه‌ی جدید با نام `fb_A11` بسازید و تغییر شاخه دهید.

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
$ git checkout -b fb_A11
Switched to a new branch 'fb_A11'
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A11)
$
```

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:

```
A11
+---Project
\---ProjectTests
        AccountTests.1.Base.cs
        AccountTests.2.Savings.cs
        AccountTests.3.Checking.cs

2 directories, 3 files
```

در فایل پایه دو پوشه وجود دارد شما باید فایل(های) موجود در پوشه‌ی Project را به پروژه‌ی اصلی (A11) و فایل(های) موجود در پوشه‌ی ProjectTests را به پروژه‌ی تست (A11Tests) اضافه کنید.

۲ پیاده‌سازی تمرین

۱.۲ تست `AccountConstructor`

یک کلاس پایه به نام `Account` درست کنید که شامل یک عضو دابل به نام `Balance` باشد که نماینده ی میزان پول موجود در یک حساب است. این کلاس باید یک `constructor` داشته باشد که یک مقدار اولیه برای موجودی حساب میگیرد و با استفاده از آن عضو داده مربطه را مقداردهی میکند. `constructor` باید چک کند که مقدار اولیه برای موجودی حساب بزرگتر یا مساوی صفر است و اگر نبود مقدار اولیه موجودی برابر صفر باشد و یک پیغام خطای مناسب (طبق تست) چاپ کند. ۱۲/۱

۲.۲ تست `AccountCredit`

کلاس `Account` باید یک متد `Credit` داشته باشد. متد `Credit` باید مقداری را به موجودی اضافه کند. ۱۱/۲

۳.۲ تست `AccountCreditNegativeAmount`

اگر مقدار واریزی منفی بود باید پیغام خطای مناسب (طبق تست) چاپ کند و مقدار موجودی را دست نخورده نگه دارد. ۱۰/۳

۴.۲ تست AccountDebit

کلاس Account باید یک متد به نام Debit داشته باشد. متد Debit باید مقداری را از حساب برداشت کند و همچنین اطمینان حاصل کند که مقدار برداشتی بیشتر از موجودی نشود. اگر شد باید موجودی دست نخورده باقی بماند و پیغام خطای مناسب (طبق تست) چاپ شود. ۹/۴

۵.۲ تست SavingsAccountConstructor

کلاس SavingsAccount از کلاس Account ارث بری میکند. این کلاس یک عضو double دارد که نماینده ی نرخ سود است که در constructor خود آن را گرفته و مقداردهی میکند. ۸/۵

۶.۲ تست SavingsAccountCredit

متد Credit ارث برده شده و بدون تغییر باقی میماند. بنابراین لازم است رفتار این متد مطابق با رفتارش در کلاس مرجع باشد. ۷/۶

۷.۲ تست SavingsAccountCreditNegativeAmount

متد Credit ارث برده شده و بدون تغییر باقی میماند. بنابراین لازم است رفتار این متد مطابق با رفتارش در کلاس مرجع باشد. ۶/۷

۸.۲ تست SavingsAccountDebit

متد Debit ارث برده شده و بدون تغییر باقی میماند. بنابراین لازم است رفتار این متد مطابق با رفتارش در کلاس مرجع باشد. ۵/۸

۹.۲ تست SavingsAccountCalculateInterest

کلاس SavingsAccount یک متد public به نام CalculateInterest دارد که مقدار سود را طبق موجودی حساب و نرخ سود آن حساب میکند ۴/۹

۱۰.۲ تست CheckingAccountConstructor

کلاس CheckingAccount از کلاس Account ارث بری میکند. این کلاس یک عصو double به نام TransactionFee دارد که نشانگر مقدار کارمزد به ازای هر تراکنش است و مقدار آن را در constructor خود دریافت میکند. ۳/۱۰

۱۱.۲ تست CheckingAccountCredit

باید متد Credit را به گونه ای تغییر دهید که مقدار کارمزد به ازای هر واریز کسر شود. ۲/۱۱

۱۲.۲ تست CheckingAccountCreditNegativeAmount

اگر تراکنش موفق نبود نباید کارمزدی کم شود ۱/۱۲

۱۳.۲ تست CheckingAccountDebit

متد Debit را به گونه ای تغییر دهید که مقدار کارمزد به ازای هر برداشت کسر شود. اگر تراکنش موفق نبود نباید کارمزدی کم شود ۰/۱۳

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_A11)
$ git status
On branch fb_A11
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    A11/

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

همان‌طور که مشاهده می‌کنید فولدر `A11` 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_A11)
$ git add A11/*
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A11)
On branch fb_A11
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   A11/A11.sln
    new file:   A11/A11/A11.csproj
    new file:   A11/A11/App.config
    new file:   A11/A11/Program.cs
    new file:   A11/A11/Properties/AssemblyInfo.cs
    new file:   A11/A11Tests/A11Tests.csproj
    new file:   A11/A11Tests/Properties/AssemblyInfo.cs
    new file:   A11/A11Tests/packages.config
    .
    .
    .
```

همانطور که مشاهده می‌کنید فولدر `A11` 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_A11)
$ git commit -m "Implement HW11"
[fb_A11 c1f21df] Implement HW11
 15 files changed, 595 insertions(+)
 create mode 100644 A11/A11.sln
 create mode 100644 A11/A11/A11.csproj
 create mode 100644 A11/A11/App.config
 create mode 100644 A11/A11/Program.cs
 create mode 100644 A11/A11/Properties/AssemblyInfo.cs
 create mode 100644 A11/A11Tests/A11Tests.csproj
 create mode 100644 A11/A11Tests/Properties/AssemblyInfo.cs
```

```
 create mode 100644 A11/A11Tests/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_A11)
$ git push origin fb_A11
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_A11 -> fb_A11
```

۵.۳ ساخت Pull Request

با مراجعه به سایت Azure DevOps یک Pull Request جدید با نام `HW11` بسازید به طوری که امکان `merge` کردن شاخه‌ی `fb_A11` 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.