گیت و مکافات

- محدودیت زمان: نامحدود
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: ساده
- طراح: احسان حبیب آگهی

آقا مهندس یک برنامه نویس تازه کار است که برای اولین بار از گیت برای مدیریت کد های پروژه پایانی خود استفاده می کند. او بعد از ساعت ها تلاش و کار کردن روی پروژه تغییرات خودش را با پیغام `feat(network): add client` کامیت(commit) می کند. او برای بررسی و دیدن تغییراتی که هم تیمی اش قبل از او commit کرده به اشتباه دستور `git reset --hard <SOME-COMMIT>` را اجرا می کند.

با این کار، HEAD به کامیت هم‌تیمی‌اش بازمی‌گردد، اما حالا با یک مشکل بزرگ مواجه شده است: **کامیت خودش و هر تغییری که پس از آن انجام شده، حذف شده‌اند!** مهندس که نگران از دست رفتن تمام زحمات خود و هم‌تیمی‌اش بود، پس از کمی جستجو و پرسش از GPT متوجه شد که با استفاده از دستور `git reflog` می توان پروژه را نجات داد. حالا با جواب دادن به سوالات زیر به مهندس کمک کنید (:

نوکته 💡 **:** چیت شیت گیت رو از اینجا و اونجا دانلود کنید و میتونید نسخه چاپ شده ش رو توی اتاقتون بچسبونید

سوالات:

۱. `git reflog` چگونه کار می‌کند و چه اطلاعاتی را نمایش می‌دهد؟ توضیح دهید که چگونه می تواند در بازیابی کامیت‌های از دست رفته کمک کند.

۲. با توجه به دستوراتی که مهندس تاکنون اجرا کرده است، تمامی گام‌های لازم برای بازگرداندن پروژه به حالت قبل را بنویسید. در نهایت، پروژه باید با یک کامیت جدید و پیام `"prev commit restore"` به وضعیت اولیه بازگردد.

۳. تفاوت بین `git reset --hard` ، `git reset --soft` و `git reset --mixed` را توضیح دهید. هرکدام چه اثری روی فایل‌های کاری (working directory)، ایندکس (staging area) و تاریخچه‌ی کامیت دارند؟

۴. برای جلوگیری از این مشکل در آینده، مهندس و هم تیمی اش برای بازگشت به کامیت‌های قبلی، به جای `git reset` از چه دستورات دیگری باید استفاده کنند؟ چند روش جایگزین نام ببرید و نحوه‌ی کار هر یک را توضیح دهید.

جواب های خودتون رو توی یک فایل zip شامل یک فایل (ترجیحا)PDF آپلود کنید ×_×

سیستم تاکسی اینترنتی

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: روزبه سلطانی

در یک شهر بزرگ، تاکسی‌های اینترنتی به یکی از اصلی‌ترین راه‌های جابه‌جایی مردم تبدیل شده‌اند. هر راننده در نقاط مختلف شهر قرار دارد و منتظر دریافت درخواست‌های سفر است. مسافران پس از ثبت درخواست، انتظار دارند که نزدیک‌ترین راننده‌ی آزاد درخواست آن‌ها را قبول کند و آن‌ها را به مقصدشان برساند.

اما همه‌ی راننده‌ها یکسان نیستند؛ برخی راننده‌های VIP هستند که خدمات بهتری ارائه می‌دهند، و برخی دیگر تجربه و امتیاز بالاتری دارند. در این سیستم، هزینه‌ی سفر بسته به مسافت، کیفیت راننده و VIP بودن او متغیر است.

اکنون شما مسئول طراحی سیستمی هستید که پس از دریافت درخواست سفر، مناسب‌ترین راننده را انتخاب کند و هزینه‌ی سفر را محاسبه نماید. آیا می‌توانید این سیستم را به‌درستی پیاده‌سازی کنید؟

شما باید 3 کلاس زیر را تعریف کنید :

کلاس `Driver` :

کلاس `Driver` اطلاعات مربوط به یک راننده را ذخیره می‌کند، از جمله نام، موقعیت مکانی، امتیاز، تعداد سفرهای انجام‌شده و اینکه آیا راننده VIP است یا خیر. همچنین شامل متدهایی برای محاسبه فاصله‌ی راننده تا مسافر و تغییر موقعیت راننده پس از انجام سفر است.

این کلاس شامل ویژگی های زیر است :

- ویژگی `name` از نوع String : نام راننده
- ویژگی `isVIP` از نوع boolean : آیا راننده VIP است یا خیر
- ویژگی `rating` از نوع double : میانگین امتیاز راننده (بین ۰ تا ۵)

- ویژگی `totalRides` از نوع int : تعداد سفرهای انجام‌شده توسط راننده
- ویژگی `x` از نوع double : مختصات `X` راننده در نقشه
- ویژگی `y` از نوع double : مختصات `Y` راننده در نقشه

این کلاس شامل متد های زیر است :

```
public double distanceTo(double x2, double y2) {
    //TODO
}

public void moveTo(double x2, double y2) {
    //TODO
}
```

متد `distanceTo` فاصله ی راننده تا مختصات داده شده را حساب کرده و برمیگرداند. متد `moveTo` مختصات راننده را به مختصات داده شده تغییر میدهد. همچنین دارای یک Constructor است که تمامی ویژگی ها را مقدار دهی میکند.

کلاس `RideRequest` :

کلاس `RideRequest` اطلاعات مربوط به یک سفر را ذخیره می‌کند، از جمله موقعیت مبدا و مقصد. همچنین دارای متدی برای محاسبه‌ی مسافت سفر است.

این کلاس شامل ویژگی های زیر است :

- `startX` و `startY` از نوع double که مختصات مبدا سفر را نگه میداند.
- `endX` و `endY` از نوع double که مختصات مقصد سفر را نگه میداند.

این کلاس دارای متد زیر میباشد :

```
public double getTripDistance() {
    //TODO
}
```

این متد فاصله مبدا تا مقصد را حساب میکند و برمیگرداند.

همچنین دارای یک Constructor است که متغیر های `startX` و `startY` و `endX` و `endY` را مقدار دهی میکند.

کلاس `RideManager` :

کلاس `RideManager` وظیفه‌ی انتخاب نزدیک‌ترین راننده‌ی آزاد، محاسبه‌ی هزینه‌ی سفر و تخصیص راننده به سفر را دارد.

این کلاس دارای متد های زیر است :

```
public static void processRideRequest(Driver[] drivers, RideRequ
{
    //TODO
}

public static double calculateFare(RideRequest request, Driver d
{
    //TODO
}
```

متد `processRideRequest` آرایه ای از `Driver` ها و یک `RideRequest` دریافت کرده و نزدیک ترین راننده به مبدا را که فاصله آن کمتر از 20 واحد باشد را پیدا میکند.اگر دو راننده فاصله یکسانی از مبدا داشته باشند, راننده ای که `rating` بالاتری دارد انتخاب میشود.

- اگر راننده ای پیدا شد , نام راننده و قیمت سفر چاپ خواهد شد و یک عدد به تعداد کل سفر های راننده اضافه خواهد شد :

```
{name}:{price}
Ali : 2000
Hossein : 1500
```

- و اگر راننده ای پیدا نشد پیام `No Driver Found` چاپ میشود.

متد `calculateFare` هزینه سفر را با فرمول زیر محاسبه میکند:

هزینه سفر = مسافت * امتیاز راننده * (اگر VIP باشد 2 اگر نباشد 1)

نکته : حداقل هزینه سفر 10 است و اگر هزینه ی حساب شده کمتر از 10 شد باید همان 10 درنظر گرفته شود.

شما باید یک فایل Zip آپلود کنید که وقتی باز میشود فقط شامل 3 فایل `Driver.java` و `RideRequest.java` و `RideManager.java` باشد.

The Chicken Man

- محدودیت زمان:۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: مهدی کریمی

آقای فرینگ به دلیل مشغله زیاد و حساسیت های مدیریتی و نظارتی از شما کمک می خواهد برنامه ای برای مدیریت رستورانش بنویسید تا بتواند با نظم بهتری به کار های مهم ترش رسیدگی کند.

پروژه اولیه را می‌توانید از این لینک دانلود کنید.

جزئیات پروژه:

کلاس Customer

شی ساخته شده از این کلاس یک مشتری را تعریف می کند.

پراپرتی ها:

`name` و `phoneNumber` که به ترتیب نام و شماره تلفن مشتری را ذخیره می کند.

متد ها:

`constructor` : نام و شماره تلفن را مقدار دهی می کند.

`displayCustomerInfo` : اطلاعات مشتری را به فرمت زیر بازمی گرداند:

```
Name: {name}, Phone: {phoneNumber}
```

گتر ها و ستر ها را بسته به نیاز پیاده سازی کنید.

کلاس MenuItem:

پراپرتی ها:

`name` : نام آیتم

`price` : قیمت آیتم

`category` : هر آیتم یکی از موارد Appetizers, Main Course, Side, Drink, Dessert می تواند باشد.

متدها:

`constructor` : سه پراپرتی را مقداردهی می کند.

گترها و سترها را بسته به نیاز پیاده سازی کنید.

کلاس Order:

پراپرتی ها:

`orderID` : هرسفارش یک آیدی دارد که به ترتیب سفارش دادن مشخص میشوند، یعنی orderID سفارش اول برابر 1 و orderID سفارش دوم برابر 2 و ....

`menuItems` : آرایه ای از آیتم های موجود در منو که قرار است سفارش داده شوند.(تضمین می شود هرکس حداکثر 10 آیتم سفارش خواهد داد).

`customer` : شخص سفارش دهنده.

متدها:

`constructor` : پراپرتی ها را مقدار دهی می کند.

`addItemToOrder` : با فراخوانی این متد روی سفارش، آیتم مد نظر به صورت ترتیبی به menuItems اضافه می شود.

`calculatePrice` : قیمت قابل پرداخت برای مشتری را باز می گرداند. درصورتی که قیمت بالای `30$` باشد 15 درصد، و درصورتی که قیمت بالای `20$` باشد 10 درصد تخفیف روی کل فاکتور باید اعمال شود. (به کمک Math.round مقدار نهایی را تا دو رقم اعشار گرد کنید.)

`displayOrder` : مشخصات سفارش را به فرمت زیر باز میگرداند:

```
Order {n}: {item1}(${price1}) - {item2}(${price2}) - item3(${price3})
```

برای مثال:

```
Order 1: Nachos($1.99) - Diet Coke($0.5)
```

گتر ها و ستر ها را بسته به نیاز پیاده سازی کنید.

کلاس Restaurant

پراپرتی ها:

`menu` : آرایه ای از آیتم های موجود در منوی این رستوران(تضمین می شود حداکثر 20 آیتم وجود خواهد داشت).

`orders` : آرایه ای از سفارش ها که مشتری ها ثبت می کنند.(تضمین می شود حداکثر 10 سفارش).

`menuCount` :تعداد آیتم های موجود در منو را ذخیره می کند.

`orderCount` : تعداد سفارش های داده شده در این رستوران را ذخیره می کند.

متد ها:

`addMenuItem` : یک آیتم به منوی این رستوران به صورت ترتیبی اضافه می کند.

`searchByCategory` : محصولات موجود در منوی رستوران برای یک کتگوری خاص را به فرمت زیر نشان می دهد:

```
{Category}: item1(${price1}) - item2(${price2}) - item3(${price3})
```

برای مثال:

```
Main Course: Fried Chicken($5.99) - Chicken Wings($4.99)
```

درصورتی که آیتمی با کتگوری مدنظر در منو موجود نباشد، مقدار `no item for this category!` بازگردانده بشود.

`placeOrder` : سفارش مشتری را در لیست سفارش های این رستوران به صورت ترتیبی اضافه می کند.

گتر ها و ستر ها را بسته به نیاز پیاده سازی کنید.

سنجش درستی:

درصورتی که کد زیر اجرا شود خروجی شما باید تطابق داشته باشد:

```
public class Lospollos {

    public static void main(String[] args) {

        Restaurant restaurant = new Restaurant();

        restaurant.addMenuItem(new MenuItem("Nachos", 1.99, "App
```

```
        restaurant.addMenuItem(new MenuItem("Fried Chicken", 5.9
        restaurant.addMenuItem(new MenuItem("Chicken Wings", 4.9
        restaurant.addMenuItem(new MenuItem("Fries", 2.49, "Side
        restaurant.addMenuItem(new MenuItem("Diet Coke", 0.5, "D
        restaurant.addMenuItem(new MenuItem("Beer", 0.5, "Drink"
        restaurant.addMenuItem(new MenuItem("Ice Cream", 3.49, "
        String category = "Main Course";
        System.out.println(restaurant.searchByCategory(category)
        Order order1 = new Order(new Customer("Jesse Pinkman", "
        order1.addItemToOrder(restaurant.getMenu()[0]);
        order1.addItemToOrder(restaurant.getMenu()[4]);
        restaurant.placeOrder(order1);
        System.out.println(order1.customerInfo());
        System.out.println(order1.displayOrder());
        System.out.println(order1.calculatePrice());
        Order order2 = new Order(new Customer("Jane Smith", "987
        order2.addItemToOrder(restaurant.getMenu()[6]);
        System.out.println(order2.customerInfo());
        System.out.println(order2.displayOrder());
        System.out.println(order2.calculatePrice());
    }
```

```
    }
```

خروجی:

```
Main Course: Fried Chicken($5.99) - Chicken Wings($4.99)
Name: Jesse Pinkman, Phone: 505-555-0199
Order 1: Nachos($1.99) - Diet Coke($0.5)
2.49
Name: Jane Smith, Phone: 987-654-3210
Order 2: Ice Cream($3.49)
3.49
```

آنچه باید ارسال کنید:

ساختار فایل `zip` ارسالی باید به صورت زیر باشد:

```
<zip_file_name.zip>
    ├── Customer.java
    ├── MenuItem.java
    ├── Restaurant.java
    └── Order.java
```

شبکه زیرزمینی

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: امیررضا یزدان پناه

الیوت اندرسون با توجه به اینکه توسط شرکت Evil Corp گروگان گرفته شده است، از شما می خواهد تا برنامه ای بنویسید که بتواند اطلاعات اشخاص خبره، از جمله کارمندان Evil Corp و هکر های مطرح را جمع آوری و مدیریت کند تا اون بتواند از این اطلاعات علیه شرکت Evil Corp برای آزادی خود استفاده کند.

پروژه اولیه را می توانید از این لینک دانلود کنید.

جزئیات پروژه:

کلاس Character

پراپرتی ها:

- `name` (نوع: String): نام شخصیت.

- `age` (نوع: int): سن شخصیت.
- `role` (نوع: String): نقش شخصیت (مانند "Financial Manager"، "Agent"، و غیره).
- یک فیلد استاتیک خصوصی به نام `totalCharacters` برای شمارش تعداد کل شخصیت‌های ایجاد شده.

متد ها:

- `constructor` : فیلدهای `name` ، `age` و `role` را مقداردهی اولیه کرده و مقدار `totalCharacters` را یک واحد افزایش دهید.
- `displayInfo` : اطلاعات (نام، سن، نقش) را به فرمت زیر برمی گرداند:

```
Name: [name]
Age: [age]
Role: [role]
```

- `displayTotalCharacters` : تعداد کل شخصیت‌های ایجاد شده را به فرمت زیر برمی گرداند:

```
Total Characters Created: [totalCharacters]
```

- گتر و ستر ها بر اساس نیاز.

کلاس Hacker (از کلاس Character ارثبری می کند):

پراپرتی ها:

- `expertise` : تخصص هکر را تعریف می کند.

متد ها:

- `constructor` : از `super` برای مقداردهی اولیه فیلدهای ارث‌بری شده استفاده کرده و فیلد `expertise` را نیز مقداردهی می کند.
- `displayHackerInfo` : اطلاعات هکر شامل جزئیات عمومی (از طریق `displayInfo`) و تخصص (expertise) را مطابق زیر برمی گرداند:

```
Hacker Details:
displayInfo()
Expertise: [expertise]
```

- گتر و ستر بر اساس نیاز

کلاس CorporateCharacter (از کلاس Character ارثبری می کند):

پراپرتی ها:

- `position` : موقعیت شغلی کارمند را تعریف می کند.

متد ها:

- `constructor` : از `super` برای مقداردهی اولیه فیلدهای ارث‌بری شده استفاده کرده و فیلد `position` را نیز مقداردهی می کند.
- `displayCorporateInfo` : اطلاعات کارمند شامل جزئیات عمومی (از طریق `displayInfo`) و موقعیت شغلی (position) را مطابق زیر برمی گرداند:

```
Corporate Character Details:
displayInfo()
Position: [position]
```

- گتر و ستر بر اساس نیاز

کلاس UndergroundNetwork

پراپرتی ها:

- `Character` : آرایه ای برای شخصیت های عادی (تضمین می شود حداکثر 10 شخص وجود خواهد داشت).
- `Hacker` : آرایه ای برای هکر ها (تضمین می شود حداکثر 10 شخص وجود خواهد داشت).
- `CorporateCharacter` : آرایه ای برای کارکنان شرکت (تضمین می شود حداکثر 10 شخص وجود خواهد داشت).

- برای هر آرایه یک شمارنده (`charCount` ، `hackerCount` و `corpCount`) داشته باشید.

متد ها:

- `constructor`
- `addCharacter` : اگر فضای کافی وجود داشت، شخصیت را به آرایه اضافه کند و پیام زیر را برگرداند:

```
Added character: [name]
```

در غیر این صورت پیام خطا مطابق زیر برگرداند:

```
Character array is full.
```

- `addHacker` : اگر فضای کافی وجود داشت، شخصیت را به آرایه اضافه کند و پیام زیر را برگرداند:

```
Added hacker: [name]
```

در غیر این صورت پیام خطا مطابق زیر برگرداند:

```
Hacker array is full.
```

- `addCorporateCharacter` : اگر فضای کافی وجود داشت، شخصیت را به آرایه اضافه کند و پیام زیر را برگرداند:

```
Added corporate character: [name]
```

در غیر این صورت پیام خطا مطابق زیر برگرداند:

```
Corporate character array is full.
```

- `displayAllCharacters` : اطلاعات تمامی شخصیت‌ها را از طریق فراخوانی متدهای مربوط به هر کلاس (برای شخصیت‌های عادی، هکرها و کارکنان شرکت) مطابق زیر برگرداند:

```
---- Regular Characters ----
display info of charachter 1
----------------------------
display info of character 2
----------------------------
...
---- Hackers ----
display info of hacker 1
-----------------
display info of hacker 2
-----------------
...
---- Corporate Characters ----
display info of CorCharachter 1
------------------------------
display info of CorCharacter 2
------------------------------
...
```

- `searchCharacter` : شخصیت با نام مشخص را در تمامی آرایه‌ها جستجو کرده و در صورت یافتن، اطلاعات او را برگرداند.

براساس اینکه شخص در کدام آرایه پیدا شد یکی از این سه حالت چاپ شود:

```
Found [regular character / hacker / corporate character]:
display character info
```

و اگر یافت نشد متن زیر را برگرداند:

```
Character not found: [name]
```

- `updateRole` : پراپرتی `role` شخصیت مورد نظر را در آرایه اپدیت کرده و متن زیر را برمی گرداند:

```
Updated role for [name] to: [newRole]
```

و در صورتی که شخص در آرایه ها یافت نشد، خطای زیر برگردد:

```
Update failed. Character not found: [name]
```

- `removeCharacter` : آرایه ها را جستجو می کند و شخص مورد نظر را از آرایه مشخص حذف می کند. با توجه به این که شخص در کدام آرایه بود یکی از پیام های زیر را برمی گرداند:

```
Removed [regular character / hacker / corporate character] [name
```

و در صورتی که شخص یافت نشد خطای زیر برگردانده شود:

```
Character not found: [name]
```

سنجش درستی

در صورتی که کد زیر اجرا شود خروجی شما باید با آن یکسان باشد:

```java
public class Main {
    public static void main(String[] args) {
        UndergroundNetwork network = new UndergroundNetwork();
        Character angela = new Character("Angela Moss", 30, "Fin
        Hacker elliot = new Hacker("Elliot Anderson", 28, "Prota
        CorporateCharacter tyrell = new CorporateCharacter("Tyre
        System.out.println(network.addCharacter(angela));
        System.out.println(network.addHacker(elliot));
        System.out.println(network.addCorporateCharacter(tyrell)
        System.out.println(network.displayAllCharacters());
        System.out.println(network.searchCharacter("Angela Moss"
        System.out.println(network.updateRole("Angela Moss", "Da
        System.out.println(network.removeCharacter("Tyrell Welli
        System.out.println(network.displayAllCharacters());
        System.out.println(Character.displayTotalCharacters());
    }
}
```

خروجی

```
Added character: Angela Moss
Added hacker: Elliot Anderson
Added corporate character: Tyrell Wellick
---- Regular Characters ----
Name: Angela Moss
Age: 30
Role: Financial Manager
----------------------------
---- Hackers ----
Hacker Details:
Name: Elliot Anderson
Age: 28
Role: Protagonist
Expertise: Cybersecurity
-----------------
---- Corporate Characters ----
Corporate Character Details:
Name: Tyrell Wellick
Age: 35
Role: Ambitious Executive
Position: CTO
------------------------------

Found regular character:
Name: Angela Moss
Age: 30
Role: Financial Manager
Updated role for Angela Moss to: Data Analyst
Removed corporate character: Tyrell Wellick
---- Regular Characters ----
Name: Angela Moss
Age: 30
Role: Data Analyst
----------------------------
---- Hackers ----
Hacker Details:
Name: Elliot Anderson
Age: 28
Role: Protagonist
Expertise: Cybersecurity
-----------------
---- Corporate Characters ----
```

```
Total Characters Created: 3
```

آنچه باید اپلود کنید:

یک `zip` با ساختار زیر ارسال کنید:

```
<zip_file_name.zip>
    ├── Character.java
    ├── Hacker.java
    ├── CorporateCharacter.java
    └── UndergroundNetwork.java
```

بوستان

- طراح : آریا زریاب
- سطح : سخت

ظاهرا به گوش اساتید رسیده که سامانه گلستان دچار اختلال شده و خب چون موقع انتخاب واحد شده اساتید ناچارند که این مشکلو برطرف کنند، به همین دلیل گروهی از آنها تصمیم گرفتند سامانه ای تحت عنوان بوستان را راه اندازی کنند.
ولی چون سرشون به دلیل تصحیح برگه ها و برگزاری کنفرانس گرم بود از شما کمک خواستند که بخشی از این سامانه را برایشان پیاده کنید.

جزئیات برنامه

ابتدا پروژه اولیه را این لینک دانلود کنید.

این پروژه یک سیستم مدیریت دانشجو و استاد و کلاس درس را شبیه سازی میکند که در آن دانشجویان میتوانند در درس دلخواه خود ثبت نام کنند.

▼ ساختار فایل پروژه

```
<zip_file_name.zip>
      ├── Course.java
      ├── Student.java
      ├── Professor.java
      ├── University.java
      ├── InPersonCourse.java
      ├── OnlineCourse.java
      └── HybridCourse.java
```

کلاس `Course`

ویژگی ها:

`courseID` و `title` از جنس استرینگ.

`maxStudent` و `enrolledStudent` از جنس int.

آرایه ای از دانشجو ها (Student).

یک استاد (Professor).

تمام سطوح دسترسی protected میباشند.

متد ها و کانستراکتور :

۱. کانستراکتور :

```
1 | public Course(String courseID, String title, int maxStudent
```

۲. متد `enrollStudent` : یک دانشجو دریافت میکند و او را در کلاس ثبت نام میکند.

اگر ظرفیت کلاس پر بود پیام زیر :

`Course is full:` `course title`

و در غیر این صورت پیام زیر چاپ میشود(البته اگر دانشجو نیز شرایطش را داشته باشد).

`student name` `enrolled in` `course title`

۳. متد `removeStudent` : یک دانشجو دریافت میکند و او را از کلاس حذف میکند.

اگر با موفقیت از کلاس حذف شود پیام زیر :

`student name` `dropped` `course title`

در غیر این صورت پیام زیر چاپ میشود :

`There is no student with this name:` `student name`

۴. متد `calculateFee` : شهریه ی کلاس را به صورت `double` برمیگرداند،

که به صورت دیفالت ۱۰۰۰ میباشد.

۵. متد `assignProfessor` : استاد کلاس را تعیین میکند.

۶. متد `removeProfessor` : استاد کلاس را حذف میکند.

۷. گتر ها.

کلاس `Student`

ویژگی ها :

`studentID` و `name` از جنس استرینگ.

آرایه ای از درس ها (Course) که به صورت دیفالت دارای ظرفیت ۵ میباشد.

آرایه ای از نمره ها (int).

`courseCount` به صورت `int` .

تمام سطوح دسترسی private میباشند.

کانستراکتور و متد ها :

۱. کانستراکتور :

نمره ی درس ها را به صورت پیشفرض `-1` ست کنید(یعنی هنوز نمره ای داده نشده است).

```
1 | public Student(String studentID, String name)
```

۲. متد `enrollInCourse` : یک درس را ورودی میگیرد و دانشجو در آن درس ثبت نام میکند.

اگر ترم هنوز شروع نشده بود فالس ریترن و پیغام زیر چاپ میشود :

```
Action denied! Semester has not started.
```

و اگر دانشجو به حداکثر درس قابل ثبت نام رسیده باشد فالس ریترن و پیغام زیر چاپ میشود :

```
Cannot enroll, maximum courses reached!
```

و در صورت موفقیت مقدار ترو برگردانده میشود.

۳. متد `dropCourse` : درس ورودی گرفته شده را حذف میکند.

اگر با موفقیت انجام شود true ریترن و پیام زیر چاپ میشود :

`student name` dropped `course title`

در غیر این صورت فالس ریترن و پیام زیر چاپ میشود :

You are not enrolled in `course title`

۴. متد `getTotalFee` : جمع کل شهریه های دانشجو را به صورت `double` برمیگرداند.

۵. متد `calculateGPA` : معدل دانشجو را به صورت `double` برمیگرداند.

۶. متد `rateProfessor` :

اگر ترم هنوز شروع نشده بود فالس ریترن و پیغام زیر چاپ میشود :

Action denied! Semester has not started.

اگر نمره ای بین ۱ تا ۵ داده شود پیغام زیر :

`student name` rated `professor name` with `rating`

در غیر این صورت پیام زیر چاپ میشود.

Invalid rating! Please rate between 1-5.

۷. متد `updateGrade` : نمره ی دانشجو آپدیت میشود.

در صورت موفقیت پیام زیر :

`student name` received grade `grade` in `course title`

و در غیر این صورت :

`student name` is not enrolled in `course title`

۸. متد `isEnrolledIn` : چک میکند که در درس ثبت نام شده است یا خیر و `boolean` برمیگرداند.

۹. متد `clearData` : تمام دیتای دانشجو را پاک میکند.

۱۰. گتر ها.

کلاس Professor

ویژگی ها :

`professorID` و `name` از جنس استرینگ.

آرایه از امتیاز ها (int).

آرایه از درس ها (Course).

`ratingCount` و `courseCount` از جنس `int` . به طور پیشفرض ظرفیت ۱۰۰ نمره و ۱۰ کلاس درس را دارند.

تمام سطوح دسترسی private میباشند.

کانستراکتور و متد ها :

۱. کانستراکتور :

```
1 | public Professor(String professorID, String name)
```

۲. متد `addCourse` : یک درس به درس های استاد اضافه میکند.

اگر ترم هنوز شروع نشده بود فالس ریترن و پیغام زیر چاپ میشود :

```
Action denied! Semester has not started.
```

اگر از ظرفیت مجاز بخواهد درس بیشتری داشته باشد پیام زیر :

```
You can't be tutor of more than 10 courses.
```

۳. متد `removeCourse` : درس ورودی گرفته شده را حذف میکند .

اگر با موفقیت درس حذف شود پیام زیر :

`professor name` stopped teaching `course title`

و در غیر این صورت پیام زیر چاپ میشود.

`professor name` is not teaching `course title`

۴. متد `removeStudent` :

اگر دانشجو در هیچدام از درس های استاد نباشد پیام زیر چاپ میشود :

Student `student name` is not in any of `professor name`'s courses.

در غیر این صورت از تمام درس های استاد حذف میشود.

۵. متد `receiveRating` : یک امتیاز برای استاد ثبت میکند.

۶. متد `getAverageRating` : میانگین امتیاز استاد را به صورت `double` ریترن میکند.

۷. متد `teaches` : چک میکند که آیا استاد درس را ارائه میدهد یا نه.

۸. متد `assignGrade` : یک نمره به دانشجو میدهد.

اگر استاد درس را نداشته باشد پیام زیر :

`professor name` cannot grade `student name` in `course title` (not teach

اگر دانجشو درس را نداشته باشد پیام زیر :

`student name` is not enrolled in `course title`, so grade cannot be ass

۹. متد `clearCourses` : فقط دیتا های مربوط به کلاس های درس استاد پاک میشوند.

۱۰. گتر ها.

کلاس `OnlineCourse`

فرزند یا زیر کلاس `Course` است.

ویژگی ها :

`platform` از جنس استرینگ.

کانستراکتور و متد ها :

۱. کانستراکتور :

```
public OnlineCourse(String courseID, String title, int maxS
```

۲. متد `calculateFee` : که شهریه ی کلاس را ست میکند. و `0.8` برابر مقدار والدش میباشد.

کلاس InPersonCourse

فرزند یا زیر کلاس `Course` است.

ویژگی ها :

`roomNumber` از جنس استرینگ.

کانستراکتور و متد ها :

۱. کانستراکتور :

```
public InPersonCourse(String courseID, String title, int ma
```

۲. متد `calculateFee` : که شهریه ی کلاس را ست میکند. و `200` واحد از مقدار والدش بیشتر است.

کلاس HybridCourse

فرزند یا زیر کلاس `Course` است.

ویژگی ها :

`weekSchedule` از جنس استرینگ.

کانستراکتور و متد ها :

۱. کانستراکتور :

```
public HybridCourse(String courseID, String title, int maxS
```

۲. متد `calculateFee` : که شهریه ی کلاس را ست میکند.

```
public double calculateFee() {
return (new OnlineCourse(courseID, title, maxStudents, "Hyb
}
```

کلاس University

ویژگی ها :

`semesterStarted` به صورت `boolean` و `static` .

آرایه هایی برای نگه داری استاد ها ، کلاس ها و دانشجو ها.

سطوح دسترسی به صورت private میباشند.

کانستراکتور و متد ها :

۱. کانستراکتور :

```
public University(int maxProfessors, int maxCourses, int ma
```

که در واقع آرایه ها را `initialize` میکند.

۲. متد `startSemester` : ترم شروع میشود.

۳. متد `endSemester` : ترم تمام میشود و تمام اطلاعات استاد ها و دانشجو ها مربوط به این ترم از بین میرود(از متد های clear استفاده کنید).

۴. متد `addProfessor` : اگر ترم شروع شده بود و ظرفیت بود استاد را اضافه میکند.

اگر ترم شروع نشده بود پیام زیر :

```
Action denied! Semester has not started.
```

و اگر ظرفیت اساتید به اتمام رسیده بود پیام زیر چاپ شود:

```
University has reached the maximum number of professors.
```

۵. متد `addStudent` : اگر ترم شروع شده بود و ظرفیت بود دانشجو را اضافه میکند.
اگر ترم شروع نشده بود پیام زیر :

```
Action denied! Semester has not started.
```

و اگر ظرفیت دانشجو ها به اتمام رسیده بود پیام زیر چاپ شود:

```
University has reached the maximum number of students.
```

۶. متد `addCourse` : اگر ترم شروع شده بود و ظرفیت بود درس را اضافه میکند. اگر ترم شروع نشده بود پیام زیر :

```
Action denied! Semester has not started.
```

و اگر ظرفیت درس ها به اتمام رسیده بود پیام زیر چاپ شود:

```
University has reached the maximum number of courses.
```

۷. متد `removeProfessorFromCourse` : استاد و درس را ورودی میگیرد و اگر ترم شروع شده بود استاد را از آن درس حذف میکند. در غیر این صورت پیام زیر چاپ میشود :

```
Action denied! Semester has not started.
```

۸. متد `addProfessorToCourse` : استاد و درس را ورودی میگیرد و اگر ترم شروع شده بود استاد را برای آن درس ست میکند. در غیر این صورت پیام زیر چاپ میشود:

```
Action denied! Semester has not started.
```

۹. متد `isSemesterStarted` : اگر ترم شروع شده باشد ترو برمیگرداند.

آنچه باید آپلود کنید

ساختار فایل `zip` ارسالی باید به صورت زیر باشد:

```
<zip_file_name.zip>
      ├── Course.java
      ├── Student.java
      ├── Professor.java
      ├── University.java
      ├── InPersonCourse.java
      ├── OnlineCourse.java
      └── HybridCourse.java
```

Risk (امتیازی)

- سطح سوال : سخت
- طراحان : آریا شاکو و برنا ماهرانی

ابتدا پروژه اولیه را از این لینک دانلود کنید.

ریسک یک بازی معروف است که برای آشنایی بیشتر از ان میتوانید به لینک مراجعه کنید

https://en.wikipedia.org/wiki/Risk_(game)

قرار است اینجا این بازی را پیاده سازی کنیم.

کلاس Country

این کلاس دو فیلد دارد

```
public class Country{
    String owner;
    int troops;
    public String getOwner{
        \\TODO
    }
    public int getTroops{
        \\TODO
    }
    public void setOwner(String owner){
        \\TODO
    }
    public void setTroops(int troops){
        \\TODO
    }
}
```

کلاس Player

این کلاس 5 فیلد دارد ستر و گتر های مناسب برای این فیلد هارا پیاده سازی کنید. بولین strike مشخص میکند که پلیر در حالت attack هست یا defend برای ستر گتر ها اولین اسم متغیر را بزرگ بذارید به طور مثال setStrike

```
public class Player{
    boolean strike;
    int dice;
    boolean tag;
    String name;
    Country[] countries;
}
```

کلاس Risk

کلاس ریسک این کلاس مهم ترین کلاس این سوال است در این کلاس 1 تابع داریم که منطق بازی را پیاده سازی میکنیم. و یک فیلد 2 تایی از 2 تا بازیکن داریم تا با انها بتوانیم بازی کنیم .

منطق بازی اینطور است که پس از اعلام حمله و دفاع باید بازی شروع شود که این بولین در کلاس پلیر وجود دارد. اگر کشوری به کشور دیگر حمله میکند باید تعداد نیرو ها از یکدیگر کم شود و اگر اختلاف تعداد کشور های بازیکنی از بازیکن دیگر بزرگتر مساوی 2 باشد بازی تمام می‌شود.

تاس را در یک حلقه مقدار دهی کنید تا عددی از 0 تا 2 باشد و طبق ان کشور یک بازیکن رندوم انتخاب میشود. برای هر بازیکن 3 تا کشور در نظر بگیرید و به ترتیب به هر کشور 5 10 15 تا نیرو بدهید.

حالا که وارد حلقه بازی شدیم با توجه به کشور انتخابی باید تعداد نیرو ها از هم کم شود و توجه داریم که اگر کشوری از یک بازیکن نیروی بیشتری از کشور بازیکن دیگر دارد کشور با نیروی ماکسیمم آن یکی کشور را تصاحب میکند و به بازیکن داده می‌شود.

در همین حلقه بازی بولین strike باید برای بازیکن اول به attack ست شود و بازیکن دوم به defense ست شود و در دور بعدی تغییر میکند و بازیکن دوم حمله می‌کند و بازیکن اول دفاع می‌کند.

و در نهایت تابعی تعریف کنید که تعداد سربازان باقی مانده بازیکنان رو می‌شمارد اگر بازیکنی در انتهای بازی بیشتر مساوی 20 سرباز داشت بولین تگ برای آن بازیکن true می‌شود. پس از اتمام بازی یک بولین quit فعال شود. در تابع initializePlayers بازیکن هارا بسازید و تابع را اول تابع game فراخوانی کنید. در تابع

conduct attack باید عملیات حمله آن طور که گفته شد پیاده سازی کنید. باقی توابع با توجه به اسم و مطابق تعاریف گفته شده پیاده سازی شوند.

```
public class Risk {
    private Player[] players = new Player[2];

    public void game() {
        \\TO DO
    }

    public Player[] getPlayers(){
        \\ TO DO
    }

    public void initializePlayers() {
        \\TO DO
    }

    public Country createCountry(String owner, int troops) {
        \\TO DO
    }

    public void conductAttack(Player attacker, Player defender,
        \\TO DO
    }

    public void checkAndSetTag(Player player) {
        \\ TO DO
    }

    public int countTotalTroops(Player player) {
        \\TO DO
    }

    public boolean checkWinCondition() {
       \\ TO DO
    }

    public int getCountryCount(Player player) {
        \\TO DO
```

```
        }
    }
```

آنچه باید آپلود کنید

یک فایل zip حاوی 3 کلاس Player , Country , Risk