نمره ها

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح : ساده
- طراح : آریا زریاب

در این سوال شما باید یک سیستم ساده مدیریت نمرات دانشجویان را پیاده‌سازی کنید. داده‌های نمرات در یک فایل متنی ذخیره می‌شوند که هر خط نمایانگر یک رکورد نمره مربوط به یک دانشجو در یک درس خاص است. هر خط در فایل به شکل زیر است و با کاما (,) از هم جدا شده است:

> StudentID,StudentName,CourseName,Grade

مثلا یک فایل میتواند به شکل زیر باشد :

```
1002,Bob,AP,90
1001,Alice,DLC,78
1002,Bob,CA,50
1003,John,FLA,81
```

فایل اولیه پروژه را می توانید از اینجا دانلود کنید.

کلاس GradeManager

شامل یک ویژگی `file` که از جنس `File` میباشد و `private` است.

کانستراکتور این کلاس اینگونه تعریف میشود و یک رشته شامل آدرس فایل را ورودی میگیرد :

```
1 public GradeManager(String filePath) {
2
3 }
```

متد ها :

1. updateGrade :

شماره دانشجویی و درس مربوطه و نمره جدید را ورودی میگیرد و به شرط وجود آن دانش آموز در آن درس و معتبر بودن نمره آن فایل را آپدیت میکند و true را برمیگرداند.

در غیر این صورت باید false برگردانده شود.

```
1 public boolean updateGrade(String studentID, String course, int
2
3 }
```

2. deleteEntry :

شماره دانشجویی و درس مربوطه را ورودی میگیرد و در صورت وجود ، آن اطلاعات (سطر) را پاک میکند.

در غیر این صورت false ریترن داده میشود.

```
1 public boolean deleteEntry(String studentID, String course) {
2
3 }
```

3. calculateStudentAverage :

شماره دانشجویی را ورودی میگیرد و معدل آن دانشجو را حساب میکند.

(اگر دانشجویی با شماره دانشجویی ورودی وجود نداشت معدلش را صفر فرض کنید)

```
1 public double calculateStudentAverage(String studentId) {
2
3 }
```

4. calculateCourseAverage :

یک درس را ورودی میگیرد و میانگین نمرات دانشجویان آن درس را حساب میکند.

(اگر آن درس وجود نداشت میانگین نمرات آن را صفر فرض کنید)

```
1 public double calculateCourseAverage(String course) {
2
3 }
```

5. addEntry :

تمام اطلاعات یک سطر را ورودی میگیرد و آن را به آخر فایل اضافه میکند.

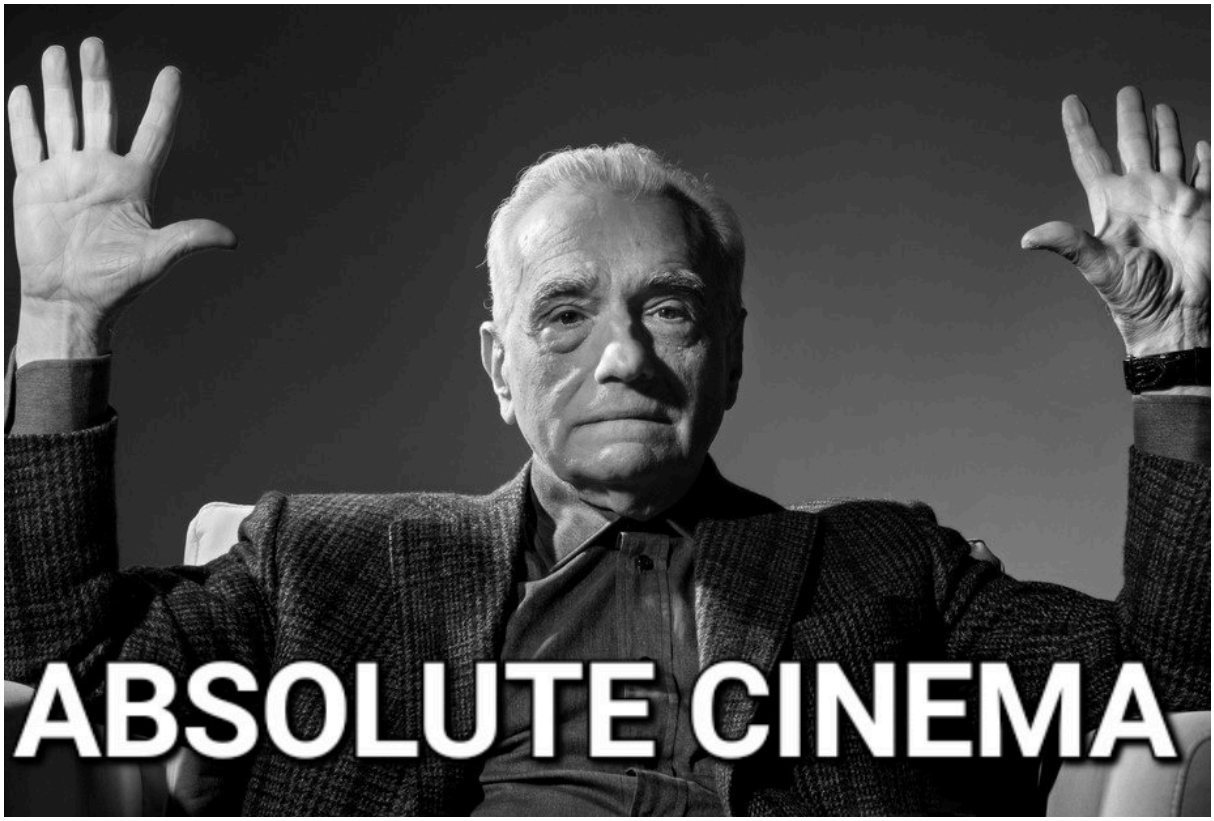
```
1 public void addEntry(String studentId, String name, String cours
2
3 }
```

نکته : در تمام متد ها فرض میشود که داده تکراری نداریم و داده تکراری اضافه نمیشود .

همچنین نمره ها همگی صفر تا ۱۰۰ میباشند. (باید باشند.)

imdb

- محدودیت زمان:۴ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: مهدی کریمی



در این سوال شما به عنوان یک علاقه مند به صنعت سینما، مشتاقانه برنامه ای برای بررسی 250 فیلم برتر imdb خواهید نوشت که بتواند به سوالاتی که ذهن افراد را به خود مشغول کرده پاسخ دهد.

سوالاتی که ذهن افراد را به خود مشغول خواهد کرد عبارتند از:

۱. بدست آوردن اطلاعات کامل مرتبط با یک فیلم مشخص.
۲. بدست آوردن میانگین فروش(box_office) یک ژانر مشخص.
۳. بدست آوردن مدت زمان (run_time) یک فیلم مشخص.
۴. بدست آوردن 3 تا از بهترین فیلم های یک کارگردان مشخص.

فایل csv مربوط به فیلم ها را می توانید از اینجا دانلود کنید.

فایل اولیه پروژه را می توانید از اینجا دانلود کنید.

جزئیات توابع:

متد 1:

```
 public String getMovieInfo(String movieName){
        // TODO
}
```

این تابع با دریافت نام یک فیلم، تمامی جزئیات مربوط به آن فیلم را که در فایل موجود است به فرمت زیر باز می گرداند:

```
rank: %s, rating: %s, genre: %s, run_time: %s, box_office: %s, directors: %s
```

متد 2:

```
public String getAverageBoxOfficeByGenre(String genreToFind) {
        // TODO
}
```

این تابع با دریافت یک ژانر مشخص، میانگین فروش آن ژانر را بدست می آورد. و مقدار بازگشتی آن به این صورت است:

```
average box office for genre 'genreToFind': $averageBoxOffice
```

نکته 1: ممکن است مقدار box_office برای برخی فیلم ها موجود نباشد، در این صورت آن ها را نادیده گرفته و در میانگین گیری حساب نمی کنیم.

نکته 2: برای محاسبه میانگین از نوع داده `long` استفاده کنید.

نکته 3: دقت کنید که یک فیلم می تواند چند ژانر داشته باشد.

متد 3:

```java
public String getMovieDuration(String movieName) {
        // TODO
    }
```

این تابع مدت زمان یک فیلم مشخص را پیدا می کند. و مقدار بازگشتی آن بدین صورت است:

```
run time of 'movieName': run_time
```

متد 4:

```java
public String findTop3MoviesByDirector(String director){
        // TODO
}
```

و بالاخره این تابع 3 تا از بهترین فیلم های یک کارگردان مشخص را به فرمت زیر باز می گرداند:

```
Top movies by {director}:
1. '{movieName}' with rating {rating}
2. '{movieName}' with rating {rating}
3. '{movieName}' with rating {rating}
```

نکته: ممکن است به جای 3 تا فیلم، تنها 1 یا 2 فیلم موجود باشد.

توجه:

- تضمین می شود ورودی متد ها صحیح و دقیقا مطابق فایل داده ها است.
- به علامت های ' در مقدار بازگشتی توابع دقت کنید.

مثال 1:

درصورتی که کد زیر اجرا شود:

```java
public static void main(String[] args) {

        String filePath = "imdb.csv";

        MovieDatabase db = new MovieDatabase(filePath);
```

```
6
7           try {
8
9               System.out.println(db.getMovieInfo("Pulp Fiction"));
10
11              System.out.println(db.getAverageBoxOfficeByGenre("Bi
12
13              System.out.println(db.getMovieDuration("Reservoir Do
14
15              System.out.println(db.findTop3MoviesByDirector("Quen
16
17          } catch (IOException e) {
18
19              e.printStackTrace();
20
21          }
22
23      }
```

خروجی به صورت زیر است:

```
rank: 8, rating: 8.9, genre: Crime_Drama, run_time: 2h 34m, box_offic
average box office for genre 'Biography': $152440865
run time of 'Reservoir Dogs': 1h 39m
Top movies by Quentin Tarantino:
1. 'Pulp Fiction' with rating 8.9
2. 'Django Unchained' with rating 8.4
3. 'Inglourious Basterds' with rating 8.3
```

مثال 2:

درصورتی که کد زیر اجرا شود:

```
    public static void main(String[] args) {

            String filePath = "imdb.csv";

            MovieDatabase db = new MovieDatabase(filePath);
```

```
        try {
            System.out.println(db.getMovieInfo("The Dark Knight"
            System.out.println(db.getAverageBoxOfficeByGenre("Th
            System.out.println(db.getMovieDuration("Interstellar
            System.out.println(db.findTop3MoviesByDirector("Chri
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

خروجی به صورت زیر است:

```
rank: 3, rating: 9.0, genre: Action_Crime_Drama, run_time: 2h 32m, bc
average box office for genre 'Thriller': $160172608
run time of 'Interstellar': 2h 49m
Top movies by Christopher Nolan:
1. 'The Dark Knight' with rating 9.0
2. 'Inception' with rating 8.8
3. 'Interstellar' with rating 8.6
```

آنچه باید آپلود کنید:

فایل MovieDatabase.java که حاوی توابع پیاده سازی شده است را آپلود کنید.

مجموع اعداد اول

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: احسان حبیب آگهی

در این تمرین، شما باید بر روی پروژه‌ای خاص کار کنید. در این پروژه، شما باید یک عدد در یک خط از ورودی بگیرید و مجموع اعداد اول تا عدد داده شده را محاسبه کنید. اما با یک twist! این کار باید با استفاده از multi-thread انجام شود. چرا که شما باید نشان دهید که نه تنها ریاضیات را می‌دانید، بلکه می‌توانید از قدرت پردازش موازی هم بهره‌برداری کنید!

> پروژه اولیه رو از این لینک دانلود کنین!

طرح ایده:

برای این کار تنها کافیست کار را به چند بخش **مساوی** تقسیم کنید و هر بخش را به یک ترد بسپارید. در نهایت منتظر بمانید تا همه ترد ها کارشان تمام شود و مجموع بدست آمده از هر ترد را با هم مجددا جمع و چاپ کنید. ما در این مثال به طور ثابت از 5 ترد همزمان استفاده می کنیم

جزئیات قسمت های کد:

متغیر ها

شروع بازه را ۲ (اولین عدد اول) قرار دهید. انتهای بازه از ورودی گرفته می شود و در متغیر `limit` قرار می گیرد. `segmentSize` را به عنوان طول بازه‌ای که هر ترد باید **به طور مساوی** پردازش کند، مشخص کنید. مقدار اولیه‌ی `end` با `segmentSize` برابر است، چرا که پایان بازه‌ی ترد اول برابر با همین مقدار در نظر گرفته می‌شود.

```
int segmentSize;
int start;
int end;
```

کلاس PrimeSumTask

در این کلاس، کد لازم برای اجرای هر ترد از قبل نوشته شده است.

لازم است کلاس `PrimeSumTask` اینترفیس مناسب برای پاس داده شدن به ترد را `implement` کند

```
static class PrimeSumTask implements //TODO
{
...
}
```

توابع

- **`getSum()`:** مقدار `sum` را بازگردانی کنید
- **`run()`:** محاسبات لازم برای جمع اعداد اول داخل بازه را انجام دهید
- **`isPrime()`:** اگر عدد اول باشد `true` و در غیر این صورت `false` بازگردانی می شود.

مقداردهی ترد ها

آرایه `threads` برای نگه داشتن لیست ترد های در حال اجرا و `tasks` آرایه ای از تسک هاست که متناظرا به ترد ها محول می شود. در کد زیر `task[i]` را به یک ترد از آرایه `threads` اختصاص دهید و ترد را شروع کنید

```
//Here we start multi-thread tasks
for (int i = 0; i < NUM_THREADS; i++) {
// Last thread takes care of remaining numbers
if (i == NUM_THREADS - 1) {
end = limit;
}
tasks[i] = new PrimeSumTask(start, end);
//TODO: Give the task to a thread and start
start = end + 1;    end += segmentSize;}
```

حاصل جمع نهایی

در نهایت باید منتظر بمانید تا کار **تمام** ترد ها تمام شود و نتیجه نهایی چاپ شود. برای اینکه از اتمام کار ترد مطمئن شوید پیشنهاد میشود از `join()` استفاده کنید. درباره متد join بیشتر بخونین

```
long sum = 0;
for (int i = 0; i < NUM_THREADS; i++) {
    //TODO: wait for all threads to die
    }
```

اختیاری:

میتونید برای بررسی این که تا چه میزان محاسبات شما با مالتی ترد شدن برنامه سریعتر شده بجای خط `threads[i].start();` از `javathreads[i].run();` استفاده کنید! تفاوت در مقیاس بزرگ (مثلا ورودی 10 میلیون) به طور چشمگیری زیاد هست! صدا زدن `run()` به جای `start()` برای استارت ترد یک خطای رایج هست! این کار به اشتباه در ازای شروع عملیات محول شده در یک ترد جدید آن را در ترد فعلی اجرا خواهد کرد. بیشتر بخونید

ورودی:

تنها یک خط که مجموع اعداد اول تا آن عدد را باید محاسبه کنید

خروجی:

یک خط که نتیجه نهایی باید چاپ شود

مثال:

ورودی نمونه ۱

```
5000000
```

خروجی نمونه ۱

```
838596693108
```

نکته

> فرایند حل مسئله را حتما از طریق کد ناقص داده شده دنبال کنید و پیاده سازی نهایی حتما باید به شکل Multi-thread باشد. به ارسال هایی که به صورت مالتی ترد نباشند نمره تعلق نخواهد گرفت

گزارش متن

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: امیر محمد گنجی زاده

یک فایل با محتوای رندوم بدست شما رسیده. ما از شما میخوایم که یک گزارش کلی از محتوای فایل درست کنید. شما باید برای هر خط از متن فایل یک گزارش بنویسید. به این صورت :

```
[Vowels: V, Consonants: C] Line
```

که V تعداد vowel ها و C تعداد consonant ها در اون خط بوده و Line نیز ورژن مورد علاقه ما از اون خطه.

ورژن مورد علاقه ما چیه؟ ما اصلا دوس نداریم حرف consonant ای رو ببینیم که lowercase باشه. پس همه اینا رو به uppercase اشون تبدیل کنید.

در اخر هم یه گزارش کلی بدین و تموم. به این صورت:

```
---
Total Vowels: TV
Total Consonants: TC
```

که TV تعداد کل vowel ها و TC تعداد کل consonant ها در فایل است.

یه نکته دیگه هم هست. یسری از خطوط با // شروع میشن که خب اطلاعات بدرد بخوری ندارن. پس این خط ها رو گزارش ندید. راستی ما از کاراکتر # هم خیلی بدمون میاد. ولی چون نمیتونیم تغییرش بدیم کلا خط هایی که شامل این کاراکتر هستن رو هم نمیخوایم.

مثال

محتوای فایل :

```
// This is a comment :)
Hello World!
Programming is fun.
Line with #
```

گزارش شما :

```
[Vowels: 3, Consonants: 7] HeLLo WoRLD!
[Vowels: 5, Consonants: 11] PRoGRaMMiNG iS FuN.
---
Total Vowels: 8
Total Consonants: 18
```

یک کلاس به اسم **FileReport** بسازید که حاوی یک متد به نام **report** بوده.

```
public String report(String path)
```

این متد آدرس **مطلق** فایل رو ورودی میگیره و گزارش نهایی رو برمیگردونه.

```
1 public class FileReport {
2     public String report(String path) {
3         // TODO
4     }
5 }
```

متد رو کامل کنید و فایل کلاس رو آپلود کنید.

اطلاعات ناجور!

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- طراح: سید محمد حسینی
- سطح: سخت

امین که به‌تازگی مدیر یک گالری ماشین (اتوگالری) شده‌است، متوجه می‌شود که اطلاعات موجود برای مدیریت ناقص است و گم می‌شوند. او می‌خواهد فایل‌هایی آماده کند تا سریع‌تر اطلاعات را ذخیره، بازیابی و ادیت کند. به او کمک کنید و برنامه‌ای برای او بنویسید تا اینکار را برای او انجام دهد. او از مدیر قبلی صرفا یک فایل دارد که در آن تمام اطلاعات مربوطه درج شده اما نیاز است که سازمان‌دهی شود.

پروژه اولیه را می توانید از اینجا دانلود کنید این لینک

توجه:تنها الگوی موجود در فایل اولیه این است که در فایل، اطلاعات ماشین ها با C و اطلاعات کارمندان با E شروع می‌شوند.

توجه: اطلاعات کارمندان به صورت: `E*Employee name*position*salary*years of experience` یا `E/Employee name/position/salary/years of experience` ذخیره می‌شود. اگر اطلاعات کارمندی با `*` از هم جدا باشد به این معنی است که آن کارمند استخدام شده‌است و اگر اطلاعات کارمند به صورت `/` از هم جدا شده باشد به این معنی است که آن شخص درخواست رزومه خود را ارسال کرده‌است اما کارمند این مرکز نیست. توجه داشته باشید که میزان درآمد ذخیره شده در فایل‌ها به صورت عدد `int` ذخیره می‌شود و مرتبه آن $1000 است.

جایگاه کارمندان:

- `M` : یعنی اطلاعات مدیر یک بخش است.
- `W` : یعنی اطلاعات یک کارمند ساده است.
- `S` : یعنی اطلاعات یک Supervisor است.

توجه:برای اتومبیل‌ها اطلاعات به صورت `C-Car name-Year of manufacture-Price-Years in race` یا `C|Car name|Year of manufacture|Price|Years in race` تعریف شده‌است. اگر اطلاعات یک ماشین با `|` از هم جدا شده‌باشد به این معنی است که آن ماشین مدرن(جدید) است و اگر اطلاعات یک ماشین با `-` از هم جدا شده‌باشد به این معنی است که آن ماشین قدیمی است. همچنین قابل ذکر است که قیمت اتومبیل‌ها به صورت `int` ذخیره می‌شود و مرتبه آن $1000 است.

متدها:

```
public void showOldCars(File file,String baseAddress)
```

در این متد، اطلاعات ماشین‌های قدیمی به ترتیب حروف الفبا در یک فایل به نام `ShowOldCars.txt` ذخیره می‌شود. یک نمونه خروجی ذخیره شده در این فایل می‌تواند به شکل زیر باشد :

```
Name: Aston Martin
Year: 2005
Cost(*1000): 20
Years in race: 2
----------
Name: BMW
Year: 2004
Cost(*1000): 36
Years in race: 2
----------
Name: Genesis
Year: 2001
Cost(*1000): 13
Years in race: 4
----------
Name: Peykan
Year: 1982
Cost(*1000): 85
Years in race: 10
----------
```

```
public void showNewCars(File file,String baseAddress)
```

این متد اطلاعات ماشین‌های مدرن را به ترتیب حروف الفبا در فایل `ShowModernCars.txt` ذخیره می‌کند. خروجی نمونه از فایل مدنظر می‌تواند به صورت زیر باشد:

```
Name: Benz
Year: 2023
Cost(*1000): 68
Years in race: 8
----------
Name: Optima
Year: 2017
Cost(*1000): 11
Years in race: 0
----------
```

```
public void showEmployees(File file,String baseAddress)
```

این متد وظیفه دارد که اطلاعات **تمام** کارمندان اعم از کسانی که استخدام شده‌اند یا کسانی که صرفا رزومه ارسال کرده‌اند و می‌خواهند استخدام شوند را در یک فایل جدید به نام `ShowEmployees.txt` ذخیره می‌کند. این متد اطلاعات را به این صورت ذخیره می‌کند: اول تمامی کارمندانی که استخدام شده‌اند، اطلاعاتشان به ترتیب حروف الفبا در فایل ذخیره می‌شود و سپس کسانی که رزومه ارسال کرده‌اند اطلاعاتشان به ترتیب حروف الفبا ذخیره می‌شود. مثال : اگر حسن و محمد استخدام شده باشند و علی هنوز استخدام نشده باشد اطلاعات ذخیره شده در فایل `ShowEmployees.txt` می‌تواند به صورت زیر باشد :

```
Name: Hasan
Position: M
Salary(*1000): 40
Years of experience: 10
----------
Name: Mohammad
Position: M
Salary(*1000): 20
Years of experience: 13
----------
Name: Ali
Position: S
Salary(*1000): 24

```

```
Years of experience: 8
----------
```

در این اطلاعات علی از نظر حروف الفبایی (به زبان انگلیسی) باید بالاتر از حسن و محمد قرار بگیرد اما چون هنوز استخدام نشده‌است، اطلاعات او بعد از همه کارمندانی که استخدام شده‌اند ذخیره شده‌است.

```
public int showValue(File file,String baseAddress)
```

این متد ارزش این اتوگالری را بر می‌گرداند. *توجه*: دقت شود که ارزش هر اتومبیل در فایل به صورت قیمت واقعی آن اتومبیل تقسیم بر 1000 ذخیر می‌شود بنابراین برای نشان دادن این ارزش مرتبه 1000 دلار را هم در نظر بگیرید.

```
public String bestCar(File file,String baseAddress)
```

امین علاقه زیادی به شرکت در مسابقات دارد. او هرازگاهی تصمیم میگیرد در مسابقه‌ای شرکت کند. این متد از فایل ذخیره شده ابتدایی اطلاعات اتومبیل با بیشترین تجربه مسابقات را برمی‌گرداند. خروجی برگردانده شده از طرف این متد داخل ترمینال به صورت زیر می‌تواند باشد :

```
Name: Peykan
Year: 1982
Cost(*1000): 85
Years in race: 10
```

مثال :

اگر ورودی فایل `info.txt` به صورت زیر باشد:

```
C-BMW-2015-20-2
E*Mohammad*M*20*13*
E/Ali/S/24/8
C|Benz|2007|4|8
C|Fidelity|2022|30|0
C-Genesis-2001-13-4
C-Peykan-1982-85-10
```

```
C-Aston Martin-2011-20-2
E*Hasan*M*40*10
E/Bahador/S/24/8
E*Zahra*M*40*10
```

اطلاعات ذخیره‌شده در فایل `ShowOldCars.txt` به صورت زیر می‌باشد:

```
Name: Aston Martin
Year: 2011
Cost(*1000): 20
Years in race: 2
----------
Name: BMW
Year: 2015
Cost(*1000): 20
Years in race: 2
----------
Name: Genesis
Year: 2001
Cost(*1000): 13
Years in race: 4
----------
Name: Peykan
Year: 1982
Cost(*1000): 85
Years in race: 10
----------
```

و اطلاعات فایل `ShowModernCars.txt` به صورت زیر می‌باشد:

```
Name: Benz
Year: 2007
Cost(*1000): 4
Years in race: 8
----------
Name: Fidelity
Year: 2022
Cost(*1000): 30
Years in race: 0
----------
```

و اطلاعات ذخیره‌شده در فایل `ShowEmployees.txt` به صورت زیر می‌باشد:

```
Name: Hasan
Position: M
Salary(*1000): 40
Years of experience: 10
----------
Name: Mohammad
Position: M
Salary(*1000): 20
Years of experience: 13
----------
Name: Zahra
Position: M
Salary(*1000): 40
Years of experience: 10
----------
Name: Ali
Position: S
Salary(*1000): 24
Years of experience: 8
----------
Name: Bahador
Position: S
Salary(*1000): 24
Years of experience: 8
----------
```

توجه: در تمامی فرآیند ذخیره و نمایش اطلاعات تعداد – ، 10 است.

آنچه باید آپلود کنید

فایل DealerShip.java را صرفا به‌صورت تک فایل آپلود کنید.

Traffic Manager (امتیازی)

- سطح : امتیازی
- طراح : آریا زریاب

در این تمرین، شما باید یک چهارراه شبیه‌سازی‌شده را پیاده‌سازی کنید که در آن ماشین‌ها از جهات مختلف وارد تقاطع می‌شوند و به یکی از سه جهت راست، مستقیم و چپ حرکت می‌کنند . ماشین‌ها به‌صورت نخ‌های موازی پیاده‌سازی شده‌اند و در این چهارراه، ممکن است چندین ماشین هم‌زمان وارد شوند.
حال شما باید این چهارراه را طوری مدیریت کنید که ماشین ها با هم برخورد نداشته باشند ، بن بست (deadlock) به وجود نیاید و از ایجاد race condition میان چند نخ هنگام دسترسی مشترک به چهارراه جلو گیری شود.

▼ اطلاعات بیشتر

برای مطالعه اهمیت رخ دادن race condition میتوانید به این لینک مراجعه کنید.

همچنین برای مطالعه ددلاک میتوانید به اینجا مراجعه کنید..

همچنین یکی از راه کار های جلوگیری از وقایع بالا استفاده از قفل میباشد که یکی از کلاس های معروف آن در جاوا ReentrantLock میباشد که میتوانید برای مطالعه آن به اینجا مراجعه کنید.

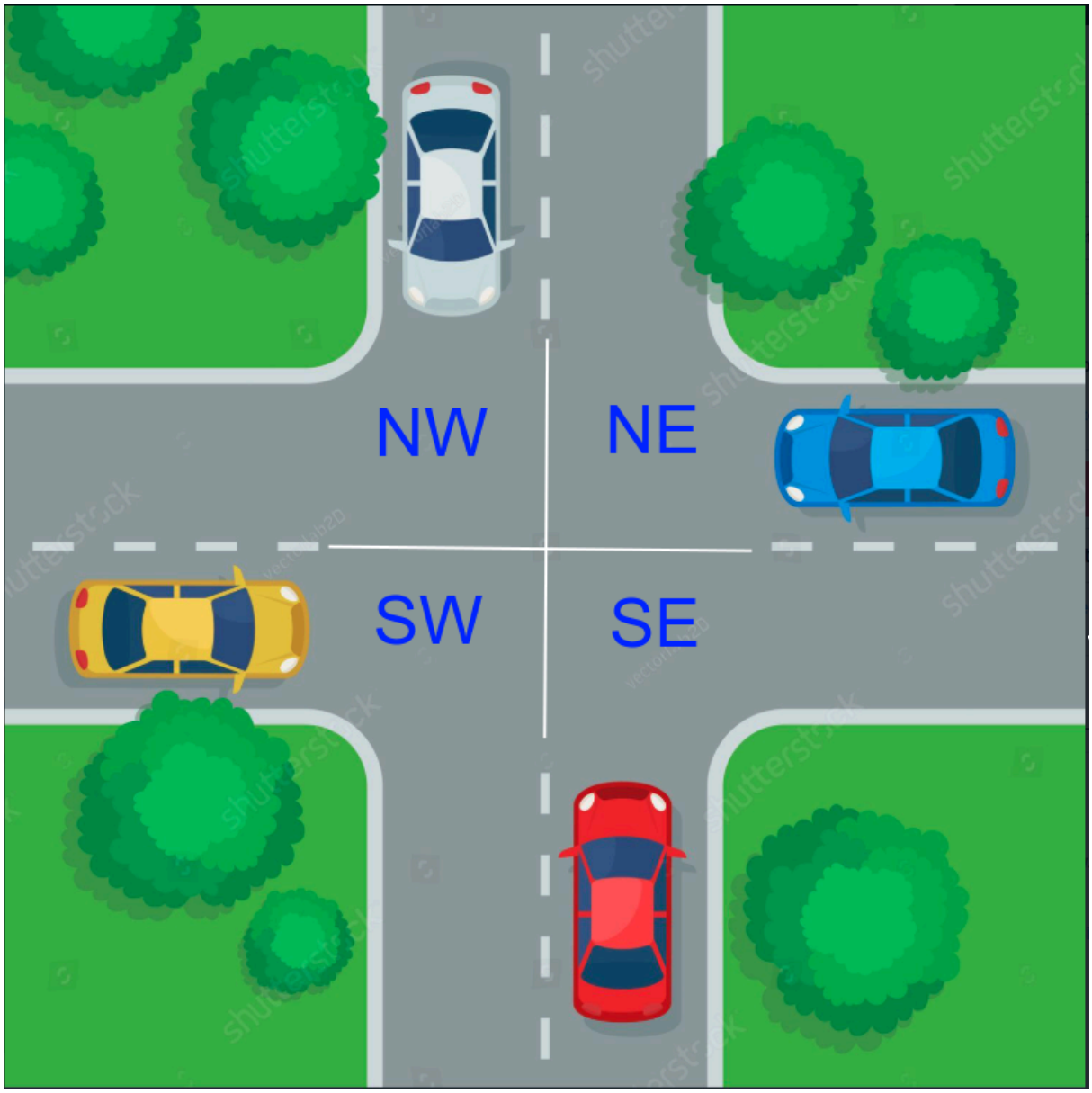
برای دانلود پروژه اولیه روی این لینک کلیک کنید.

▼ ساختار فایل پروژه

```
<zip_file_name.zip>
      ├── Car.java
      ├── Direction.java
      ├── IntersectionController.java
      ├── IntersectionControllerAllAtOnce.java
      ├── IntersectionControllerStepByStep.java
      ├── Main.java
      └── Turn.java
```

جزئیات برنامه

ابتدا فرض میکنیم که تقاطع ما به شکل زیر میباشد :



کلاس Car :

ترد یا نخ های برنامه میباشند که دارای ویژگی های زیر هستند :

۱. `ID` : یک شماره برای هر ماشین که از نوع `int` است.

۲. `direction` : تعیین میکند که ماشین از چه جهتی به تقاطع میرسد و از جنس `Direction` است.

۳. `turn` : تعیین میکند که ماشین به کدام جهت میخواهد حرکت کند و از جنس `Turn` است.

۴. `controller` : یک کنترلر که تعیین میکند این ترد متعلق به کدام کنترلر چهارراه است و از جنس `IntersectionController` میباشد.

که همگی آن ها `final` و `private` هستند.

متد ها :

۱. کانستراکتور :

تمام پراپرتی هایی که دارد را مقدار دهی میکند :

```
public Car (int ID, Direction direction, Turn turn, Intersection
```

۲. متد `run` که از والدش ارثبری میکند را `override` میکند.

در این متد ، شئ کلاس در صورت موفقیت فقط وارد تقاطع میشود.

```
@Override
public void run() {
}
```

متد های گتر و ستر مورد نیاز.

همچنین متد toString پیاده شده است و نیازی به تغییر آن ندارید.

اینترفیس `IntersectionController` :

شامل ۳ متد زیر میباشد :

۱. متد `enterIntersection` :

که پیاده سازی آن به عهده فرزندانش است و با توجه به فرزند مربوطه باید پیاده شود. کار این متد این است که شئ ماشین ورودی را وارد تقاطع کند.

```
void enterIntersection(Car car);
```

۲. متد `getZones` :

به صورت `default` در همین اینترفیس تعریف و پیاده میشود.

دو ورودی `direction` و `turn` را به ترتیب میگیرد و لیستی از مناطق (zone) هایی که ماشین هنگام عبور از تقاطع نیاز دارد را به ترتیب ورود ماشین به آنها برمیگرداند.

به عبارتی به ازای هر `turn` و `direction` ای یک لیست خواهیم داشت.

```
default List<Integer> getZones(Direction dir, Turn turn)
```

۳. متد `zoneToName` :

عدد zone را به رشته مربوطه نگاشت میکند و از قبل پیاده شده است.

کلاس `IntersectionControllerAllAtOnce` :

فرزند اینترفیس `IntersectionController` است و این کنترلر به این صورت عمل میکند که ماشین هایی که خواهیم داشت هر کدام که از تقاطع خارج شود دیگری وارد میشود یعنی در هر لحظه حداکثر یک ماشین در تقاطع خواهیم داشت حتی اگر مسیر دو ماشین با هم اشتراکی نداشته باشند.

تنها پراپرتی این کلاس آرایه ای از قفل ها به نام `zones` است که `final` و `private` میباشد.

کانستراکتور این کلاس بدون ورودی میباشد و قفل ها را مقدار دهی اولیه (new) میکند.

تنها متد این کلاس متد `enterIntersection` است که از والدش `Override` میکند.

در ابتدا وقتی شئ ماشین ورودی گرفته شد مناطقی که میخواهد به این صورت چاپ میشوند :

```
car + " wants zones: " + neededZones
```

که اینجا car همان خروجی toString اش میباشد و neededZones به صورت یک لیست (List<Integer>) چاپ میشود.

حال هر وقت ماشین وارد تقاطع شد پیغام زیر چاپ خواهد شد :

```
car + " has entered the intersection."
```

برای نزدیک شدن مسئله به واقعیت وقتی نخ ماشینی وارد تقاطع میشود باید یک مدت زمانی آنجا بماند و این مدت زمان را با فرمول زیر حساب میکنیم که در واحد میلی ثانیه است :

100 + (random number between 0 to 100)

و همچنین هر وقت خارج شد پیغام زیر چاپ خواهد شد :

```
car + " has exited the intersection."
```

کلاس IntersectionControllerStepByStep :

فرزند اینترفیس IntersectionController است و این کنترلر به این صورت عمل میکند که ماشین هایی که خواهیم داشت هر کدام که در تقاطع حضور داشته باشند ماشین دیگری هم نیز میتواند در تقاطع وجود داشته باشد ، در واقع میتوانیم چند ماشین را در تقاطع داشته باشیم ولی حواسمان باید باشد که برخورد و تصادفی رخ ندهد.

تنها پراپرتی این کلاس آرایه ای از قفل ها به نام zones است که final و private میباشد.

کانستراکتور این کلاس بدون ورودی میباشد و قفل ها را مقدار دهی اولیه (new) میکند.

تنها متد این کلاس متد enterIntersection است که از والدش Override میکند.

در ابتدا وقتی شئ ماشین ورودی گرفته شد مناطقی که میخواهد به این صورت چاپ میشوند :

```
car + " needs zones: " + neededZones
```

که اینجا car همان خروجی toString اش میباشد و neededZones به صورت یک لیست (List<Integer>) چاپ میشود.

حال هر وقت ماشین قفل یک zone را گرفت پیغام زیر چاپ میشود.

```
car + " pre-locked zone: " + zoneToName(zone)
```

و با وارد شدن ماشین به این zone پیغام زیر چاپ میشود :

```
car + " entered zone: " + zoneToName(zone)
```

همچنین با باز کردن آن قفل پیغام زیر چاپ میشود :

```
car + " released zone: " + zoneToName(prevZone)
```

و با خارج شدن از تقاطع این دو خط چاپ خواهند شد :

```
car + " released last zone: " + zoneToName(prevZone)
car + " has exited the intersection."
```

همچنین زمان در نظر گرفته شده برای حضور هر ماشین در تقاطع در این کلاس 100 میلی ثانیه میباشد.

Direction :

یک `enum` میباشد که شامل جهت های زیر است :

> NORTH, SOUTH, EAST, WEST

Turn :

یک `enum` میباشد که شامل عملیات های زیر میباشد :

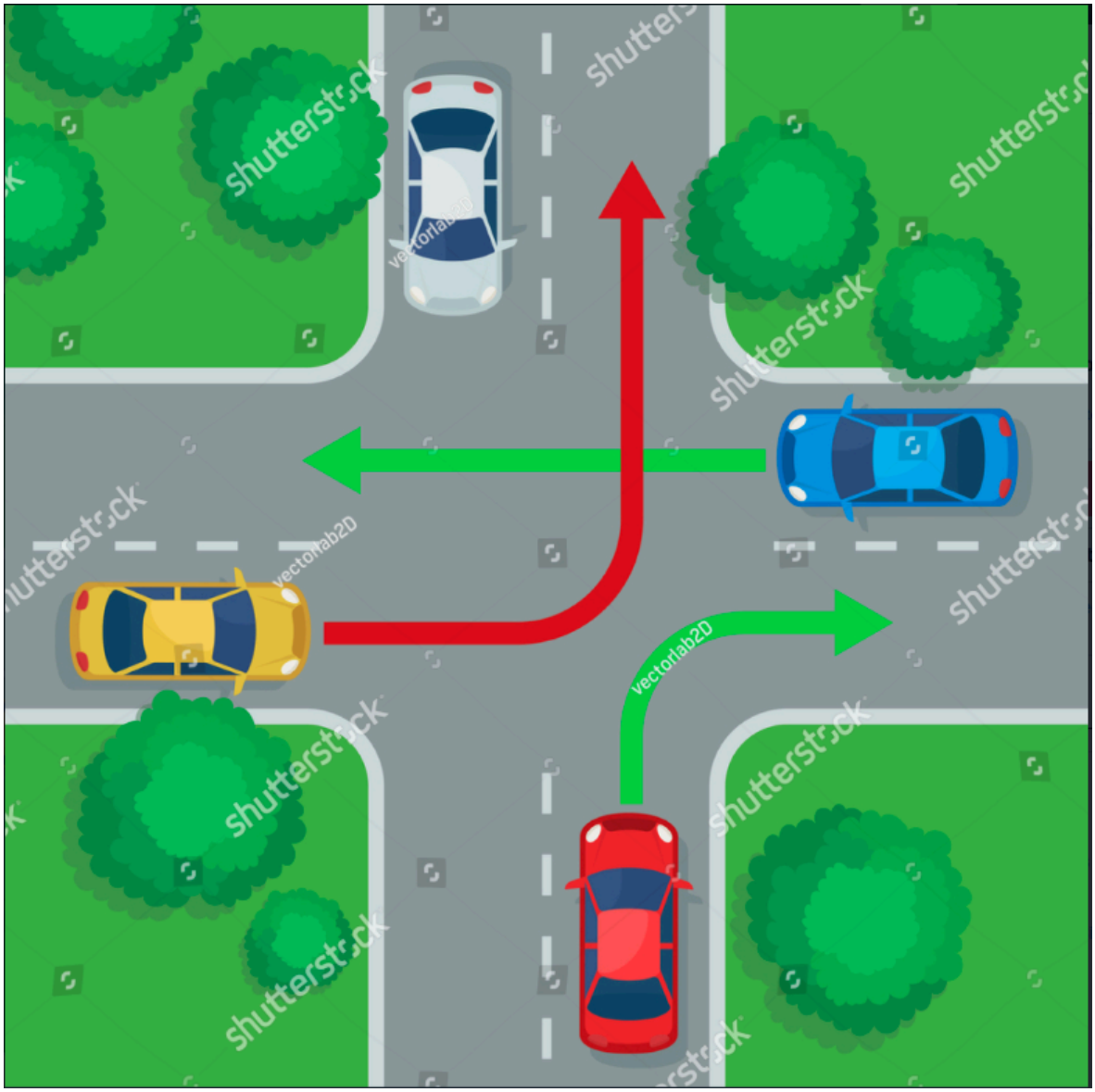
> LEFT, STRAIGHT, RIGHT

همچنین در پروژه اولیه تمام قسمت هایی که پیاده سازی آن بر عهده شما میباشد مشخص شده است.

نکته : همچنین به اکسپشن های مرتبط با ترد هنگام پیاده سازی توجه داشته باشید.

▼ مثال

به عنوان مثال شکل زیر را در نظر بگیرید



در این مثال برای هر ماشین مناطق زیر را به ترتیب از چپ به راست نیاز خواهیم داشت :

🟨 ماشین زرد :

از WEST می آید و عملیات LEFT را انجام میدهد.

SW, SE,NE

🟥 ماشین قرمز :

از SOUTH می آید و عملیات RIGHT را انجام میدهد.

SE

🟦 ماشین آبی :

از EAST می آید و عملیات STRAIGHT را انجام میدهد.

NE, NW

همچنین در فایل اولیه میتوانید با اجرای Main یک نمونه از خروجی را مشاهده کنید.

آنچه باید آپلود کنید

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

```
<zip_file_name.zip>
      ├── Car.java
      ├── Direction.java
      ├── IntersectionController.java
      ├── IntersectionControllerAllAtOnce.java
      ├── IntersectionControllerStepByStep.java
      └── Turn.java
```