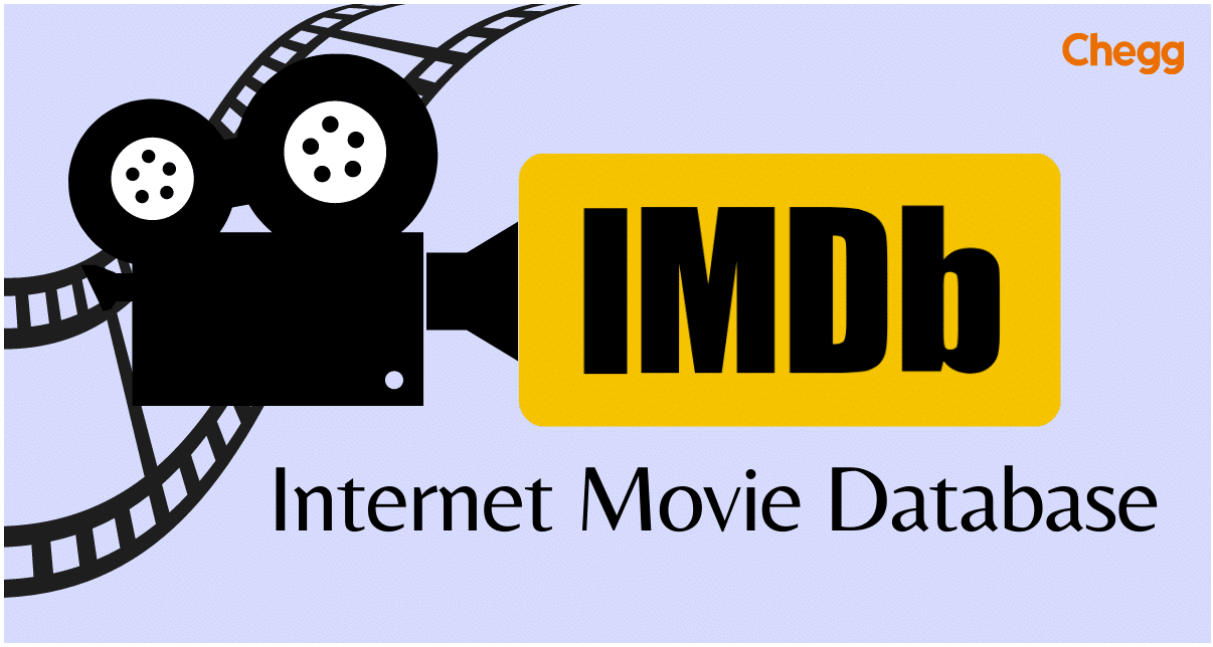
تحلیل فیلم

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



مهدی که یک تازه وارد به دنیای سینما است، به تازگی با سایت imdb آشنا شده و به دلایلی نامعلوم(احتمالا از روی کنجکاوی)، می خواهد به یک سری سوال که ذهنش را درگیر کرده پاسخ دهد.

او فایل csv حاوی داده های ۲۵۰ برتر فیلم های imdb را در اختیار دارد و می خواهد یک سری عملیات روی آن انجام دهد.

عملیات شماره ۱: بدست آوردن میانگین فروش(box_office) یک ژانر مشخص.

عملیات شماره ۲: بدست آوردن مدت زمان (run_time) یک فیلم مشخص.

عملیات شماره ۳: بدست آوردن بهترین فیلم از یک کارگردان مشخص.

از آنجایی که خودش مهارت چندانی در برنامه نویسی ندارد، از شما می خواهد در انجام این کار به او کمک کنید.

فایل داده ها را می توانید از اینجا دانلود کنید.

ساختار کلی برنامه را از اینجا دانلود کنید.

توابع برنامه:

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

این تابع میانگین فروش یک ژانر مشخص را محاسبه می کند. و مقدار بازگشتی آن بدین صورت است:

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

```
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
```

```
public String findHighestRatingMovieByDirector(String director)
        // TODO
    }
```

و در آخر این تابع بهترین فیلم براساس نمره دریافتی از یک کارگردان مشخص را پیدا می کند. و مقدار بازگشتی آن بدین صورت است:

```
'highestRatedMovie' with rating highestRating
```

نکات مهم:

- ممکن است مقدار `box_office` برای بعضی فیلم ها موجود نباشد. در این صورت آن ها را نادیده می گیریم.
- در محاسبه تابع میانگین فروش از نوع داده `long` استفاده کنید.

- نوشتار کلمات دقیقا مطابق فایل داده خواهد شد.
- به علامت های ' در مقدار بازگشتی توابع دقت کنید.

مثال عملیات 1

```
genreToFind = "Drama"
returnValue = average box office for genre 'Drama': $184938994
```

مثال عملیات 2

```
movieName = Fight Club
returnValue = run time of 'Fight Club': 2h 19m
```

مثال عملیات 3

```
director = Quentin Tarantino
returnValue = 'Pulp Fiction' with rating 8.9
```

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

فایل بهم‌ریخته

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

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: نام فایل اولیه `info.txt` می‌باشد.

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` ذخیر 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.

Task Manager

- محدودیت زمان : ۱ ثانیه
- طراح: برنا ماهرانی
- سطح: متوسط

به علت گرانی پردازنده، میثم قصد دارد سیستم عاملی طراحی کند که از پردازنده قدیمی او نهایت استفاده را ببرد تا برای مدتی نیازی به پردازنده جدید نداشته باشد. او در پیاده‌سازی Task Manager این سیستم عامل به مشکل خورده است. به او کمک کنید تا سیستم عاملش را تکمیل کند.

برای راهنمایی و ساختارمندی کدتان، می‌توانید پروژهٔ اولیه را از این لینک دانلود کنید؛ کلاس‌هایی که در سؤال توضیح داده نشده‌اند و در پروژهٔ اولیه موجود اند، صرفاً جهت نظم‌دهی به کد اضافه شده‌اند و مستقیماً تست نمی‌شوند. بنابراین می‌توانید آن‌ها را حذف یا به دلخواه خود تغییر دهید.

کلاس `TaskManager`

- این کلاس وظیفه مدیریت درخواست‌ها (task) و تقسیم آن‌ها بین تعداد محدودی ترد (thread) را بر عهده دارد.
- پیاده‌سازی باید به گونه‌ای باشد که درخواست‌ها ابتدا به یک صف اضافه شوند. پس از فارغ شدن یکی از تردها از پردازش قبلی خود، آن ترد یک درخواست جدید را از صف خارج می‌کند و به پردازش آن می‌پردازد.
- دقت کنید که امکان *race condition* بین تردها برای گرفتن درخواست جدید وجود دارد. در پیاده سازی خود از این موضوع جلوگیری کنید.
- هر درخواست باید فقط یک بار و توسط یک ترد انجام شود. درصورت خالی بودن صف، تردها کاری انجام نمی‌دهند و منتظر آمدن درخواست جدید می مانند.

سازنده

```
1 | public TaskManager(ThreadGroup group, int threadCount)
```

پارامتر اول، گروه تردها را مشخص می‌کند و پارامتر دوم تعداد تردها را. این کلاس وظیفه دارد به تعداد `threadCount` که یک عدد مثبت است، ترد جدید بسازد. این تردهای جدید باید عضو گروه `group` شوند. برای این کار، نیاز است تا از سازنده مناسبی از میان سازنده‌های متفاوت کلاس `Thread` استفاده کنید.

متد doTask

```
1 | public void doTask(Runnable task)
```

این متد صرفا یک درخواست را به صف اضافه می‌کند تا بعداً یکی از تردهای موجود آن را پردازش کند (منظور از پردازش کردن، فراخوانی متد `run` پارامتر task است).
این متد نباید منتظر اجرای درخواست توسط تردها بماند و به محض اضافه کردن آن به صف، باید به اتمام برسد.

نکات

- تضمین می‌شود حداکثر 100 درخواست به صف اضافه شود.
- استفاده از کتابخانه‌های جاوا برای پیاده سازی صف **ممنوع است** و لازم است که مکانیزم صف را خودتان پیاده‌سازی کنید. برای این کار می‌توانید از این لینک ایده بگیرید.
- خوب است که قبل از حل سوال، در مورد مسئله *Producer-Consumer* و راه‌حل‌های آن مطالعه کنید.
- متد `doTask` و سایر متدهایی که به کلاس `TaskManager` اضافه می‌کنید را در صورت نیاز با کلمه کلیدی `synchronized` تعریف کنید:

```
1 | public synchronized void doTask(Runnable task)
```

نمونه

```
public class Main {
    public static void main(String[] args) {
        TaskManager tm = new TaskManager(new ThreadGroup("tm"),
        tm.doTask(new TimedPrinter(1500, "onc")); // 3
```

```
        tm.doTask(new TimedPrinter(500, "wri")); // 1
        tm.doTask(new TimedPrinter(500, "te ")); // 2
        tm.doTask(new TimedPrinter(1000, "e \n")); // 4
        tm.doTask(new TimedPrinter(1000, "n e")); // 6
        tm.doTask(new TimedPrinter(200, "rui")); // 5
        tm.doTask(new TimedPrinter(800, "veryw")); // 7
        tm.doTask(new TimedPrinter(2000, "here!\n\n")); // 8
    }
}

class TimedPrinter implements Runnable {
    private final long delay;
    private final String str;

    TimedPrinter(long delay, String str) {
        this.delay = delay;
        this.str = str;
    }

    @Override
    public void run() {
        try {
            Thread.sleep(delay);
        } catch (InterruptedException ignored) {
        }
        System.out.print(str);
        if (str.endsWith("\n\n")) {
            System.exit(0);
        }
    }
}
```

خروجی کد بالا به صورت زیر خواهد بود.

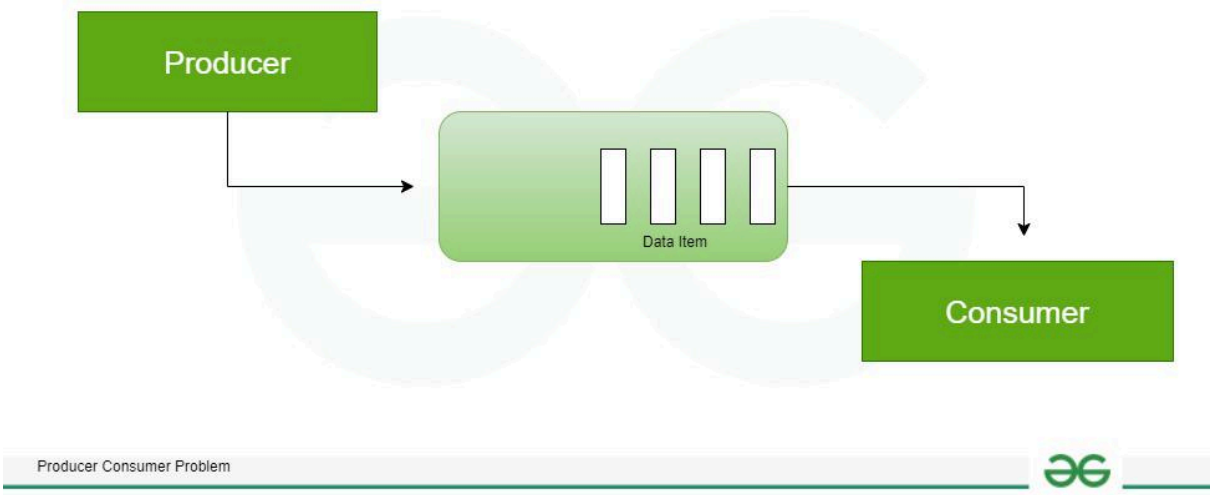
```
write once
ruin everywhere!
```

آنچه باید آپلود کنید

فایل `TaskManager.java` را بدون هرگونه پوشه‌بندی و فشرده‌سازی آپلود کنید.

Producer (امتیازی)

- محدودیت زمان: ۱ ثانیه
- طراح : آریا شاکو



Producer

شما در این تمرین باید مسئله‌ی تولیدکننده و مصرف‌کننده را شبیه‌سازی کنید. در مسئله دو نخ `Producer` و `Consumer` وجود دارد. نخ تولیدکننده همیشه در حال اضافه کردن یک شی به لیست مشترک است و نخ مصرف‌کننده همیشه در حال برداشتن اشیای موجود از لیست مشترک است.

شما باید به گونه‌ای کلاس `SafeResource` را پیاده‌سازی کنید که از ایجاد race-condition میان دو نخ تولیدکننده و مصرف‌کننده هنگام دسترسی مشترک به لیست جلوگیری شود.

پروژه اولیه را می توانید از اینجا دانلود کنید این لینک

برای پیاده سازی این سوال به FIFO توجه کنید.

کلاس SafeResource

```
1 import java.util.ArrayList;
2 import java.util.List;
```

```
3
4  public class SafeResource {
5      private final List<Object> objectList = new ArrayList<>();
6      public static final int  MAX_SIZE = 10;
7
8      public SafeResource(int capacity) {
9          // TODO
10     }
11
12     public void addNewObject(Object object) {
13         //TODO
14     }
15
16     public  Object getNextObject()  {
17         // TODO
18     }
19
20     public int getCurrentSize() {
21         // TODO
22     }
23
24     public boolean isFull() {
25         // TODO
26     }
27
28     public boolean isEmpty() {
29     // TODO
30     }
31
32     public List<Object> getObjectList() {
33         return new ArrayList<>(objectList);
34     }
35 }
```

پراپرتی‌ها

این کلاس دارای ویژگی `objectList` است که لیست اشیای تولید شده توسط نخ تولیدکننده را نگهداری می‌کند.

متد `addNewObject`

این متد یک شی از نوع `Object` ورودی می‌گیرد و آن را به لیست `objectList` اضافه می‌کند. در همین متد باید از race-conditionهای احتمالی جلوگیری شود. و توجه کنید که شی به ته لیست اضافه می شود. این متد توسط کلاس `Producer` برای اضافه کردن یک شی جدید فراخوانی می‌شود.

متد `getNextObject`

این متد آخرین شی اضافه شده را از لیست `objectList` حذف می‌کند و آن را بر می‌گرداند. در همین متد باید از race-conditionهای احتمالی جلوگیری شود. توجه کنید برای گرفتن شی جدید باید شی قبلی را بر دارید.

متد `isFull` **و** `isEmpty`

در این دو متد باید پر یا خالی بودن لیست شی ها را بررسی کنید.

متد `getCurrentSize`

باید سایز لیست شی را بررسی کنید.

این متد توسط کلاس `Consumer` برای دریافت یک شی فراخوانی می‌شود.

آنچه باید آپلود کنید

تنها فایل `SafeResource.java` بدون `zip` کردن ارسال کنید:

`SafeResource.java`