# دیتابیس کاربرها

- سطح: آسان
- طراح: روزبه سلطانی

پروژه اولیه از این لینک دانلود بفرمایید.

در این پروژه باید یک **سیستم ثبت‌نام و ورود کاربر (Login/Register)** با استفاده از **فایل** طراحی کنید.

هر کاربر دارای نام کاربری (**username**)، رمز عبور (**password**) و شماره تلفن (**phone**) است.
اطلاعات کاربران باید در فایل ذخیره شده و در اجراهای بعدی نیز قابل خواندن باشند.

## قوانین برنامه

۱. اطلاعات کاربران در فایل متنی (مثلاً `users.txt` ) ذخیره می‌شود.
هر خط از فایل، اطلاعات یک کاربر را به فرمت زیر دارد:

```
username,password,phone
```

۲. اگر فایل موجود نباشد، برنامه باید آن را ایجاد کند.
۳. کاربران می‌توانند از طریق متد `register` ثبت‌نام کنند.
- نام کاربری نباید تکراری باشد.
- شماره تلفن باید معتبر باشد (یعنی ۱۱ رقمی و با `09` شروع شود).
- در صورت موفقیت، اطلاعات باید در فایل ذخیره شود.

۴. ورود کاربران از طریق متد `login` انجام می‌شود.
- اگر نام کاربری وجود داشته باشد و رمز عبور درست باشد ← `"Login successful."`
- اگر رمز اشتباه باشد ← `"Invalid password."`
- اگر نام کاربری وجود نداشته باشد ← `"User not found."`

۵. تمام عملیات خواندن و نوشتن باید با فایل واقعی انجام شود.
آرایه فقط برای نگهداری موقت داده‌ها در زمان اجرا استفاده می‌شود.

۶. استفاده از هیچ‌گونه **کالکشن (Collection)** مانند `ArrayList` یا `HashMap` مجاز نیست.

فقط از آرایه و کلاس‌های استاندارد فایل ( `File` , `Scanner` , ...) استفاده کنید.

## ساختار کلاس‌ها

### 1. کلاس `User`

نمایانگر اطلاعات یک کاربر در سیستم است.

```
class User {
    private String username;
    private String password;
    private String phone;

    public User(String username, String password, String phone)
    public String getUsername()
    public String getPassword()
    public String getPhone()
}
```

### 2. کلاس `UserManager`

مدیریت ثبت‌نام، ورود، و خواندن داده‌ها از فایل را برعهده دارد.

```
class UserManager {
    private User[] users;
    private int userCount;
    private String filename;

    public UserManager(String filename)
    public void loadFromFile()
    public boolean register(String username, String password, String phone)
    public String login(String username, String password)
    public boolean isValidPhone(String phone)
}
```

**توضیحات متدها:**

- **UserManager(String filename)**

مسیر فایل را دریافت کرده، اگر فایل وجود نداشته باشد آن را ایجاد می‌کند

و سپس با استفاده از متد `loadFromFile` اطلاعات کاربران را بارگذاری می‌کند.

- **void loadFromFile()**

تمام خطوط فایل را می‌خواند و هر کاربر را در آرایه `users` ذخیره می‌کند.

- **boolean register(String username, String password, String phone)**

ابتدا با استفاده از متد `loadFromFile` اطلاعات کاربران را بارگذاری می‌کند و سپس یک کاربر جدید

به فایل اضافه می‌کند، اگر شماره تلفن معتبر نباشد یا نام کاربری تکراری باشد،

عملیات ثبت انجام نمی‌شود و `false` برمی‌گرداند.

در غیر این صورت، `true` بازمی‌گردد.

- **String login(String username, String password)**

در فایل جستجو می‌کند. اگر کاربر و رمز صحیح باشد، `"Login successful."` ، اگر رمز اشتباه

باشد `"Invalid password."` و در غیر این صورت `"User not found."` برمی‌گرداند.

- **boolean isValidPhone(String phone)**

بررسی می‌کند شماره تلفن معتبر است یا خیر (۱۱ رقم و شروع با 09).

## نکات مهم

- داده‌ها باید پس از ثبت در فایل باقی بمانند و در اجرای بعدی نیز قابل خواندن باشند.
- فایل در همان پوشه‌ی پروژه قرار دارد.

## فایل زیپ آپلودی وقتی باز میشود باید فقط شامل فایل های زیر باشد:

```
├── User.java
└── UserManager.java
```

# شمارش آرا

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: آناهیتا بیانی

فایل اولیه پروژه را این‌لینک دریافت کنید.

یک فایل شامل آرا در اختیار دارید که در هر خط نام یک کاندیدا نوشته شده است. هدف این است که با استفاده از چند ترد(Thread)، فایل آراء را **به چند بازه** تقسیم کنید، تعداد آرا را بشمارید و در پایان نتایج را تجمیع و چاپ کنید. هر ترد فقط خطوط بازه خودش را می‌خواند و تعداد رأی‌های کاندیداها و رای‌های نامعتبر را می‌شمارد.

## کلاس worker:

هر نمونه از این کلاس یک ترد است که مسئول خواندن بخشی از فایل (یک بازه از خطوط) و شمارش رأی‌ها در همان محدوده است. باید از کلاس مناسب ارث‌بری کند.

### فیلدهای کلاس Worker:

```
1 private final File file;
2 private final long startLine;
3 private final long endLine;
4 private final String[] cands;
5 private final int[] localCounts;
6 private long invalid = 0;
```

`file File`: فایل رأی‌ها. همهٔ تردها این فایل را دارند ولی هرکدام بازه مربوط به خود را بررسی می‌کند.

`startLine long`: شمارهٔ خط شروع بازهٔ کاری این ترد.

`endLine long`: شمارهٔ خط پایان بازهٔ کاری این ترد.

`String[] cands` آرایه‌ای از نام‌های کاندیداها به صورت lowercase.( بدون حساسیت به بزرگی و کوچکی حروف آرا را بشمارید.)

`int[] localCounts`: آرایه‌ای هم‌طول با cands که تعداد رأی هر کاندیدا در همین بازه را نگه می‌دارد.

`long invalid`: شمار رأی‌هایی که معتبر نیستند (یعنی نامی غیر از کاندیداها دارند).

### متد run():

این متد هنگامی که ()start فراخوانی شود صدا زده می‌شود.

▼ تفاوت فراخوانی ()run و ()start

متد ()run فقط شامل منطق کار ترد است، اما خودش ترد جدید ایجاد نمی‌کند. اگر ()run را مستقیماً صدا بزنید، برنامه به‌صورت تک‌نخی (Sequential) اجرا می‌شود، متد ()start وظیفه دارد یک Thread جدید در JVM بسازد و سپس متد ()run را در آن ترد جدید اجرا کند.

در این متد فایل را خط‌به‌خط بخوانید. فقط خط‌ها در بازه‌های لازم را بررسی کنید. نام‌ها را بدون توجه به بزرگی یا کوچکی حروف بررسی کنید.

## کلاس runVoteCounter:

این متد آرگومان‌های مسیر فایل (path)، تعداد خطوط (N)، تعداد تردها (T) و تعداد کاندیداها (M) را می‌گیرد. در این متد تقسیم‌بندی بازه‌ها انجام می‌شود تا هر ترد بداند از کدام خط تا کدام خط را بخواند و سپس ساخت و شروع تردها انجام می‌شود. پس از باز کردن فایل، یک آرایه از تردها بسازید و در یک حلقه به هر عنصر از آرایه یک شی از کلاس worker اختصاص دهید.(از constructor کلاس worker استفاده کنید.) و سپس آن ترد را ()start کنید. به هر ترد به میزان base = N / T خط اختصاص دهید و در صورتی که N بر T بخش‌پذیر نباشد، به ترد پایانی باقی‌مانده خطوط را نیز اختصاص دهید. هر ترد به شکل مستقل بخش مربوطه از فایل را می‌خواند و نتایج خودش را در فیلدهای localCounts و invalid نگه می‌دارد. در آخر باید منتظر بمانید تا کار هر ترد تمام شود. برای اطمینان از این موضوع بر روی هر ترد از متد ()join استفاده کنید، سپس نتیجه را چاپ کنید.

### نمونه فایل ورودی:

```
Ali
invalid
ali
Maryam
maryam
invalid
invalid
Ali
maryam
maryam
```

خروجی:

```
ali: 3
maryam: 4
Invalid votes: 3
```

# Task Manager

- سطح: متوسط
- طراح: آرتین عضدی فر

ابتدا فایل اولیه پروژه را از این لینک دانلود کنید.

یک تیم تازه کار که برای مدیریت و برنامه ریزی پروژه‌شان اعتقادی به برنامه های مرسوم ندارند برنامه هایشان را در یک فایل متنی ذخیره می‌کنند. شما استخدام شده‌اید که برنامه ای بنویسید که مدیریت کارهایشان را برایشان آسان‌تر کند.

## ساختار فایل متنی:

هر خط از این فایل فرمتی به شکل زیر دارد:

```
TaskID|TaskName|AssignedTo|DueDate|Status|HoursSpent
```

توضیح: هر تسک شناسه یکتای خود را دارد، بخش بعدی نام و توضیحات تسک است.

`AssignedTo` : انجام دهنده کار

`DueDate` : ددلاین تسک با فرمت `Year-Month-Day`

`Status` : وضعیت تسک (به پایان رسیده، شروع نشده یا در حال انجام)

`HoursSpent` : عدد صحیحی که نشان می‌دهد چند ساعت روی تسک مورد نظر وقت گذاشته شده.

## کلاس TaskManager:

### فیلدها:

تنها فیلد این کلاس `file` از جنس `File` است.

## متدها:

### سازنده:

```
1 | public TaskManager(String filePath)
```

یک رشته شامل آدرس فایل را ورودی میگیرد.

۱. `getTaskInfo` : یک شناسه تسک ورودی میگیرد و در صورت موجود بودن آن در فایل، اطلاعات آن را به فرمت زیر بر می‌گرداند:

```
Task: [Task name]
Assigned to: [name]
Deadline: [date]
Status: [status]
Hours Spent: [hours]
***
```

و در صورت وجود نداشتن تسکی با آن شناسه:

```
Not found.
***
```

۲. `getTasksByStatus` : یک وضعیت ورودی می‌گیرد و لیست شناسه تسک هایی که در آن وضعیت هستند را بر می‌گرداند.

```
[status] tasks:
[ID1]: [task name]
[ID2]: [task name]
.
.
.
[IDn]: [task name]
***
```

۳. `getTasksBefore` : یک تاریخ (با فرمت `Year-Month-Day` ) ورودی می‌گیرد و لیست شناسه تمام

تسک‌هایی که ددلاینشان حداکثر همان تاریخ است را بر می‌گرداند.

```
[ID1]: [task name] should end in [deadline]
[ID2]: [task name] should end in [deadline]
.
.
.
[IDn]: [task name] should end in [deadline]
***
```

۴. `getAverageHoursByStatus` : یک وضعیت را ورودی می‌گیرد و میانگین ساعاتی که روی تسک هایی با آن وضعیت کار شده است را با جنس `double` بر میگرداند. در صورتی که هیچ تسکی با آن وضعیت وجود نداشت `-1` بر می‌گرداند.

۵. `updateStatus` : یک شناسه، وضعیت و میزان ساعات کار شده ورودی می‌گیرد و وضعیت و ساعات کاری تسک با شناسه گفته شده را (در صورت وجود) به مقدارهای ورودی آپدیت کند. در صورت موفقیت آمیز بودن true و در غیر این صورت false بر می‌گرداند.

**نکته**: ساعات کاری نمی‌تواند منفی باشد، وضعیت قابل قبول فقط یکی از سه حالت: `Not Started` ، `In Progress` و `Completed` است.

همچنین تسکی که در وضعیت شروع نشده قرار می‌گیرد نمی‌تواند ساعت کاری بالاتر از صفر داشته باشد. و در نهایت ساعات صرف شده روی تسک از مقدار قدیمی آن نمی‌تواند کمتر باشد.

## مثال:

در صورت اجرای کد زیر:

```
1  public static void main(String[] args) {
2
3      TaskManager tm = new TaskManager("File.txt");
4
5      try {
6
7          System.out.println(tm.getTaskInfo("T004"));
8
9
```

```
            System.out.println(tm.getTasksBefore("2025-06-1"));

            System.out.println(tm.getTasksByStatus("Completed"));

            System.out.println("Average hours spent on Completed Tasks: " + tm.ge

        } catch (IOException e) {

            e.printStackTrace();

        }

    }
```

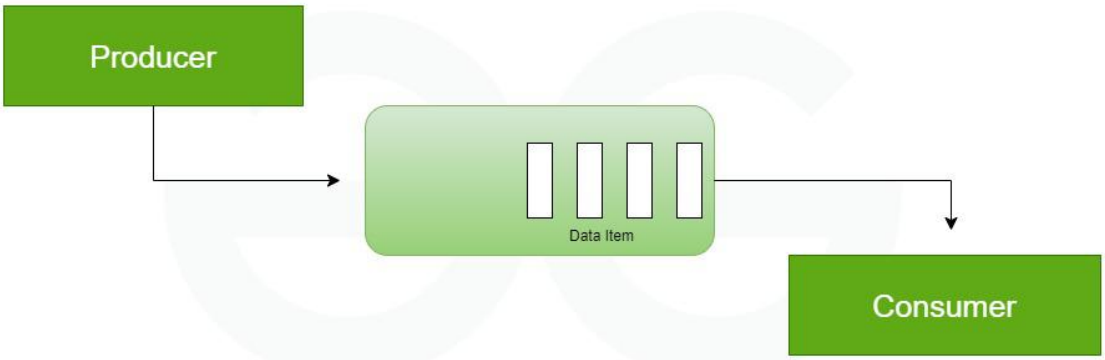
خروجی باید به شکل زیر باشد:

```
Task: Bug Fix
Assigned to: Alice
Deadline: 2025-03-12
Status: Completed
Hours Spent: 7
***
T001: Database Backup should end in 2025-01-12
T002: Database Backup should end in 2025-02-18
T003: API Testing should end in 2025-03-05
T004: Bug Fix should end in 2025-03-12
T005: Code Review should end in 2025-04-22
T006: UI Design should end in 2025-05-10
T007: Database Backup should end in 2025-05-18
***
Completed tasks:
T001: Database Backup
T004: Bug Fix
T006: UI Design
T007: Database Backup
T009: Deployment
***
Average hours spent on Completed Tasks: 5.4
```

## آنچه باید آپلود کنید:

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

# Producer Consumer




Producer Consumer Problem

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.

▼ اطلاعات بیشتر

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

پروژه اولیه را می توانید از این لینک دانلود کنید.

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 {
```

```
    private final List<Object> objectList = new ArrayList<>();
    public static final int  MAX_SIZE = 10;

    public SafeResource(int capacity) {
        // TODO
    }

    public void addNewObject(Object object) {
        //TODO
    }

    public  Object getNextObject()  {
        // TODO
    }

    public int getCurrentSize() {
        // TODO
    }

    public boolean isFull() {
        // TODO
    }

    public boolean isEmpty() {
    // TODO
    }

    public List<Object> getObjectList() {
        return new ArrayList<>(objectList);
    }
}
```

### 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`