Ctrl + H

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

آریا از مرورگر لپ‌تاپش کار زیاد می‌کشد! به همین دلیل بخش *History* مرورگر لپ‌تاپش از کار افتاده است؛ بنابراین می‌خواهد *History* جدیدی برای آن درست کند. ولی با این تفاوت که *History* ای که درست می‌کند خیلی پیشرفته است و از یک رشته تشکیل شده که در ابتدا خالی است، و در هر مرحله عملیات زیر روی آن انجام می‌شود:

- `insert(i,x)` : کاراکتر `x` را در جایگاه iام رشته اضافه کن (برای مثال دستور `insert(1,x)` در ابتدای رشته حرف `x` را اضافه می‌کند و اگر طول رشته `len` باشد دستور `insert(len+1,x)` در اخر رشته حرف `x` را اضافه می‌کند.
- `delete(i)` : کاراکتر `i` ام را حذف کن
- `undo` : آخرین عملیات انجام شده روی رشته را باطل کن (یعنی به حالت قبلی برگردان) (دقت کنید صرفا دستورهای `insert` و `delete` باطل می‌شوند و خود `undo` باطل نمی‌شود)

حال آریا از شما کمک میخواهد تا برایش این *History* پیشرفته را پیاده کنید تا او هم به ددلاین هایش برسد :)

ورودی

در خط اول ورودی به شما عدد q داده می‌شود که نشان دهنده‌ی تعداد عملیات است.

سپس در q خط بعدی در هر خط یکی از `insert i x` یا `delete i` یا `undo` ظاهر می‌شود.

$$1 \leq q \leq 100$$

تضمین می‌شود عملیات `insert` و `delete` معتبر هستند یعنی از طول رشته در آن زمان بیشتر نمی‌باشند و همچنین `undo` زمانی‌که هیچ عملیاتی برای بازگرداندن نیست پدید نمی‌آید.

خروجی

رشته‌ای که درنهایت پس از عملیات ها به دست می‌آید را خروجی دهید.

مثال

ورودی نمونه ۱

```
7
insert 1 a
insert 2 b
insert 1 c
delete 2
undo
undo
insert 1 c
```

خروجی نمونه ۱

```
cab
```

رشته بعد از هر مرحله به صورت زیر است:

شماره دستور	رشته
1	a
2	ab
3	cab
4	cb
5	cab

شماره دستور	رشته
6	ab
7	cab

بوستان

- محدودیت زمان: # ثانیه
- محدودیت حافظه: ندارد
- سطح: متوسط
- طراح: نیلا چناری

در این سوال باید سامانه ای مانند گلستان برای دانشگاه درست کنید. اما چون دسترسی به دیتابیس ندارید باید از کالکشن های موجود در زبان ها استفاده کنید. دقت کنید که در این سوال محدودیت حافظه ندارید و فقط تمام عملیات های خواسته شده از شما را باید در زمان O(1) انجام دهید.

کلاس ها

- **توجه!!!** ابتدا باید با توجه به درخواست های صورت سوال نحوه پیاده سازی و دیتااستراکچرهای مورد نیاز خود را انتخاب کرده و سپس شروع به پیاده سازی کنید.
- کلاس های درنظر گرفته شده و پیشنهادی برای پیاده سازی سوال به این صورت هستند.

```
Stuent:
    String nationalCode;
    String firstName;
    String lastName;
    long studentId;
Professor:
    String nationalCode;
    String firstName;
    String lastName;
    long professorId;
Course:
    String courseName;
    int courseCredit ;
University:
    -class for storing data and doing the operations
Main:
    -main
```

اضافه کردن دانشجو:

- شما باید در صورتی داشنجو را قبول کنید که دانشجویی با این **کدملی** تا کنون ثبت نام نکرده باشد. وقتی دانشجویی در دانشگاه قبول میشود، به آن با توجه به قاعده زیر شماره دانشجویی میدهیم و برای آن ست میکینم:
- شماره دانشجویی به فرمت ######403:
- مثلا برای دانشجوی 93م: 403000093 (منطور 93مین دانشجو قبول شده است)

```
AddStudent nationalCode firstName lastName
```

اضافه کردن استاد:

- شما باید در صورتی استادرا قبول کنید که استادی با این **کدملی** تا کنون ثبت نام نکرده باشد.
- وقتی استادی در دانشگاه قبول میشود، به آن با توجه به قاعده زیر شماره استاد میدهیم و برای آن ست میکینم:
- شماره استادان به فرمت ###1:
- مثلا برای استاد93م: 1093 (منطور 93مین استاد قبول شده است)

```
AddProfessor nationalCode firstName lastName
```

اضافه کردن درس:

- شما باید در صورتی درس را قبول کنید که درسی با این **نام** تا کنون ثبت نشده باشد.

```
AddCourse courseName credit
```

متد های کلاس دانشگاه

- اگر هر مشکلی اعم از نبود شماره دانشجویی، یا ارائه نشدن درس توسط این استاد یا هر مشکل این چنین دیگر وجود داشت، هیچ عملیاتی نباید انجام شود.

استاد درس را آموزش دهد:

- استاد این درس را ارائه میکند.

```
TeachCourse professorId courseName
```

دانشجو درس را بردارد:

- هر دانشجو یک درس را فقط یکبار و با یکی از استاد هایی که ان را درس میدهد بردارد (اگر درس را با استاد دیگری برداشته نمیتواند عوض کند)

```
TakeCourse studentId professorId courseName
```

متد های نمایش وضعیت

- توجه! تمامی موارد را باید به ترتیب اضافه شدن باید چاپ کنید.

اساتیدی که درس را ارائه میدهند:

- به فرمت خواسته شده، اساتیدی را که این درس را ارائه میکنند نمایش دهید.
- اگر خالی بود empty چاپ شود.

```
PrintCourseProfessors courseName
```

```
professor full name: ali alavi
professor full name: zohre zibaie
```

درس های داشنجو :

- باید به فرمت خواسته شده، همه درس ها به همراه استادی که آن را درس میدهد نمایش دهید.
- اگر خالی بود empty چاپ شود

```
PrintStudentCourses studentId
```

```
course name: math, credit: 3, professor full name: ali alav
course name: biology, credit: 2, professor full name: nima
```

ورودی و خروجی

در ورودی ابتدا عدد n و سپس در n خط بعدی دستوراتی که باید اجرا کنید داده میشود.

مثال

ورودی نمونه ۱

```
17
AddProfessor 9876 Andrew Anderson
AddProfessor 9877 Kelly Kale
AddStudent 12346 Jane Jana
AddStudent 12345 John johnson
AddCourse Math 3
AddCourse Physics 4
PrintCourseProfessors Math
PrintCourseProfessors Physics
PrintStudentCourses 403000001
TeachCourse 1001 Math
TeachCourse 1001 Physics
TakeCourse 403000002 1002 Math
TakeCourse 403000002 1001 Math
TakeCourse 403000002 1001 Physics
PrintCourseProfessors Math
PrintStudentCourses 403000001
PrintStudentCourses 403000002
```

خروجی نمونه ۱

```
empty
empty
empty
professor full name: Andrew Anderson
empty
course name:Math, credit: 3, professor full name: Andrew Anderson
course name:Physics, credit: 4, professor full name: Andrew Anderson
```

مسیجینگ

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: مهرسا سمیع‌زاده

در این سوال میخواهیم امنیت پیام های رد و بدل شده را تضمین کنیم. تصور کنید شما مسئول طراحی یک سیستم پیام‌رسانی برای یک گروه از کاربران هستید. در این سیستم، کاربران می‌توانند با ساخت یک لینک ارتباطی پیام‌های خود را با افراد مورد نظرشان به اشتراک بگذارند.

جزئیات برنامه

ابتدا پروژه اولیه را این لینک دانلود کنید.

- کاربر ها برای اشتراک گذاری پیام با یکدیگر باید حتما mailbox مشترک داشته باشند.
- هر mailbox میتواند شامل حداقل 2، کاربر باشد.

شما باید یک **سیستم پیام‌رسانی** طراحی کنید که شامل سه کلاس اصلی باشد:

کلاس `User`

این کلاس نمایانگر یک کاربر در سیستم است. هر کاربر می‌تواند پیام‌های خود را ارسال و دریافت کند، با این امکان که پیام‌ها می‌توانند به گروه‌های مختلف ارسال شوند.

کلاس دارای فیلد های زیر می‌باشد:

```
private String userName;
private Map<String, List<Message>> mailboxes;
```

فیلدها:

۱. `username` (نام کاربری): نام کاربری کاربر که به صورت `String` ذخیره می‌شود.

۲. mailboxes (صندوق پیام‌ها): یک Map که کلید آن نام گروه و مقدار آن لیستی از پیام‌ها (Message) است.

توابع:

۱. **سازنده:** User(String username) : برای مقداردهی اولیه نام کاربری و ایجاد یک Map برای صندوق پیام‌ها.

```
public User(String username){
    //TODO
}
```

۲. **ارسال پیام:** پیام را به سیستم پیام‌رسان ارسال می‌کند. کاربران برای ارسال پیام از این تابع استفاده میکنند، اما از انجایی که دسترسی به دیگر کاربران ندارند، باید از سیستم برای انجام کامل روند، کمک بگیرید. با فراخوانی این متد توسط کاربر، Messaging System باید با گرفتن لیست یوزرنیم کاربران، روند ارسال پیام را کنترل کند.،

```
public boolean send(MessagingSystem system, String message, List
    //TODO
}
```

۳. **دریافت پیام از یک گروه:** پیام‌های گروه مشخص‌شده را که کاربر نخوانده است دریافت می‌کند. در صورتی که کاربر عضوی از ان mailbox نبود User is not authorized to access this mailbox. ریترن شود. درصورتی که پیامی وجود نداشت، و یا هیچ پیام خوانده نشده ای وجود نداشت No new messages in this mailbox. برگردانده شود. توجه کنید که هرپیام در یک خط قرار گیرد.

```
public String receiveMailbox(String groupName){
    //TODO
}
```

۴. **دریافت از یک کاربر:** تمامی پیام‌های خوانده نشده از ان کاربر را نمایش میدهد .

```
public String receiveSender(String Sender){
    //TODO
}
```

۵. برای کاربر یک mailbox جدید اضافه میکند و به یک mailbox پیام میگذارد.

```
public void addMailbox(String groupName){
    //TODO
}
public void addMessage(String mailboxID,Message message){
    //TODO
}
```

کلاس MessagingSystem :

این کلاس وظیفه مدیریت گروه‌ها و ارسال پیام‌ها به کاربران را بر عهده دارد. پیام‌ها می‌توانند از سمت کاربر به گروه ارسال شوند و ویژگی‌هایی مانند دریافت پیام‌های عمومی یا خصوصی برای هر کاربر موجود است.

فیلدها:

۱. `users` (کاربران): یک `Map` که کلید آن نام کاربری و مقدار آن یک شیء از کلاس `User` است.
۲. `mailboxes` (گروه‌ها): یک `Map` که کلید آن نام گروه و مقدار آن لیستی از اعضای گروه است.

توابع:

۱. **سازنده:** برای مقداردهی اولیه Map کاربران و گروه‌ها.

```
public MessagingSystem() {
    //TODO
}
```

۲. **افزودن کاربر:** یک کاربر جدید به سیستم اضافه می‌کند.

```
public boolean addUser(String username) {
//TODO
}
```

۳. **ایجاد گروه:** گروهی با اعضای مشخص ایجاد می‌کند و صندوق پیام‌ها را برای هر کاربر گروه تنظیم می‌کند.

```
public boolean createMailbox(String groupName, List<String> memb
    //TODO
}
```

۴. **ارسال پیام:** پیام رمزنگاری‌شده را به mailbox مشخص‌شده ارسال می‌کند.

```
public boolean sendMessage(String sender,String mailboxID, Strin
    //TODO
}
```

۵. پیام مشخص شده را به لیست کاربران مشخص شده ارسال میکند. توجه کنید تنها در صورتی پیام ارسال میشود و به میل باکس انها اضافه میشود، که mailbox ای وجود داشته باشد که اعضای ان دقیقا اعضای لیست ارسالی و کاربری که درخواست ارسال پیام را دارد، باشند. و در غیر این صورت ، mailbox جدید ساخته نمیشود، و false ریترن میشود.

```
public boolean sendMessageToRecipients(String sender, List<Strin
    //TODO
}
```

کلاس Message

این کلاس نمایانگر یک پیام رمزگذاری‌شده است که قابلیت رمزگذاری و رمزگشایی را دارد. همچنین، می‌تواند بررسی کند که آیا پیام خوانده شده یا حذف شده است و از ویژگی‌هایی مانند خصوصی بودن پیام پشتیبانی می‌کند.

فیلدها:

۱. `sender` (فرستنده): نام کاربری فرستنده پیام.

۲. `recipients` (گیرندگان): لیستی از گیرندگان پیام.

۳. `messageText` (متن رمزنگاری‌شده): پیام رمزنگاری‌شده به صورت `String` .

۴. `private Map<String, Boolean> readStatus` اطلاعات مربوط به خوانده شدن پیام توسط هرکدام از دریافت کنندگانش.

توابع:

۱. **سازنده:** پیام را رمزنگاری کرده و فیلدهای لازم را مقداردهی می‌کند.

```
public Message(String sender, String message, List<String> recip
    //TODO
}
```

۲. **چک کردن خوانده شدن پیام:** چک میکند که ایا این کاربر این پیام را خوانده است.

```
public boolean isRead(String recipient) {
    //TODO
}
```

۳. **خوانده شدن:** ازین تابع برای تغییر وضعیت isRead استفاده میکنیم.

```
public void markAsRead(String recipient) {
    //TODO
}
```

۴. **برگرداندن پیام:** خود پیام را برمیگرداند.

```
public String getMessage() {
    //TODO
}
```

جاواگرام (2.0)

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: شایان حقیقت

پس از موفقیت بزرگ **جاواگرام**، شرکت فناوری اطلاعات **"اورین‌تک"** تصمیم گرفته است تا پیام‌رسان خود را برای حوزه‌های **حسابداری** و **بانکداری** ارتقا دهد. هدف این ارتقا، افزودن قابلیت‌های تخصصی به جاواگرام است تا بتواند نیازهای ارتباطی و مدیریتی این دو حوزه را به طور موثر پوشش دهد. این ارتقا شامل اضافه کردن ویژگی‌هایی مانند ارسال پیام‌های رمزنگاری‌شده، مدیریت حساب‌های کاربری با سطوح دسترسی متفاوت، و قابلیت‌های گزارش‌گیری مالی می‌باشد. اما در مرحله اول تنها به حساب های ساده اکتفا میکنیم .

ساختار کلی برنامه

ابتدا پروژه اولیه را از این لینک دانلود کنید.

برنامه‌ی ما شامل دو بخش اصلی است:

۱. **سرور (Server.java):** مسئول مدیریت ارتباطات، احراز هویت کاربران، و پردازش دستورات دریافتی از کلاینت‌ها.
۲. **کلاینت (Client.java):** نماینده کاربر نهایی است که از طریق آن می‌توان به سرور متصل شده و دستورات مختلفی را ارسال کرد.

این سیستم شامل قابلیت‌های زیر است:

- ثبت‌نام کاربران جدید
- ورود کاربران با اطلاعات قبلی
- اجرای عملیات ریاضی ساده
- خروج کاربران
- دستورات مدیریتی برای ادمین (خاموش کردن سرور و مشاهده کاربران)

کلاس Server

این کلاس وظیفه انجام محاسبات **ریاضی ساده** و **Signup**و **login** را به عهده دارد و دستورات **(خاموش کردن سرور و مشاهده کاربران)** تنها با ورود ادمین با کلید مخصوص امکان پذیر می باشد . سرور باید به گونه ای طراحی شود که با خروج کاربر سرور خاموش **نشود** .

پراپرتی ها

```
private static Map<String, String> users = new HashMap<>()
```

بخش `user` وظیفه: ذخیره اطلاعات کاربران **(کلید: نام کاربری، مقدار: رمز عبور)**.

```
private static boolean isRunning = true;
```

وظیفه نشان‌دهنده‌ی وضعیت روشن یا خاموش بودن سرور.

```
private static String currentUser = null
```

وظیفه ذخیره نام کاربری کاربری که در حال تعامل با سرور است.

متد ها

`main(String[] args)` (متد اصلی):

- وظیفه راه‌اندازی سرور و مدیریت اتصال کلاینت‌ها.
- پورت سرور **(port = 1239)** تعریف می‌شود .
- یک کاربر پیش‌فرض (ادمین) اضافه می‌شود :

```
users.put("admin", "adminpass");
```

- سرور راه‌اندازی می‌شود:

```
ServerSocket serverSocket = new ServerSocket(port);
```

+حلقه بی‌نهایت برای پذیرش کلاینت‌ها باید در نظر گرفته بشود :

```
Socket clientSocket = serverSocket.accept();
```

- دستورات دریافتی از کلاینت را پردازش می‌کند
- `LOGIN`
- `SIGNUP`
- `math` ، `logout` ، `serverdown` ، و غیره

`login(BufferedReader br, BufferedWriter bw)` (متد کمکی):

- وظیفه مدیریت فرآیند ورود کاربر.
- سرور درخواست "نام کاربری" می‌کند.

- کاربر نام کاربری خود را ارسال می‌کند.
- سرور درخواست "رمز عبور" می‌کند.
- اگر اطلاعات صحیح باشد:
- پیام خوش‌آمدگویی ارسال می‌شود.
- مقدار `true` برمی‌گردد.
- اگر اطلاعات نادرست باشد:
- پیام "Invalid creds. Bye." ارسال می‌شود.
- مقدار `false` برمی‌گردد.

```

private static boolean login(BufferedReader br, BufferedWriter

        bw.write("Username:");

    //TODO

        bw.write("Password:");

    //TODO

        String password = br.readLine();

        if (//TODO) {

            currentUser = username;

            bw.write("Welcome " + username + "!");

    //TODO

        } else {

            bw.write("Invalid creds. Bye.");
            //TODO

        }

    }
```

`signup(BufferedReader br, BufferedWriter bw)` (متد کمکی):

- وظیفه مدیریت فرآیند ثبت‌نام کاربر.
- سرور درخواست "نام کاربری جدید" می‌کند.
- اگر نام کاربری موجود باشد: **پیام "User exists. Bye." ارسال می‌شود.** و مقدار `false` برمی‌گردد.
- اگر نام کاربری جدید باشد:
- سرور درخواست "رمز عبور جدید" می‌کند.
- کاربر رمز عبور را ارسال می‌کند.
- پیام خوش‌آمدگویی ارسال می‌شود و مقدار `true` برمی‌گردد.

```
private static boolean signup(BufferedReader br, BufferedWriter

        bw.write("New username:");
    //TODO

        if (//TODO) {

            bw.write("User exists. Bye.");
            //TODO

        }

        bw.write("New password:");
    //TODO
        currentUser = newUsername;

        bw.write("Welcome " + newUsername + "!");

    //TODO

    }
```

`calculate(String expr)` (متد کمکی):

- وظیفه پردازش دستورات ریاضی.
- ورودی: یک رشته به شکل `math:<expression>` (مثلاً `math:2 + 3`).
- خروجی: رشته‌ای حاوی نتیجه محاسبات.

- اگر شامل + باشد: جمع انجام می‌شود.
- اگر شامل * باشد: ضرب انجام می‌شود.
- اگر فرمت اشتباه باشد یا ورودی عدد نباشد، پیام خطا برمی‌گرداند.

```
private static String calculate(String expr) {
    try {
        if (//TODO) {
            String[] parts = expr.split("\\+");
            if (parts.length != 2) return "Invalid format.";
//TODO
            return "Result: " + (a + b);
        } else if (//TODO) {
            String[] parts = expr.split("\\*");
            if (parts.length != 2) return "Invalid format.";
//TODO
            return "Result: " + (a * b);
        } else {
            return "Use + or *.";
        }
    } catch (NumberFormatException e) {
        return "Numbers only.";
    }
```

```
    }
```

نکات مهم این کلاس که می تواند کمک کننده باشد

- تعریف پورت سرور و اضافه کردن کاربر پیش‌فرض:

```
int port = 1239;
users.put("admin", "adminpass");
```

- راه‌اندازی سرور:

```
try (ServerSocket serverSocket = new ServerSocket(port)) {
    System.out.println("Server started on port " + port);
}
```

- پذیرش کلاینت و مدیریت دستورات:

مثال برای پردازش دستورات:

```
Socket clientSocket = serverSocket.accept();
BufferedReader br = new BufferedReader(new InputStreamReader(cli
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(cl
String command = br.readLine();

if (command.equalsIgnoreCase("LOGIN")) {
    login(br, bw);
} ....
```

کلاس Client

ویژگی‌ها و متغیرهای کلاس Client:

۱. **address (متغیر از نوع String):**

- وظیفه: ذخیره آدرس سرور.

- مقدار پیش‌فرض: `"localhost"` .

۲. **`port` (متغیر از نوع `int`):**

- وظیفه: ذخیره شماره پورت سرور.
- مقدار پیش‌فرض: `1239` .

متدها:

`main(String[] args)` (متد اصلی):

- وظیفه: مدیریت ارتباط با سرور و ارسال دستورات.

مراحل:

- اتصال به سرور

```
Socket socket = new Socket(address, port);
```

- خواندن پیام‌های سرور

```
String serverMsg = br.readLine();
```

- ارسال دستورات

```
String command = sc.nextLine();
bw.write(command);
bw.newLine();
bw.flush();
```

مثال:

```
Client Input:
SIGNUP
Server Output:
Welcome! Type LOGIN or SIGNUP:

Client Input:
```

```
john_doe
Server Output:
New username:

Client Input:
mypassword123
Server Output:
New password:

Server Output:
Welcome john_doe!

Client Input:
LOGIN
Server Output:
Welcome! Type LOGIN or SIGNUP:

Client Input:
john_doe
Server Output:
Username:

Client Input:
mypassword123
Server Output:
Password:

Server Output:
Welcome john_doe!

Client Input:
math:10 + 5
Server Output:
Result: 15

Client Input:
math:8 * 3
Server Output:
Result: 24

Client Input:
math:10 - 5
Server Output:
```

```
Use + or *.

Client Input:
math:two + three
Server Output:
Numbers only.

Client Input:
logout
Server Output:
Logged out.

Client Input:
LOGIN
Server Output:
Welcome! Type LOGIN or SIGNUP:

Client Input:
admin
Server Output:
Username:

Client Input:
adminpass
Server Output:
Welcome admin!

Client Input:
listusers
Server Output:
Users:
admin
john_doe

Client Input:
serverdown
Server Output:
Shutting down.
```

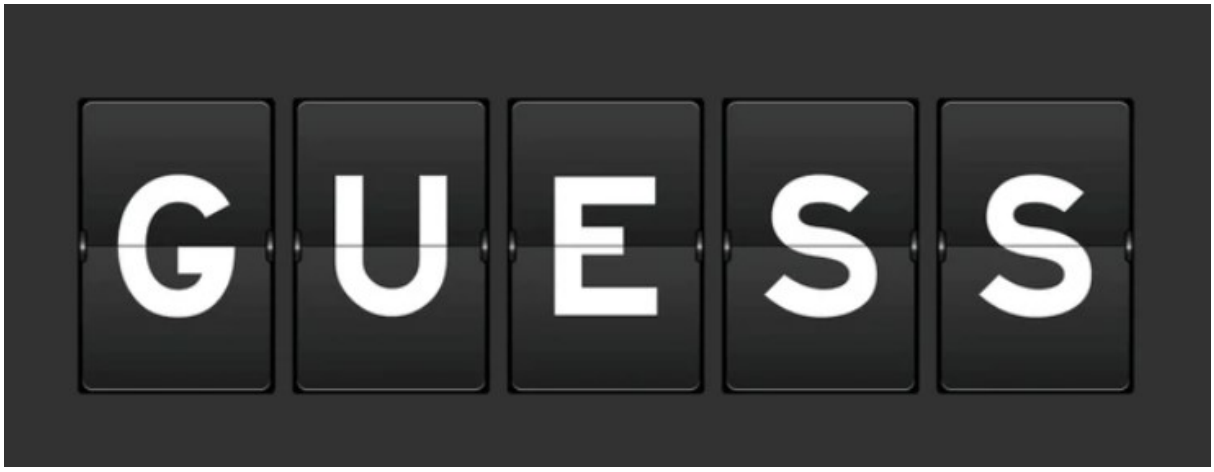
آنچه باید آپلود کنید

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

```
<zip_file_name.zip>
    ├── Server.java
    ├── Client.java
```

SocketWord (امتیازی)

- محدودیت زمان: ندارد
- محدودیت حافظه: ندارد
- سطح : سخت
- طراح : زهرا روشنی



شرکت بازی‌سازی "کلمه‌پرداز" در حال توسعه یک بازی آنلاین حدس کلمه است. این شرکت به تازگی تصمیم گرفته نسخه تحت شبکه بازی محبوب خود را با استفاده از معماری سرور-کلاینت پیاده‌سازی کند. به عنوان یک برنامه‌نویس جدید در این شرکت، وظیفه شما پیاده‌سازی بخش ارتباطات شبکه این بازی با استفاده از برنامه‌نویسی سوکت در جاوا است. شما باید یک بازی حدس کلمه با معماری سرور-کلاینت پیاده‌سازی کنید که:

جریان کلی بازی:

۱. سرور روی پورت مشخص شده راه‌اندازی می‌شود
۲. یک کلمه تصادفی انتخاب می‌شود
۳. منتظر اتصال کلاینت‌ها می‌ماند
۴. برای هر کلاینت:
   - پیام خوش‌آمدگویی ارسال می‌شود
   - طول کلمه اعلام می‌شود
   - تا 5 بار به کاربر اجازه حدس داده می‌شود

- برای هر حدس، بازخورد مناسب ارسال می‌شود
- در صورت حدس درست یا اتمام تلاش‌ها، بازی پایان می‌یابد

ویژگی‌های کلیدی:

۱. چند کاربره (Multi-threaded)
۲. مدیریت خطا
۳. بازخورد مناسب به کاربر
۴. محدودیت تعداد تلاش
۵. انتخاب تصادفی کلمات
۶. استفاده از سوکت برای ارتباط تحت شبکه

ابتدا پروژه اولیه را از این لینک دانلود کنید.

کلاس WordServer :

فیلد ها :

۱. `serverSocket` : سوکت سرور برای پذیرش اتصالات کلاینت‌ها
۲. `words` : لیستی از کلمات که بازیکن باید حدس بزند
۳. `currentWord` : کلمه فعلی که باید حدس زده شود
۴. `MAX_ATTEMPTS` : تعداد حداکثر تلاش‌های مجاز (5 بار)
۵. `isRunning` : مقدار ان تا زمانی که سرور روشن است true است

سازنده کلاس :

- یک سوکت سرور روی پورت مشخص شده ایجاد می‌کند
- لیست کلمات را مقداردهی اولیه می‌کند
- یک کلمه تصادفی انتخاب می‌کند

۱. `initializeWords()` :

```
private void initializeWords() {
    words = Arrays.asList("java", "python", "programming", "comp
                          "database", "socket", "server", "client

}
```

- لیست کلمات بازی را مقداردهی می‌کند
- یک کلمه تصادفی از لیست انتخاب می‌کند

۱. `selectRandomWord()` :

```
private void selectRandomWord() {
        //todo
    }
```

- یک کلمه تصادفی از لیست کلمات انتخاب می‌کند.

۱. `start()` :

```
public void start() {
        System.out.println("Server started. Waiting for clients.
        while (isRunning) {
            try {
                //todo
                System.out.println("New client connected");

                //todo

            } catch (IOException e) {

                if (isRunning) {
                   System.out.println("Error accepting client co
                }
            }
        }
    }
```

باید به این صورت پیاده‌سازی شود:

- در یک حلقه بی‌نهایت به دنبال اتصالات جدید می‌گردد
- برای هر اتصال جدید یک `ClientHandler` جدید ایجاد می‌کند
- هر کلاینت در یک thread جداگانه اجرا می‌شود

کلاس ClientHandler:

این کلاس مسئول مدیریت ارتباط با هر کلاینت است.

فیلد ها :

- `clientSocket` : سوکت اختصاصی برای ارتباط با کلاینت
- `out` : برای ارسال داده به کلاینت
- `in` : برای دریافت داده از کلاینت
- `attempts` : تعداد تلاش‌های انجام شده

متد `run` :

```
public void run() {
            try {
                out.println("Welcome to Word Guessing Game! You
                out.println("Word length: " + currentWord.length

                    // todo

            } catch (IOException e) {
                System.out.println("Error handling client: " + e
            } finally {
                try {

                    clientSocket.close();

                } catch (IOException e) {
                   System.out.println("Error closing client sock
                }
            }
        }
```

۱. ابتدا نیاز دارید حلقه اصلی بازی را بنویسید که حدس‌های کاربر را بخواند

۲. سپس باید شرایط مختلف بازی را بررسی کنید:

- آیا کاربر برنده شده؟ (حدس درست) : `Congratulations! You've won!`
- آیا تلاش‌ها تمام شده؟ (باخت) : `Game Over! The word was: {currentWord}`
- آیا بازی ادامه دارد؟ (نمایش تلاش‌های باقیمانده) :

`Attempts left: {تعداد تلاش باقی مانده}`

- بازی در دو حالت پایان می‌یابد:
  - کاربر برنده شود
  - تلاش‌ها تمام شود

متد `checkGuess` :

```
private String checkGuess(String guess) {
            //todo }
```

- متغیر های متد:
  - `guess` :ورودی متد، رشته‌ای که کاربر حدس زده است
  - `currentWord` : کلمه‌ای که باید حدس زده شود (متغیر عضو کلاس)
- بررسی اولیه :
  - ابتدا طول حدس کاربر با طول کلمه اصلی مقایسه می‌شود
  - اگر طول‌ها برابر نباشند، پیام خطا برمی‌گرداند:

`Invalid guess length. The word has  {currentWordLength} letters.`

- اگر طول حدس با کلمه برابر بود:
  - برای هر حرف در کلمه اصلی:
    - اگر حرف متناظر در حدس کاربر با حرف کلمه اصلی یکسان باشد: `T`
    - اگر حرف متناظر متفاوت باشد: `X`
- خروجی :
  - رشته‌ای به طول کلمه اصلی که شامل `T` و `X` است

- مثال:
  - کلمه اصلی: "java"
  - حدس کاربر: "jiva"
  - خروجی: "TTXT"

کلاس WordClient :

فیلدها :

- `Socket socket` : سوکتی که برای ارتباط با سرور استفاده می‌شود
- `PrintWriter out` : برای ارسال داده و پیام به سمت سرور
- `BufferedReader in` : برای خواندن داده و پیام‌های دریافتی از سرور
- `BufferedReader consoleInput` : برای خواندن ورودی کاربر از کنسول

متد `connect` :

- این متد مسئول برقراری اتصال با سرور است. وقتی فراخوانی می‌شود:
- یک سوکت جدید با آدرس و پورت مشخص شده ایجاد می‌کند
- کانال‌های ارتباطی ورودی و خروجی را راه‌اندازی می‌کند
- یک کانال برای خواندن ورودی کاربر از کنسول ایجاد می‌کند
- در صورت موفقیت پیام اتصال و در صورت خطا پیام خطا نمایش می‌دهد

متد `play` :

- این متد منطق اصلی بازی را پیاده‌سازی می‌کند:
- مرحله آماده‌سازی:
  - پیام خوش‌آمدگویی را از سرور دریافت و نمایش می‌دهد
  - طول کلمه مورد نظر را از سرور دریافت و نمایش می‌دهد
- حلقه اصلی بازی:
  - از کاربر می‌خواهد حدس خود را وارد کند
  - حدس را به سرور ارسال می‌کند
  - بازخورد سرور (الگوی T و X) را دریافت و نمایش می‌دهد

- پیام وضعیت بازی را دریافت و نمایش می‌دهد
- بررسی می‌کند آیا بازی تمام شده است یا خیر

- نمونه پیام ها :
- دریافت بازخورد از سرور : `Server Feedback: {text}`
- پیام وضعیت (برد / باخت / تعداد تلاش ها) : `Server: {text}`

نمونه ورودی خروجی :

```
Connected to server.
Welcome to Word Guessing Game! You have 5 attempts.
Word length: 8
Enter your guess: java
Server: Invalid guess length
Server: Attempts left: 5
Enter your guess: javajava
Server: XXXXXXXX
Server: Attempts left: 4
Enter your guess: computer
Server: TTTTTTTT
Server: Congratulations! You've won!
Game ended. Connection closed.
```

```
Connected to server.
Welcome to Word Guessing Game! You have 5 attempts.
Word length: 6
Enter your guess: javaja
Server: XXXXXX
Server: Attempts left: 4
Enter your guess: javaja
Server: XXXXXX
Server: Attempts left: 3
Enter your guess: javaja
Server: XXXXXX
Server: Attempts left: 2
Enter your guess: javaja
Server: XXXXXX
Server: Attempts left: 1
Enter your guess: javaja
Server: XXXXXX
Server: Game Over! The word was: socket
Game ended. Connection closed.
```