Palindrome Server

- محدودیت زمان: 4 ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: آسان
- طراح: مهدی کریمی

در این سوال قرار است به کمک client و server تشخیص بدهید آیا یک رشته پالیندروم هست یا خیر.

پالیندروم: رشته ای که از ابتدا و انتها یکسان خوانده می شود.

ابتدا client یک رشته به server ارسال می کند، سپس server باید تشخیص دهد که آیا این رشته palindrome هست یا خیر.(حروف بزرگ و کوچک تفاوتی ندارند و از space در جملات صرف نظر می شود.) اگر باشد YES و در غیر این صورت NO باز می گرداند.

فایل اولیه پروژه را می توانید از اینجا دانلود کنید.

کلاس PalindromeServer

```
public class PalindromeServer {
    public static final int PORT = 5000;
    public static void setupServer() {
        //TODO
    }
}
```

کلاس سرور است که مسوولیت بررسی palindrome بودن یا نبودن رشته دریافتی از client را برعهده دارد.

به طور دقیق تر در متد setupServer کار های زیر باید انجام شود: ایجاد سرور- گوش دادن برای اتصالات کلاینت- خواندن پیام از کلاینت و پردازش آن- بررسی اینکه آیا پیام یک Palindrome است یا نه و در نهایت ارسال پاسخ به کلاینت.

کلاس PalindromeClient

```java
public class PalindromeClient {
    private final String serverAddress;
    private final int serverPort;
    public PalindromeClient(String serverAddress, int serverPort
        //TODO
    }
    public String sendMessage(String message) {
        //TODO
    }
}
```

کلاس کلاینت 2 پراپرتی serverAddress و serverPort دارد که در مرحله تست به ترتیب مقادیر `localhost` و `5000` را دریافت خواهند کرد.

متد sendMessage:

این متد یک رشته دریافت و آن را به سرور ارسال می کند و درنهایت پاسخ سرور را به عنوان مقدار بازگشتی return می کند.

سنجش درستی:

درصورتی که کد زیر اجرا شود:

```java
public class Main {
    public static void main(String[] args) {
        Thread serverThread = new Thread(() -> PalindromeServer.
        serverThread.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        PalindromeClient client = new PalindromeClient("localhos
        String testMessage1 = "deed";
        String response1 = client.sendMessage(testMessage1);
        System.out.println(response1);
```

```
        String testMessage2 = "Never odd or even";
        String response2 = client.sendMessage(testMessage2);
        System.out.println(response2);
        String testMessage3 = "ali";
        String response3 = client.sendMessage(testMessage3);
        System.out.println(response3);
    }
}
```

خروجی مطابق زیر است:

```
YES
YES
NO
```

آنچه باید ارسال کنید:

فایل زیپ حاوی PalindromeClient و PalindromeServer

آزمون ورودی

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

پروژه اولیه را از این لینک دانلود کنید.

یک دانشگاه بزرگ تصمیم گرفته برای پذیرش دانشجویان، یک آزمون آنلاین برگزار کند.
این آزمون شامل سوالاتی از موضوعات متنوع مثل ریاضی، برنامه‌نویسی، منطق و تاریخ است.

هر سوال دارای چند گزینه است و فقط یکی از گزینه‌ها پاسخ صحیح می‌باشد.
علاوه‌براین، هر سوال متعلق به یک «دسته‌بندی موضوعی» است که **وزن مشخصی** دارد (مثلاً برنامه‌نویسی = ۳، ریاضی = ۲ و ...).

هر داوطلب فقط یک بار می‌تواند به هر سوال پاسخ دهد.
نمره‌ی سوالات درست با توجه به وزن موضوع آن سوال محاسبه می‌شود.
بر اساس نمره‌ها، لیست برترین شرکت‌کنندگان مشخص می‌گردد.

شما باید این سیستم را طراحی و پیاده‌سازی کنید.

شما باید **۴ کلاس** زیر را پیاده‌سازی کنید:

کلاس `Question`

نماینده‌ی یک سوال آزمون.

پراپرتی ها:

- `id` → شماره سوال (`int`)
- `text` → متن سوال (`String`)
- `choices` → لیست گزینه‌ها (`List<String>`)

- `correctAnswerIndex` → اندیس (index) گزینه صحیح (`int`)
- `category` → موضوع سوال (`QuestionCategory`)

سازنده:

```
public Question(int id, String text, List<String> choices, int c
```

متد :

```
public boolean isCorrect(int selectedIndex)
```

این متد درستی گزینه وارد شده را بررسی کرده و در صورت صحیح بودن `true` و در غیر این صورت `false` برمیگرداند.

کلاس `Participant`

نماینده‌ی یک داوطلب آزمون است.

پراپرتی‌ها:

- `id` → شناسه داوطلب (`int`)
- `name` → نام داوطلب (`String`)
- `answers` → نگهدارنده پاسخ‌ها برای هر آزمون به صورت:

```
Map<Integer, Map<Integer, Integer>>
// ExamID → (QuestionID → SelectedAnswer)
```

سازنده:

```
public Participant(int id, String name)
```

متد ها :

```
1 | public void submitAnswer(Exam exam, Question question, int selec
```

این متد پاسخ یک سوال را در یک آزمون مشخص ثبت میکند.

در صورتی که داوطلب در آزمون مورد نظر شرکت نکرده باشد باید استثنای `NotEnrolledException` و در صورتی که برای سوال مورد نظر قبلا پاسخ ثبت کرده باشد باید استثنای `AlreadySubmittedException` پرتاب شود. (استثنا ها در فایل اولیه پروژه قرار گرفته اند.)

```
1 | public void registerForExam(Exam exam)
```

این متد , آزمون مورد نظر را به `answers` داوطلب اضافه میکند. (به فرمت مپ `answers` که بالا تر توضیح داده شد توجه بفرمایید.)

```
1 | public int getScore(Exam exam) throws NotEnrolledException
```

این متد مجموع امتیاز داوطلب در یک آزمون را برمیگرداند. (با کمک لیست `questions` که در کلاس `Exam` وجود دارد و پایین تر توضیح داده شده.)

در صورتی که داوطلب در آزمون مورد نظر شرکت نکرده باشد باید استثنای `NotEnrolledException` (در فایل اولیه پروژه قرار گرفته) پرتاب شود.

توجه: متدهایی که از `getScore()` استفاده می‌کنند نیز ممکن است نیاز به مدیریت `NotEnrolledException` داشته باشند.

امتیاز هر سوال با توجه به `QuestionCategory` مروبطه متفاوت و به شرح زیر میباشد :

```
MATH → 2
PROGRAMMING → 3
LOGIC → 1
HISTORY → 1
LANGUAGE → 2
```

(در فایل اولیه پروژه تعریف شده اند.)

کلاس Exam

نماینده‌ی یک آزمون کامل است.

پراپرتی‌ها:

- `id` → شناسه آزمون (`int`)
- `title` → عنوان آزمون (`String`)
- `questions` → لیست سوالات آزمون (`List<Question>`)
- `participants` → لیست داوطلبان ثبت‌نام‌شده (`List<Participant>`)

سازنده:

```
public Exam(int id, String title)
```

متد ها :

```
public void addQuestion(Question question)
```

این متد Question داده شده را به لیست `questions` این آزمون اضافه میکند.

```
public void registerParticipant(Participant participant)
```

این متد داوطلب را به لیست `participants` اضافه میکند و همچنین این آزمون را به `answers` داوطلب اضافه میکند. (برای اضافه کردن آزمون به آزمون های داوطلب از متد `registerForExam(Exam exam)` در کلاس `Participant` استفاده کنید.)

کلاس ExamSystem

سیستم مدیریت چند آزمون و داوطلبان است.

پراپرتی‌ها:

- `exams` → لیست آزمون‌ها (`List<Exam>`)
- `participants` → لیست کل داوطلبان (`List<Participant>`)

سازنده:

```
public ExamSystem()
```

متد ها :

```
public void createExam(Exam exam)
public void registerParticipant(Participant participant)
```

متد `createExam` یک آزمون را به لیست همه ی آزمون ها (`exams`) اضافه میکند.

متد `registerParticipant` یک داوطلب را به لیست همه ی داوطلب ها (`participants`) اضافه میکند.

```
public void enrollParticipantInExam(Participant participant, Exa
```

این متد با استفاده از متد `registerParticipant` در کلاس `Exam` داوطلب را به آزمون مورد نظر و آزمون را به مپ داوطلب اضافه میکند.

```
public Map<Participant, Integer> ExamScoresMapGetter(Exam e)
```

این متد یک `Map<Participant, Integer>` برمیگرداند که شامل همه داوطلبان آزمون و نمره ی آنها است. (توجه داشته باشید اگر از متد های کلاس های دیگر که استثنا پرتاب میکنند استفاده کردید باید catch شوند و عملیات داخل بلاک catch در این سوال اهمیتی ندارد.)

```
public double getAverageScore(Exam exam) throws NotEnrolledExcep
```

این متد میانگین امتیاز همه داوطلبان در یک آزمون را تا دو رقم اعشار برمیگرداند.

```
1 | public List<String> getTopThreeParticipants(Exam exam)
```

این متد یک لیست مرتب شده (نزولی) از **نام** داوطلبان که شامل 3 نفر اول با بیشترین امتیاز در آن آزمون هستند را برمیگرداند.

توجه داشته باشید اگه دو داوطلب امتیاز یکسانی داشتند, داوطلبی که `id` کمتری دارد اولویت دارد.

اگر آزمون مورد نظر 2 داوطلب داشت ,یک لیست با نام همان دو نفر به ترتیب امتیاز و اگر 1 داوطلب داشت , یک لیست با نام آن داوطلب برگردانده میشود.

نکته: توجه داشته باشید که همه ی پراپرتی ها باید به صورت private تعریف شده باشند , نیازی به ستر (setter) نیست و گتر (getter) های مورد نیاز هر کلاس نیز در فایل پروژه اولیه مشخص شده اند. همچنین در صورت نیاز میتوانید متد های کمکی دیگری نیز پیاده سازی کنید.

اگر در متدی از متد های کلاس های دیگر که استثنا پرتاب میکنند استفاده کردید باید catch شوند و عملیات داخل بلاک catch در این سوال اهمیتی ندارد.

در فایل اولیه پروژه یک فایل `Main.java` قرار داده شده است تا درستی کد خود را تست بفرمایید, خروجی مورد نظر نیز به صورت کامنت در آخر فایل قرار گرفته.

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

```
├── ExamSystem.java
├── Exam.java
├── Participant.java
├── Question.java
├── QuestionCategory.java
├── NotEnrolledException.java
└── AlreadySubmittedException.java
```

تحلیل مسابقه برنامه‌نویسی

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- طراح: آریا صدیق

پروژه اولیه رو از این لینک دانلود کنید.

باشگاه الگوریتم دانشگاه در حال تحلیل داده‌های مربوط به یک مسابقه برنامه‌نویسی است. اطلاعات ارسالی شرکت‌کننده‌ها در قالب کلاس Submission ذخیره شده‌اند. مسئول تحلیل داده (یعنی شما) باید ابزارهایی برای تحلیل و پردازش این داده‌ها فراهم کنید.

کلاس پایه Submission:

```
public class Submission {
    private String participantName;
    private String problemName;
    private int score;

    public Submission(String participantName, String problemName
        this.participantName = participantName;
        this.problemName = problemName;
        this.score = score;
    }

    public String getParticipantName() { return participantName;
    public String getProblemName() { return problemName; }
    public int getScore() { return score; }
}
```

🧠متدهایی که باید پیاده‌سازی شوند:

شما باید کلاس ContestAnalyzer را طراحی کرده و متدهای زیر را پیاده‌سازی کنید:

1. calculateTotalScores(List<Submission> submissions)

هدف: محاسبه مجموع بهترین امتیاز هر شرکت‌کننده در هر سوال.

خروجی:

Map<String, IntegeR> (نام شرکت‌کننده ← مجموع امتیازها)

2. getTopNParticipants(Map<String, Integer> totalScores, int n)

هدف: بر اساس خروجی متد اول، n شرکت‌کننده با بیشترین امتیاز را به ترتیب نزولی امتیاز پیدا کن.

خروجی:

List<String> (فقط نام شرکت‌کننده‌ها، به ترتیب)

3. getProblemWiseTopScorer(List<Submission> submissions)

هدف: برای هر سوال، شرکت‌کننده‌ای که بالاترین امتیاز را دارد (در صورت تساوی، هر کدام کافی است).

خروجی:

Map<String, String> (نام سوال ← نام شرکت‌کننده)

4. getParticipantsWhoSolvedAll(List<Submission> submissions, Set<String> allProblems)

هدف: پیدا کردن شرکت‌کنندگانی که حداقل یک بار برای **همه سوالات موجود** (ورودی allProblems) ارسال داشته‌اند. (**ارسال‌ها مهم هستند نه امتیازات**)

خروجی:

Set<String> (نام شرکت‌کننده‌ها)

مثال

ورودی نمونه

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

```
        List<Submission> submissions = List.of(
            new Submission("Ali", "A", 50),
            new Submission("Ali", "A", 70),
            new Submission("Ali", "B", 30),
            new Submission("Sara", "A", 80),
            new Submission("Sara", "B", 60),
            new Submission("Reza", "A", 20),
            new Submission("Reza", "B", 10),
            new Submission("Reza", "C", 90),
            new Submission("Sara", "C", 85)
        );

        Set<String> allProblems = Set.of("A", "B", "C");

        Map<String, Integer> totalScores = ContestAnalyzer.calcu
        System.out.println("Total Scores: " + totalScores);

        List<String> top2 = ContestAnalyzer.getTopNParticipants(
        System.out.println("Top 2 Participants: " + top2);

        Map<String, String> topScorers = ContestAnalyzer.getProb
        System.out.println("Top Scorer Per Problem: " + topScore

        Set<String> completeParticipants = ContestAnalyzer.getPa
        System.out.println("Participants who solved all problems
    }
}
```

خروجی نمونه

```
Total Scores: {Ali=100, Reza=120, Sara=225}
Top 2 Participants: [Sara, Reza]
Top Scorer Per Problem: {A=Sara, B=Sara, C=Reza}
Participants who solved all problems: [Sara, Reza]
```

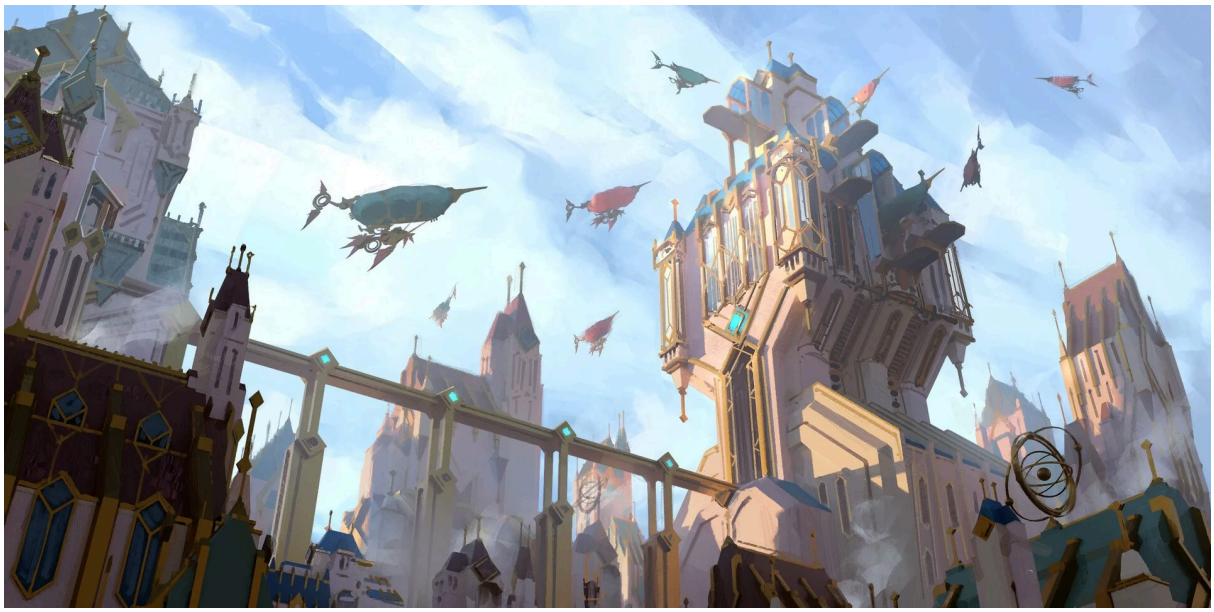
آنچه باید ارسال کنید:

فایل زیپ حاوی ContestAnalyzer.java

کتابخانهٔ پیلت اور

- سطح: سخت
- طراح: زهرا عزیزی

در یک دنیای موازی، *Heimerdinger* یک کتابخانه را در *Piltover* اداره می کند. اخیرا *Heimerdinger* تصمیم گرفته است تا مسئولیت ادارهٔ کتابخانه را به *Ekko* واگذاری کند تا خودش مشغول تحقیق درباره *Arcane* بشود. *Ekko* شما را به عنوان یک برنامه نویس شرافتمند از *Zaun* پیدا کرده است و تکمیل کردن کدهای سیستم مدیریت کتابخانه را به شما سپرده است. بر اساس دستورات داده شده، کدهای کتابخانه را تکمیل کنید.



پروژه اولیه را از اینجا دانلود کنید.

ساختار پروژه:

```
├── Book.java
├── BookNotAvailableException.java
├── BookNotFoundException.java
├── DuplicateBookException.java
├── Genre.java
└── Library.java
```

استثناها

BookNotAvailableException

زمانی که کتابی برای امانت گرفتن در دسترس نباشد یا به دلیل اینکه در امانت است در دسترس نباشد، این استثنا پرتاب می شود.

BookNotFoundException

زمانی که یک کتاب در کتابخانه یا کتاب های امانت گرفته شده پیدا نشود، این استثنا پرتاب می شود.

DuplicateBookException

زمانی که یک کتاب تکراری وارد کتابخانه بشود، این استثناء پرتاب می شود.

ژانرها

ژانرهای کتاب ها به صورت یک enum به شکل زیر در نظر گرفته شده است.

```
public enum Genre {
    CLASSIC,
    FANTASY,
    SCIENCE_FICTION,
    DYSTOPIAN
}
```

کلاس Book

کتاب های کتابخانه را پیاده سازی می کند.

Properties

```
    private int id;
    private String title;
    private String author;
    private Genre genre;

```

```
    private boolean isAvailable;
    private int timesBorrowed;
```

به ترتیب:

- شناسه کتاب
- عنوان کتاب
- نام نویسنده کتاب
- ژانر کتاب
- در دسترس بودن یا نبودن کتاب
- تعداد دفعاتی که کتاب امانت گرفته شده است.

Methods

- سازنده و تمامی getterها را پیاده سازی کنید. (در سازنده برای `isAvailable` مقدار اولیه true و برای `timesBorrowed` مقدار اولیه صفر را در نظر بگیرید.)
- تنها برای `isAvailable` متود `setAvailable` را پیاده سازی کنید. توجه کنید که اگر ورودی تابع false باشد به این معنی است که کتاب یکبار قرض گرفته شده است و باید به `timesBorrowed` یکی اضافه شود.

```
    @Override
    public String toString() {
        // TODO
    }
```

مشخصات کتاب را با فرمت زیر بر می گرداند.

```
Book ID: {id}, Title: {title}, Author: {author}, Genre: {genre}, Avai
```

مثال:

```
Book ID: 1, Title: 1984, Author: George Orwell, Genre: DYSTOPIAN, Avc
```

```
    @Override
    public boolean equals(Object obj) {
        // TODO
    }
```

برابری دو شی کتاب را بررسی می کند.

- برابری شی ها (obj پاس داده شده و شی فعلی) را بررسی می کند و اگر برابر بودند true بر می گرداند.
- اگر obj پاس داده شده به متود null بود یا کلاس شی فعلی با کلاس obj یکی نبود false بر می گرداند.
- در صورت برابری شناسه کتاب ها true و در غیر اینصورت false بر می گرداند.

```
    @Override
    public int hashCode() {
        // TODO
    }
```

با توجه به شناسه کتاب، هش کد را می سازد. (راهنمایی)

کلاس Library

کتابخانه را پیاده سازی می کند.

Properties

```
    private Map<Integer, Book> libraryBooks;
    private Map<Integer, String> borrowedBooks;
    private Map<String, Set<Book>> booksByAuthor;
    private Map<Genre, Set<Book>> booksByGenre;
    private Set<String> borrowers;
    private Queue<BookRequest> bookRequests;
```

به ترتیب:

- کتاب های کتابخانه؛ هر شناسه به یک کتاب نگاشت شده است.

- کتاب های امانت گرفته شده؛ هر شناسه به اسم یک امانت گیرنده نگاشت شده است.
- کتاب های یک نویسنده؛ اسم نویسنده به مجموعه کتاب هایش نگاشت شده است.
- کتاب های یک ژانر؛ ژانر به مجموعه کتاب های مختص خودش نگاشت شده است.
- مجموعه اسامی امانت گیرنده ها
- صف درخواست های داده شده

Methods

```
    public Library() {
        this.libraryBooks = new HashMap<>();
        this.borrowedBooks = new HashMap<>();
        this.booksByAuthor = new HashMap<>();
        this.booksByGenre = new HashMap<>();
        this.borrowers = new HashSet<>();
        this.bookRequests = new PriorityQueue<>(new Comparator<B
            @Override
            public int compare(BookRequest r1, BookRequest r2) {
                return Integer.compare(r1.getPriority(), r2.getP
            }
        });
    }
```

سازنده کلاس است!

```
    public void addBook(int id, String title, String author, Gen
        // TODO
    }
```

یک کتاب را به کتابخانه اضافه می کند. کتاب باید به کتاب های کتابخانه، کتاب های نویسنده خودش و کتاب های ژانر خودش اضافه شود. اگر کتاب تکراری بود یک استثنا `DuplicateBookException` با پیام `Book already exists in the library!` پرتاب می شود.

```
    public void removeBook(int id) throws BookNotFoundException,
        // TODO
    }
```

یک کتاب را از کتابخانه حذف می کند. کتاب باید از بین کتاب های کتابخانه، کتاب های نویسنده خودش و کتاب های ژانر خودش حذف شود. اگر با حذف کتاب، نویسنده ای یا ژانری بی کتاب ماند(!) باید از لیست booksByAuthor یا booksByGenre حذف شود. اگر کتاب در کتابخانه نبود یک استثنا BookNotFoundException با پیام Book was not found in the library. پرتاب می شود. اگر کتاب در حال حاضر در امانت بود یک استثنا BookNotAvailableException با پیام Book is currently borrowed. پرتاب می شود.

```
    public void borrowBook(int id, String borrower) throws BookN
        // TODO
    }
```

یک کتاب را به لیست کتاب های امانت گرفته شده اضافه می کند. کتاب از دسترسی خارج می شود، به لیست کتاب های امانت گرفته شده اضافه می شود و اسم امانت گیرنده به لیست امانت گیرنده ها اضافه می شود. اگر کتاب در کتابخانه نبود یک استثنا BookNotFoundException با پیام Book was not found in the library. پرتاب می شود. اگر کتاب در دسترس نبود یک استثنا BookNotAvailableException با پیام Book is currently unavailable for borrowing. پرتاب می شود.

```
    public void returnBook(int id) throws BookNotFoundException
        // TODO
    }
```

کتاب امانت گرفته شده را باز می گرداند. کتاب در دسترس قرار می گیرد، از لیست کتاب های امانت گرفته شده خارج می شود و اگر درخواستی برای این کتاب داده شده بود، درخواست بررسی می شود. اگر کتاب در بین کتاب های امانت گرفته شده نبود، یک استثنا BookNotFoundException با پیام Book was not found in the borrowed books list. پرتاب می شود.

```
    public void requestBook(String borrower, int id, int priorit
        // TODO
    }
```

یک کتاب را که در دسترس نیست، به صف درخواست ها اضافه می کند. اگر کتاب در کتابخانه نبود یک استثنا `BookNotFoundException` با پیام `Book was not found in the library.` پرتاب می شود.

```
1     private void processBookRequests(int id) {
2         // TODO
3     }
```

از بین کتاب های درخواست داده شده، کتاب هایی که شناسه داده شده را دارند و در دسترس هستند را به امانت می گذارد و از صف کتاب های درخواستی خارج می کند.

```
1     public void displayAvailableBooks() {
2         // TODO
3     }
```

کتاب های در دسترس را بر اساس عنوان (به ترتیب حروف الفبا) مرتب کرده و با فرمت زیر چاپ می کند. (از toString که در کلاس Book پیاده سازی کرده اید استفاده کنید)

```
Available Books:
Book ID: {id], Title: {title}, Author: {author}, Genre: {genre}, Avai
```

▼ راهنمایی

از Comparator برای مرتب کردن استفاده کنید.

▼ راهنمایی بیشتر!

به سازنده همین کلاس رجوع کنید.

مثال:

```
Available Books:
Book ID: 1, Title: 1984, Author: George Orwell, Genre: DYSTOPIAN, Ava
```

```
    public void displayBorrowedBooks() {
        // TODO
    }
```

کتاب های امانت گرفته شده را با فرمت زیر چاپ می کند. (از toString که در کلاس Book پیاده سازی کرده اید استفاده کنید)

```
Borrowed Books:
Book ID: {id], Title: {title}, Author: {author}, Genre: {genre}, Avai
```

مثال:

```
Borrowed Books:
Book ID: 2, Title: Crime and Punishment, Author: Fyodor Dostoevsky, (
```

```
    public Map<Integer, Book> getLibraryBooks() {
        return libraryBooks;
    }
```

کتاب های کتابخانه را بر می گرداند.

```
    public Map<Integer, String> getBorrowedBooks() {
        return borrowedBooks;
    }
```

کتاب های امانت گرفته شده را بر می گرداند.

```
    public List<Book> getBooksByAuthor(String author) {
        Set<Book> authorBooks = booksByAuthor.get(author);
        return authorBooks != null ? new ArrayList<>(authorBooks
    }
```

لیست کتاب های یک نویسنده را بر می گرداند.

```
    public List<Book> getBooksByGenre(Genre genre) {
        Set<Book> genreBooks = booksByGenre.get(genre);
        return genreBooks != null ? new ArrayList<>(genreBooks)
    }
```

لیست کتاب ها یک ژانر را بر می گرداند.

کلاس BookRequest

به صورت یک کلاس private داخل کلاس library تعریف شده است. این کلاس برای ایجاد درخواست برای امانت گرفتن کتاب استفاده می شود.

Properties

```
        private String borrower;
        private int bookId;
        private int priority;
```

به ترتیب:

- نام فرد امانت گیرنده
- شناسه کتاب درخواست داده شده
- اولویت در خواست؛ هرچقدر این عدد کمتر باشد، درخواست اولویت بیشتری دارد.

Methods

سازنده و تمامی getterها را پیاده سازی کنید. (نیازی به setterها نیست.)

تست

```
import java.util.*;

public class LibrarySystem {
    public static void main(String[] args) {
        Library library = new Library();
```

```
6
7           try {
8               library.addBook(1, "1984", "George Orwell", Genre.DY
9               library.addBook(2, "Crime and Punishment", "Fyodor D
10              library.addBook(3, "Animal Farm", "George Orwell", G
11              library.addBook(4, "The Hobbit", "J.R.R. Tolkien", G
12
13              System.out.println("=== AVAILABLE BOOKS ===");
14              library.displayAvailableBooks();
15
16
17              library.borrowBook(2, "Jinx");
18              library.borrowBook(3, "Vi");
19              System.out.println("\n=== BORROWED BOOKS ===");
20              library.displayBorrowedBooks();
21
22
23              library.returnBook(2);
24              library.returnBook(3);
25
26              library.borrowBook(2, "Ekko");
27              library.requestBook("Caitlyn", 2, 1);  // Higher pri
28              library.requestBook("Jayce", 2, 2);
29              library.returnBook(2);  // Should automatically go t
30
31              System.out.println("\n=== AFTER BOOK REQUESTS ===");
32              library.displayBorrowedBooks();
33
34
35              System.out.println("\n=== BOOKS BY GEORGE ORWELL ===
36              List<Book> orwellBooks = library.getBooksByAuthor("G
37              for (Book book : orwellBooks) {
38                  System.out.println(book);
39              }
40
41          } catch (Exception e) {
42              System.out.println("Error: " + e.getMessage());
43              e.printStackTrace();
44          }
45      }
46  }
```

در صورتی اجرای کد بالا، خروجی باید به صورت زیر باشد:

```
=== AVAILABLE BOOKS ===
Available Books:
Book ID: 1, Title: 1984, Author: George Orwell, Genre: DYSTOPIAN, Avc
Book ID: 3, Title: Animal Farm, Author: George Orwell, Genre: FANTASY
Book ID: 2, Title: Crime and Punishment, Author: Fyodor Dostoevsky, C
Book ID: 4, Title: The Hobbit, Author: J.R.R. Tolkien, Genre: FANTASY

=== BORROWED BOOKS ===
Borrowed Books:
Book ID: 2, Title: Crime and Punishment, Author: Fyodor Dostoevsky, C
Book ID: 3, Title: Animal Farm, Author: George Orwell, Genre: FANTASY

=== AFTER BOOK REQUESTS ===
Borrowed Books:
Book ID: 2, Title: Crime and Punishment, Author: Fyodor Dostoevsky, C

=== BOOKS BY GEORGE ORWELL ===
Book ID: 1, Title: 1984, Author: George Orwell, Genre: DYSTOPIAN, Avc
Book ID: 3, Title: Animal Farm, Author: George Orwell, Genre: FANTASY
```

آنچه باید بارگزاری کنید:

تمام فایل های پروژه را به صورت زیپ در آورده و بارگزاری کنید.

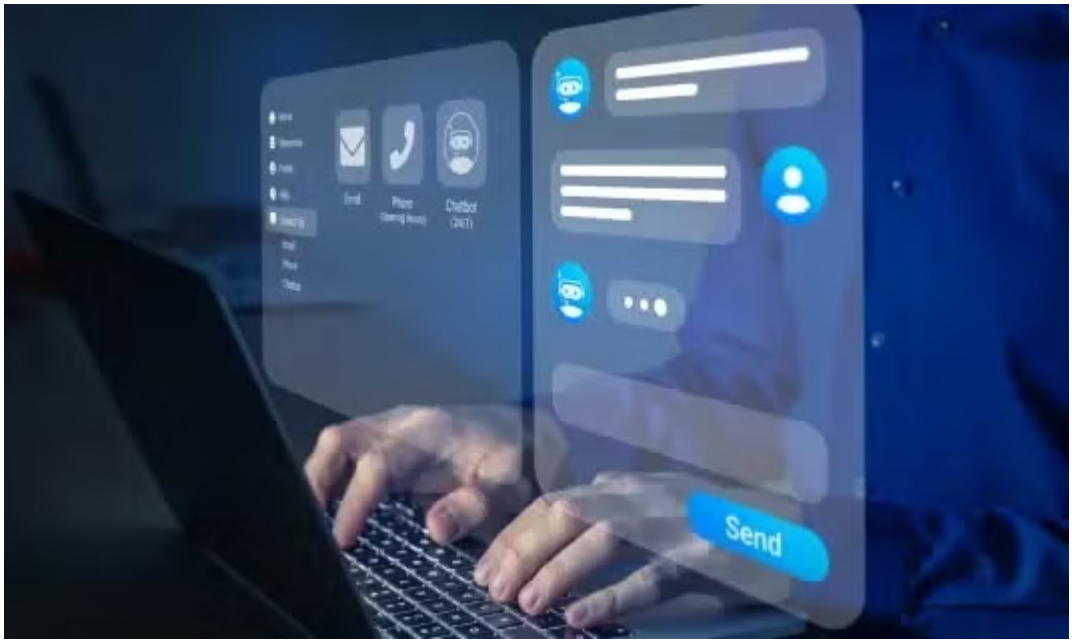
سرور چت بومی!!

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

پروژه اولیه رو از این لینک دانلود کنید.

توجه: داخل فایل خام پروژه مثالی وجود دارد. به تمامی موارد گفته شده داخل این مثال توجه کنید.

محمد شخصی است که به شدت به حفظ اطلاعات شخصی خودش در فضای مجازی اهمیت می‌دهد. در اخبار اخیر تلگرام که گفته شده‌است اطلاعات کاربران ممکن است در اختیار دولت‌ها قرار گیرد محمد کمی نگران اطلاعات شخصی خودش شده‌است. به همین دلیل او تصمیم می‌گیرد با مفاهیمی که تا اینجا از جاوا یاد گرفته‌است یک سرور چت بالا بیاورد تا از این طریق بتواند با دوستانش چت کند. هدف محمد در قدم اول صرفا ایجاد این سرور به منظور فرستادن پیام است. محمد فعلا قصد جابه‌جا کردن عکس و فایل‌هایش را با این روش ندارد. محمد در راستای ایجاد این برنامه به مشکل برخورد می‌کند و از آنجایی که می‌داند شما برنامه‌نویس ماهری هستید از شما خواسته است تا این برنامه را برای او بنویسید.



توجه : این برنامه شامل یک فایل `ChatServer.java` و `ChatClient.java` است و اگر می‌خواهید صحت کارکرد کد خود را داخل این برنامه متوجه شوید، میتوانید چند کاربر را بالا بیاورید و چندین پیام با

هرکدام بدهید. برای اینکار لازم است تا یک فایل جدید داخل همان پروژه جاوا ایجاد کرده و سپس کدهای داخل فایل `ChatClient.java` را داخل آن قرار دهید و آنرا هم اجرا کنید. توجه کنید که در زمان انتقال کدها باید اسم کلاس را به اسم فایل تغییر دهید زیرا همانطور که داخل کلاس آموختید اسم یک کلاس عمومی(public class) باید هم نام اسم فایل خودش باشد.

کلاس ChatClient:

توجه: این کلاس از قبل نوشته شده و نیازی به تغییر ندارد.

فیلدها:

```
1 | private static final String SERVER_ADDRESS = "localhost";
2 | private static final int SERVER_PORT = 12345;
```

۱. `SERVER_ADDRESS` : آدرس سرور که در اینجا `localhost` (یعنی همان کامپیوتر محلی) تنظیم شده است.

۲. `SERVER_PORT` : پورت سروری که کلاینت به آن متصل می‌شود. این پورت باید با پورت سرور چت که در فایل `ChatServer.java` مشخص شده باشد (پورت 12345) یکسان باشد.

متدها:

```
1 |     public static void main(String[] args) {
2 |         try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_P
3 |              BufferedReader in = new BufferedReader(new InputStr
4 |              PrintWriter out = new PrintWriter(socket.getOutputS
5 |              BufferedReader consoleInput = new BufferedReader(ne
```

۱. در این قسمت، ابتدا یک اتصال سوکت به سرور (`SERVER_ADDRESS` و `SERVER_PORT`) ایجاد می‌شود.

۲. از `BufferedReader` برای خواندن ورودی‌های دریافتی از سرور و ورودی‌های کاربر استفاده می‌شود.

۳. `PrintWriter` برای ارسال پیام‌ها به سرور استفاده می‌شود.

۴. `userInput` برای گرفتن ورودی از کاربر (برای ارسال پیام به سرور) ایجاد می‌شود.

۵. **ارسال و دریافت پیام‌ها:** پس از اتصال موفق به سرور، برنامه وارد یک حلقه می‌شود که به کاربر این امکان را می‌دهد که پیام‌هایی را تایپ کند و به سرور ارسال کند:

```
String message;
// Read the user input until a null input (end of stream)
while ((message = userInput.readLine()) != null) {
 out.println(message);  // Send the message to the server
 System.out.println("Message sent: " + message);  // Displa
 System.out.println("Server response: " + in.readLine());
}
```

- حلقه اصلی این برنامه این گونه است که منتظر می‌ماند تا کاربر پیامی را وارد کند. پس از تایپ هر پیام، آن را به سرور ارسال می‌کند.
- سپس پساخ سرور را دریافت کرده و نمایش می‌دهد.
- **پایان اتصال**: پس از اتمام پیام‌ها و یا قطع اتصال، سوکت و منابع مرتبط بسته می‌شوند.

کلاس ChatServer:

فیلدها:

```
    private static final int PORT = 12345;
    private static final HashMap<String, PrintWriter> clients =
```

توجه:در اینجا، یک ثابت `PORT` برای پورت سرور (که 12345 است) و یک `HashMap` به نام `clients` تعریف شده که برای نگهداری اطلاعات کلاینت‌ها (نام کاربری به عنوان کلید و `PrintWriter` برای ارسال داده به کلاینت‌ها به عنوان مقدار) استفاده می‌شود.

متدها:

```
public static void main(String[] args) {
    System.out.println("Chat server started...");
    try (ServerSocket serverSocket = new ServerSocket(PORT)) {

```

```
        while (true) {
            Socket socket = serverSocket.accept();
            new ClientHandler(socket).start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

۱. **`System.out.println("Chat server started...");`** : این خط برای نمایش پیامی است که نشان می‌دهد سرور چت با موفقیت راه‌اندازی شده است. این پیام به کنسول چاپ می‌شود تا کاربر بداند که سرور آماده است.

۲. **`ServerSocket serverSocket = new ServerSocket(PORT);`** : در اینجا یک شیء از کلاس `ServerSocket` ساخته می‌شود که به پورت 12345 متصل است. این شیء مسئول گوش دادن به اتصالات ورودی از طرف کلاینت‌ها است.

- **`PORT`** ثابت است که به پورت 12345 اشاره دارد. این پورت باید با پورت سرور در فایل `ChatClient.java` همخوانی داشته باشد تا ارتباط برقرار شود.

۳. **`serverSocket.accept();`** : این متد متوقف می‌شود و منتظر می‌ماند تا یک کلاینت به سرور متصل شود. زمانی که یک کلاینت به سرور متصل می‌شود، این متد یک شیء `Socket` را برمی‌گرداند که ارتباط برقرار شده را نمایان می‌کند.

- **`Socket socket`** : این شیء نشان‌دهنده ارتباط بین سرور و کلاینت است. از این شیء برای ارسال و دریافت داده‌ها بین سرور و کلاینت استفاده می‌شود.

۴. **`new ClientHandler(socket).start();`** : پس از دریافت اتصال از طرف کلاینت، یک شیء از کلاس `ClientHandler` ساخته می‌شود که مسئول مدیریت ارتباط با این کلاینت خواهد بود. سپس با استفاده از متد `start()` ، این شیء به یک نخ جدید تبدیل می‌شود و اجرای آن آغاز می‌شود.

- **`ClientHandler(socket)`** : این بخش، شیء جدیدی از کلاس `ClientHandler` را با ارسال شیء `Socket` به سازنده آن می‌سازد.
- **`.start()`** : متد `start()` باعث می‌شود که متد `run()` کلاس `ClientHandler` اجرا شود و عملیات مربوط به دریافت و ارسال پیام‌ها انجام گردد.

کلاس داخلی:

کلاس ClientHandler:

```
private Socket socket;
private String username;
private PrintWriter out;
private BufferedReader in;
```

۱. **`Socket socket`** : این فیلد شیء `Socket` است که ارتباط شبکه‌ای بین سرور و کلاینت را نشان می‌دهد. این شیء برای دریافت و ارسال داده‌ها بین سرور و کلاینت استفاده می‌شود.

۲. **`String username`** : این فیلد برای ذخیره نام کاربری کلاینت است. هر کلاینت باید یک نام کاربری منحصر به فرد داشته باشد.

۳. **`PrintWriter out`** : این فیلد برای ارسال داده‌ها (پیام‌ها) به کلاینت استفاده می‌شود.

۴. **`BufferedReader in`** : این فیلد برای دریافت داده‌ها (پیام‌ها) از کلاینت استفاده می‌شود.

سازنده کلاس ClientHandler:

```
public ClientHandler(Socket socket) {
    this.socket = socket;

}
```

- **`Socket socket`** : این پارامتر از نوع `Socket` است و زمانی که یک کلاینت به سرور متصل می‌شود، از طریق متد `accept()` از `ServerSocket` به دست می‌آید. این شیء به سرور این امکان را می‌دهد که بتواند داده‌ها را از کلاینت دریافت کرده و به آن ارسال کند.

متد run:

```
public void run() {
    //TODO
}
```

هدف متد `run()` در کلاس `ClientHandler` این است که پس از اتصال یک کلاینت به سرور، ارتباط با آن کلاینت را مدیریت کند. این متد عملیات‌هایی از جمله دریافت نام کاربری، ثبت آن در لیست کلاینت‌ها، ارسال پیام‌های ورودی به سایر کلاینت‌ها، و مدیریت قطع ارتباط را انجام می‌دهد.

1. ایجاد ورودی و خروجی برای ارتباط با کلاینت:

ابتدا، باید جریان‌های ورودی و خروجی (Streams) را برای ارتباط با کلاینت تنظیم کنید. این کار به شما این امکان را می‌دهد که پیام‌ها را از کلاینت دریافت کرده و به آن ارسال کنید.

2. دریافت نام کاربری از کلاینت:

سپس از کلاینت خواسته می‌شود که نام کاربری خود را وارد کند. اگر نام کاربری قبلاً توسط یک کلاینت دیگر استفاده شده باشد، باید یک خطا ایجاد کنیم تا نشان دهد نام کاربری تکراری است.

توجه: برای قسمت نام کاربری تکراری از `UsernameAlreadyTakenException` استفاده کنید. این فایل ساختار ساده‌ای به شکل زیر دارد:

```
1 public class UsernameAlreadyTakenException extends Exception {
2     public UsernameAlreadyTakenException(String username) {
3         super("The username '" + username + "' is already taken.
4     }
5 }
```

3. ثبت نام کاربری در لیست کلاینت‌ها:

در صورتی که نام کاربری معتبر باشد و تکراری نباشد، باید این نام کاربری را به لیست کلاینت‌ها اضافه کنیم. نام کاربری نباید رشته خالی باشد.

توجه:بررسی و اضافه کردن نام کاربری به لیست کاربران باید در یک بلوک synchronized انجام شود تا از شرایط رقابتی (race condition) جلوگیری شود، خصوصاً وقتی چند کلاینت هم‌زمان تلاش به ورود دارند.

4. ارسال پیام خوش‌آمدگویی به سایر کلاینت‌ها:

پس از ثبت نام کاربری، باید به سایر کلاینت‌ها اطلاع بدهیم که یک کاربر جدید به چت پیوسته است.

پیام خوش‌آمدگویی: این خط پیامی را به تمام کلاینت‌های متصل ارسال می‌کند که نشان می‌دهد یک کاربر جدید به چت پیوسته است : `username + " has joined the chat!"`

5. دریافت و پردازش پیام‌های کلاینت:

حالا که کلاینت به سرور متصل است، باید به طور مداوم منتظر دریافت پیام‌ها از او باشیم. هنگامی که پیام دریافت شد، باید آن را پردازش کنیم. اگر پیام "QUIT" باشد، ارتباط باید قطع شود.

پایان ارتباط با دستور "QUIT": اگر پیام دریافتی "QUIT" باشد، ارتباط با کلاینت قطع می‌شود.

6. مدیریت خطاها:

میدانیم که در رابطه با برنامه‌هایی که با سوکت کار می‌کنند، میتواند خطاهایی بروز دهد. یکی از این خطاها این است که یکی از `Client` ها از سیستم قطع شود و دیگر به سیستم دسترسی نداشته باشد (این حالت با حالتی که کاربر پیام `QUIT` را وارد می‌کند فرق دارد)، در این حالت باید پیام `" + username + " disconnected.` نمایش داده شود. این پیام در ترمینال سرور نمایش داده می‌شود.

7.مدیریت پیام‌ها:

دقت کنید که پیام‌ها میتوانند به صورت عمومی یا خصوصی داده شوند. اگر پیامی به صورت عادی نوشته شود، آن پیام برای تمامی اعضای فعال (که در آن لحظه به سرور متصل هستند) ارسال می‌شود.

برای ارسال پیام خصوصی هم توجه کنید که باید در ابتدا اسم مخاطب در فرمت: `@username` وارد شود. بعد از آن پیام دلخواه نوشته و ارسال شود. دقت داشته باشید که درصورتی که اسم وارد شده به سرور متصل نباشد هیچ اتفاقی نمی‌افتد و هیچ پیامی برای هیچ کسی ارسال نمی‌شود.

آنچه باید ارسال کنید:

تمام فایل های پروژه را به همراه فایل `UsernameAlreadyTakenException` آپلود کنید.