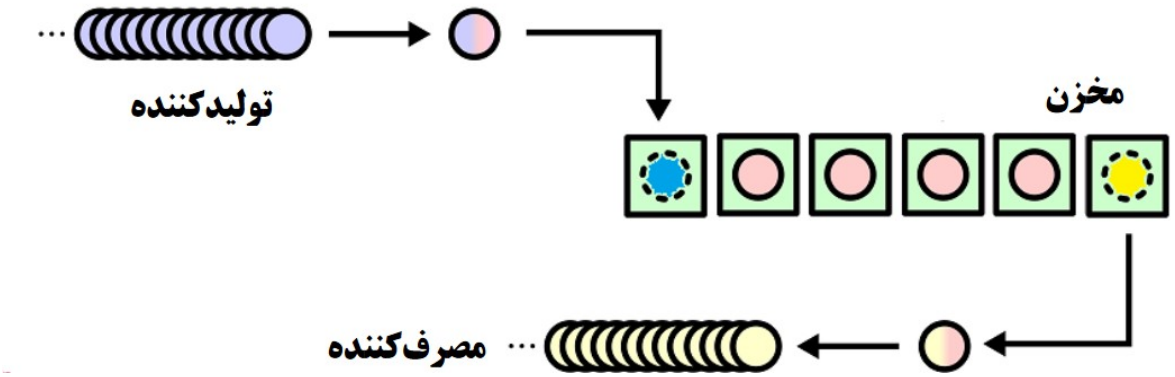
# از تولید به مصرف

- مباحث مرتبط: Concurrency, Multi-Threading

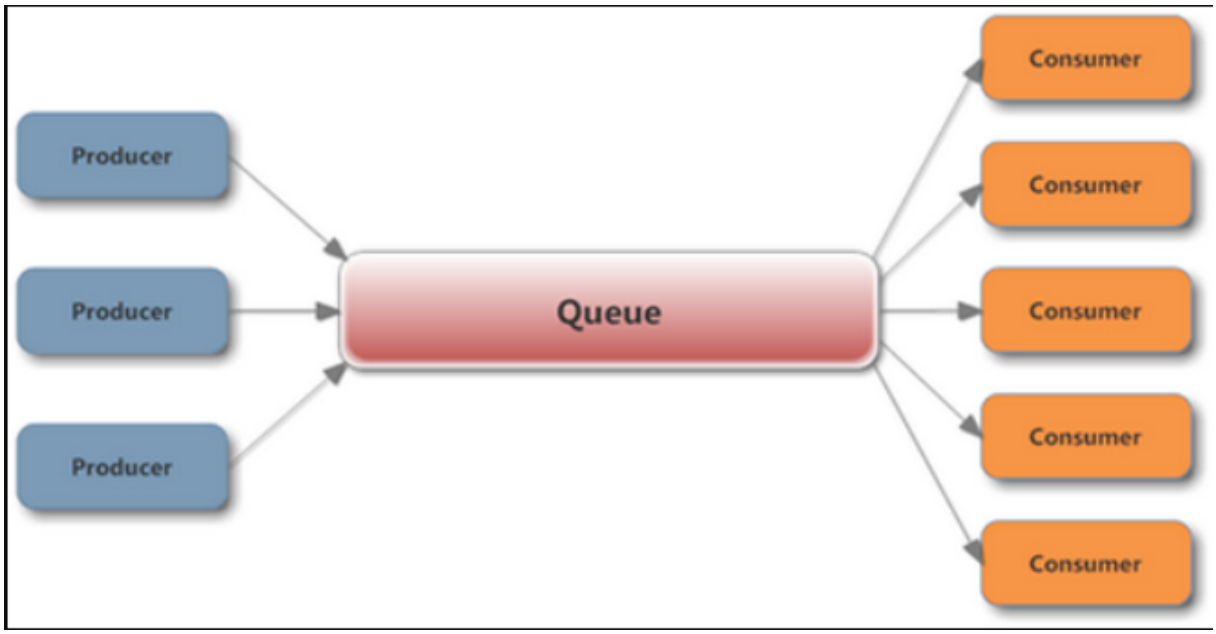
## مروری بر مسئله‌ی Producer-Consumer

اصول کلی این مسئله این است که یک یا چند Thread، یک نوع داده‌ای را **تولید** و یک یا چند Thread، آن داده‌ها را **پردازش** (مصرف) می‌کنند.




کلیه‌ی داده‌ها در یک مخزن مشترک جمع آوری می‌شوند و از طرفی دیگر، مصرف‌کننده آن داده‌ها را پردازش می‌کند. به زبان ساده، این مخزن مشترک را می‌توان یک ساختمان داده مثل صف (Queue) در نظر گرفت.

لازم به ذکر است که در این مسئله تعداد تولید کننده و تعداد مصرف‌کننده یا ظرفیت مخزن می تواند محدود باشد.مثلا به شکل زیر توجه کنید در اینجا سه تولید کننده و پنج مصرف کننده وجود دارد .




## جزئیات پروژه

بسته‌ی ir.javacup.thread را دانلود کنید. کلاس‌های Consumer و Producer را ببینید. شما باید متدهای set و get در کلاس Resource را به گونه‌ای پیاده‌سازی کنید که با اجرای کد زیر:

```
package ir.javacup.thread;

import java.util.concurrent.ConcurrentLinkedDeque;

public class Main {
    static final ConcurrentLinkedDeque<Integer> holder = new ConcurrentLinked

    public static void main(String[] args) throws InterruptedException {
        Resource reource = new Resource();
        Producer producer = new Producer(reource, holder, 10);
        Consumer consumer = new Consumer(reource, holder, 10);
        consumer.start();
        producer.start();
        consumer.join();
        producer.join();
        System.out.println(holder);
    }
}
```

خروجی دقیقاً به صورت زیر باشد:

```
[0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9]
```

## نکات

۱. دو Thread همزمان نباید با مخزن (شی مشترک) کار کنند، مثلا اگر یک Thread در حال تولید داده است، Thread مصرف کننده نباید به داده ها دسترسی داشته باشد تا زمانی که کار تولید کننده به پایان برسد یا به عبارت دیگری اگر یک Thread در حال خواندن یا نوشتن بر روی مخزن است، Thread دیگری نباید مزاحم شود.

۲. اگر مخزن مشترک خالی باشد، Thread مصرف کننده باید منتظر بماند (wait) تا Thread تولید کننده، داده ای جدید تولید کند (notify) و به همین ترتیب اگر طول مخزن محدود فرض شود و مخزن پر باشد Thread تولید کننده نباید داده ای تولید کند (wait) تا Thread مصرف کننده شروع به پردازش داده های مخزن کند (استفاده) و بلافاصله با آزاد شدن یک مکان از مخزن تولید کننده دوباره فعال شود.

## آن‌چه باید آپلود کنید

یک فایل زیپ شامل بسته‌ی ir.javacup.thread است. به صورتی که وقتی فایل زیپ را باز می‌کنیم، دقیقا شاخه‌ی ir را ببینیم که درون آن شاخه‌ی javacup و درون آن نیز شاخه‌ی thread قرار دارد. در داخل شاخه‌ی thread فقط فایل Resource.java وجود دارد.

# مرتب‌سازی حین خواب

- مبحث مرتبط: *Thread*

چندین سال پیش، الگوریتمی برای مرتب‌سازی اعداد به ذهن مریض برنامه‌نویسان خطور کرده بود. نام این الگوریتم، *sleep sort* است. این الگوریتم با استفاده از امکانات هم‌روندی زبان‌های برنامه‌نویسی کار می‌کند. نحوه‌ی اجرای این الگوریتم به این‌صورت است که به ازای هر عدد جهت مرتب‌سازی (که آن را $a_i$ می‌نامیم)، یک ترد در برنامه ساخته می‌شود که ابتدا به مدت $t \times a_i$ ثانیه می‌خوابد ( `sleep` ) و سپس $a_i$ را را به لیست مرتب‌شده اضافه می‌کند. در رابطه‌ی قبل، $t$ معمولاً عددی بین صفر تا ۱ (برحسب ثانیه) است، چون اگر $t$ عددی بزرگ باشد، مرتب‌سازی بیشتر طول می‌کشد. از طرفی اگر $t$ خیلی نزدیک به صفر باشد، به دلیل این که تردها لزوماً به‌صورت *parallel* اجرا نمی‌شوند (به‌صورت *concurrent* اجرا می‌شوند)، الگوریتم ممکن است برخی اعداد نزدیک به هم را به‌درستی مرتب نکند.

از شما می‌خواهیم برنامه‌ای برای مرتب‌سازی اعداد با استفاده از الگوریتم *sleep sort* بنویسید.

## جزئیات پروژه

پروژه‌ی اولیه را از این لینک دانلود کنید.

### کلاس `SleepSort`

این کلاس شامل یک متد استاتیک با نام `sort` است که لیستی از اعداد صحیح را دریافت می‌کند. این متد را طوری پیاده‌سازی کنید که اعداد را با الگوریتم *sleep sort* مرتب کرده و اعداد مرتب‌شده را در قالب یک لیست برگرداند.

## نکات

- برای مرتب‌سازی، حتماً باید از الگوریتم *sleep sort* استفاده کنید. این مورد به‌صورت دستی بررسی شده و اگر رعایت نشده باشد، نمره‌ای از این سؤال کسب نخواهید کرد.
- اعداد ورودی برای تست، مثبت و کوچک هستند.
- در صورت امکان، می‌توانید متدهای دیگری نیز در برنامه تعریف کنید.

## مثال

با اجرای متد `main` در کلاس `SleepSort` ، خروجی باید به‌صورت زیر باشد:

```
[1, 2, 3]
```

## آن‌چه باید آپلود کنید

پس از پیاده‌سازی متد `sort` ، فایل `SleepSort.java` را آپلود کنید.

# تعمیر کولر

- مباحث تحت پوشش: *concurrency* و *synchronization*
- سطح: متوسط

هوا کم‌کم در حال گرم شدن است و مردم به فکر راه‌اندازی کولرهایشان هستند. کولر شرکت *کدنویس‌گستران شرق به‌جز محمدرضا* اخیراً دچار مشکل شده و کارمندان باید به فکر تعمیر آن باشند، زیرا مدیرعامل شرکت به تعمیرکاران محلی اعتماد ندارد.

یک روند مشخص برای تعمیر کولر این شرکت در نظر گرفته شده است، به این‌صورت که هر کسی می‌تواند شروع به تعمیر کولر یا تست کولر بکند، اما **حداکثر یک نفر** می‌تواند مشغول تعمیر کولر باشد. این در حالی است که همه می‌توانند به‌صورت هم‌زمان به تست کولر بپردازند. نکته‌ی دیگری وجود دارد و آن این است که اگر یک نفر مشغول تعمیر کولر باشد، هیچ فردی نباید کولر را تست کند، زیرا ممکن است فردی که در حال تعمیر کولر است با برق‌گرفتی مواجه شود. افرادی که می‌خواهند کولر را تست کنند، باید صبر کنند تا فرایند تعمیر توسط فرد تعمیرکننده (در صورت وجود) خاتمه یابد. هم‌چنین اگر کسی در حال تست کولر باشد و فردی بخواهد به تعمیر کولر بپردازد، باید صبر کند تا تست کولر توسط فرد تست‌کننده خاتمه یابد.

از آن‌جایی که این فرایند برای کارمندان پیچیده است و می‌ترسند با برق‌گرفتگی مواجه شوند، آن‌ها تعمیر کولر را به‌صورت پنهانی به یک تعمیرکار محلی سپرده‌اند، زیرا می‌دانند که او کاردرست است؛ اما مدیرعامل شرکت از آن‌ها گزارشی شامل ترتیب شروع و پایان فرایندهای تعمیر یا تست هر نفر را از کارمندان خواسته است. برنامه‌ای برای کارمندان *کدنویس‌گستران شرق به‌جز محمدرضا* بنویسید که بتواند گزارش معتبری برای آن‌ها چاپ کند.

## پیاده‌سازی

فایل‌های اولیه‌ی برنامه را از این لینک دانلود کنید.

### کلاس `AirConditioner`

از این کلاس برای مدیریت تست و تعمیر کولر استفاده می‌شود. محتوای اولیه‌ی این کلاس به‌صورت زیر است:

```
public class AirConditioner {
    public void startTesting() {
        // TODO: Implement
    }

    public void finishTesting() {
        // TODO: Implement
    }

    public void startRepairing() {
        // TODO: Implement
    }

    public void finishRepairing() {
        // TODO: Implement
    }
}
```

فرایندهای تعمیر یا تست هر فرد در قالب یک ترد اجرا خواهند شد. مثلاً اگر ۳ نفر داشته باشیم که هر کدام بخواهند ۲ بار تست و ۱ بار تعمیر داشته باشند، ۳ ترد برای این کار ساخته خواهد شد.

یک نفر پیش از شروع تست کولر، متد `startTesting` را فراخوانی کرده و پس از اتمام تست، متد `finishTesting` را فراخوانی می‌کند. هم‌چنین، یک نفر پیش از شروع تعمیر کولر، متد `startRepairing` را فراخوانی کرده و پس از اتمام تعمیر، متد `finishRepairing` را فراخوانی می‌کند.

## کلاس Person

این کلاس از کلاس `Thread` ارث‌بری کرده و دارای این کانستراکتور است:

```
public Person(AirConditioner airConditioner, String name, int testsCount, int
```

با اجرای متد `start` این کلاس، عملیات تعمیر یا تست برحسب تعداد واردشده به‌صورت تصادفی اجرا می‌شود.

## کلاس Test

این کلاس شامل یک متد main به‌صورت زیر است:

```java
public class Test {
    public static void main(String[] args) {
        AirConditioner airConditioner = new AirConditioner();
        String[] names = {"Nima", "Mohammadreza", "Parsa"};
        for (String name : names) {
            (new Person(airConditioner, name, 2, 1)).start();
        }
    }
}
```

این متد باید پتانسیل چاپ **همه‌ی گزارش‌های معتبر** را داشته باشد. در غیر این‌صورت، نمره‌ی کامل سؤال را کسب نخواهید کرد (این سؤال به‌صورت دستی نیز بررسی خواهد شد).

یک خروجی معتبر برای این متد به‌صورت زیر است:

```
Nima started testing the air conditioner.
Nima finished testing the air conditioner.
Nima started repairing the air conditioner.
Nima finished repairing the air conditioner.
Nima started testing the air conditioner.
Mohammadreza started testing the air conditioner.
Nima finished testing the air conditioner.
Mohammadreza finished testing the air conditioner.
Mohammadreza started testing the air conditioner.
Mohammadreza finished testing the air conditioner.
Parsa started testing the air conditioner.
Parsa finished testing the air conditioner.
Parsa started repairing the air conditioner.
Parsa finished repairing the air conditioner.
Mohammadreza started repairing the air conditioner.
Mohammadreza finished repairing the air conditioner.
Parsa started testing the air conditioner.
Parsa finished testing the air conditioner.
```

یک خروجی نامعتبر برای این متد به‌صورت زیر است (در خط ۴، نیما ممکن است با برق‌گرفتگی مواجه شود):

```
Nima started testing the air conditioner.
Nima finished testing the air conditioner.
Nima started repairing the air conditioner.
Mohammadreza started testing the air conditioner.
Nima finished repairing the air conditioner.
Nima started testing the air conditioner.
Nima finished testing the air conditioner.
Mohammadreza finished testing the air conditioner.
Mohammadreza started testing the air conditioner.
Mohammadreza finished testing the air conditioner.
Parsa started testing the air conditioner.
Parsa finished testing the air conditioner.
Parsa started repairing the air conditioner.
Parsa finished repairing the air conditioner.
Mohammadreza started repairing the air conditioner.
Mohammadreza finished repairing the air conditioner.
Parsa started testing the air conditioner.
Parsa finished testing the air conditioner.
```

## راهنمایی

- در این سوال لازم نیست درگیر جزئیات `Person` شوید، بلکه از شما می‌خواهیم مسئله‌ی `reader-writers problem` را حل کنید. برای این منظور می‌توانید از چند `Mutex` یا `Semaphore` یا هر ساختار هم‌روندی دیگری که فکر می‌کنید مناسب است استفاده کنید. (برای پیدا کردن ساختار مناسب می‌توانید سرچ کنید یا اسلایدهای هم‌روندی‌پیشرفته را هم نگاه کنید)
- توجه کنید که در قسمت های finish باید فقط قفل‌ها آزاد شوند ولی کسی که می‌خواهد start کند (شروع به تعمیر یا شروع به تست) متد start کولر آنقدر بلاک می‌شود که شرایط مناسب پیش بیاید. مثلا start repair آنقدر بلاک می‌شود که هیچ کس دیگری مشغول تست یا تعمیر نباشد. متد start test هم آنقدر بلاک می‌شود که هیچ کس دیگری مشغول تعمیر نباشد.
- استفاده از متغیرهای معمولی برای نگهدار تعداد تسترهای فعال گزینه خوبی نیست چون این متغیرها برای محیط های هم‌روند ایمن نیستند و دچار race condition می‌شوند. در عوض باید از

متغیر های atomic استفاده کنیم.

- اگر خطای دریافت نتیجه یا خطای محدودیت زمان اجرا پیدا کردید احتمالا به این دلیل است که کدتان به deadlock میخورد.
- می‌توانید از این لینک استفاده کنید.

## آن‌چه باید آپلود کنید

پس از پیاده‌سازی کلاس `AirConditioner` ، فایل `AirConditioner.java` را آپلود کنید.

# Redis

- مباحث تحت پوشش: کار با ترد، سوکت، `Map` ، *stream* ، *serialization* و متغیر اتمیک
- سطح: متوسط و طولانی

شاید نام نرم‌افزار *Redis* را شنیده باشید. از این نرم‌افزار برای ذخیره‌ی مقادیری به‌صورت جفتِ کلید-مقدار و دسترسی به مقادیر کلیدها در زمان موردنیاز استفاده می‌شود. این نرم‌افزار از درهم‌سازی (*hashing*) برای نگاشت کلیدها به مقادیر استفاده می‌کند. *Redis* به‌صورت پیش‌فرض داده‌ها را در حافظه‌ی *RAM* نگه می‌دارد، اما هر چند وقت یک‌بار برای جلوگیری از نابود شدن اطلاعات، این اطلاعات را در دیسک ذخیره می‌کند (اصطلاحاً *consistent* است). در این تمرین، از شما می‌خواهیم نسخه‌ای ساده‌سازی‌شده از *Redis* را با جاوا پیاده‌سازی کنید.

## پیاده‌سازی

فایل‌های اولیه‌ی برنامه را از این لینک دانلود کنید.

### بسته‌ی `Models`

این بسته شامل یک اینترفیس با نام `Message` است. پیام‌هایی که بین سرور و کلاینت رد و بدل می‌شوند از نوع `Message` هستند. سرور با فراخوانی متد `handle` پیام، آن را پردازش می‌کند.

```
1 public interface Message extends Serializable {
2     void handle(ServerHandler serverHandler);
3 }
```

کلاس‌های زیر، این اینترفیس را پیاده‌سازی کرده‌اند. **پیاده‌سازی تمامی جزئیات این کلاس‌ها از قبل انجام شده است**:

- کلاس `ConnectMessage` : از این کلاس برای *initialize* کردن مقادیر مرتبط با کاربر و بررسی اتصال هم‌زمان استفاده می‌شود. این کلاس در کانستراکتور خود، نام کاربری کلاینت را در قالب یک رشته دریافت می‌کند.

- کلاس `DisconnectMessage` : از این کلاس برای بستن *input stream* و *output stream* کاربر در سمت سرور استفاده می‌شود. این کلاس هیچ کانستراکتوری ندارد.
- کلاس `ErrorMessage` : از این کلاس برای انتقال خطای اتصال هم‌زمان یک کاربر استفاده می‌شود. این کلاس در کانستراکتور خود، متن خطا را در قالب یک رشته دریافت می‌کند.
- کلاس `TextMessage` : از این کلاس برای انتقال پیام‌های متنی (نظیر پیام موفقیت‌آمیز بودن عملیات) استفاده می‌شود. این کلاس در کانستراکتور خود، متن پیام را در قالب یک رشته دریافت می‌کند.
- کلاس `CacheEntry` : از این کلاس برای ذخیره‌ی جفت کلید-مقدار استفاده می‌شود. این کلاس در کانستراکتور خود به‌ترتیب کلید و مقدار را در قالب دو رشته دریافت می‌کند.
- کلاس `GetKeyMessage` : از این کلاس برای دریافت مقدار کلید استفاده می‌شود. این کلاس در کانستراکتور خود کلید را در قالب یک رشته دریافت می‌کند. در کلاس `handle` ، سرور جفت کلید-مقدار را در قالب یک `CacheEntry` به کاربر ارسال می‌کند.
- کلاس `SetKeyMessage` : از این کلاس برای ست کردن مقدار یک کلید استفاده می‌شود. این کلاس در کانستراکتور خود به‌ترتیب کلید و مقدار را در قالب دو رشته دریافت می‌کند.

## بسته‌ی `connection`

این بسته شامل یک کلاس با نام `Connection` است. از این کلاس برای ایجاد ارتباط بین کلاینت و سرور استفاده می‌شود. با اتصال هر کاربر به سرور، یک نمونه از این کلاس تولید می‌شود. متدهای زیر را در کلاس `Connection` پیاده‌سازی کنید:

```
1 | public Connection(String username, String ip, int port)
```

این متد با دریافت نام کاربری کلاینت و آدرس و پورت سرور، یک `Socket` ساخته، یک پیام از نوع `ConnectMessage` به سرور ارسال کرده و *input stream* و *output stream* را برای استفاده‌های بعدی ذخیره می‌کند (برای این کار، لازم است تا فیلدهای جدیدی در کلاس تعریف کنید).

```
1 | public ObjectInputStream getInputStream()
```

این متد، *input stream* کانکشن در سمت کلاینت را برمی‌گرداند.

```
public ObjectOutputStream getOutputStream()
```

این متد، *output stream* کانکشن در سمت کلاینت را برمی‌گرداند.

```
public void sendMessage(Message message)
```

این متد با دریافت یک پیام، آن را در *output stream* می‌نویسد.

```
public void disconnect()
```

این متد، پیامی از نوع `DisconnectMessage` را به سرور ارسال می‌کند.

```
public Message getResponse()
```

این متد، آخرین پیام موجود در *input stream* را برمی‌گرداند.

## بسته‌ی `server`

این بسته شامل دو کلاس `Server` و `ConnectionHandler` است.

کلاس `ConnectionHandler` وظیفه‌ی مدیریت درخواست‌های یک کاربر توسط سرور را برعهده دارد. به ازای هر کاربر، یک نمونه از این کلاس ساخته می‌شود. متدهای زیر را در این کلاس پیاده‌سازی کنید:

```
public ConnectionHandler(ObjectInputStream inputStream, ObjectOutputStream ou
```

این متد با دریافت یک *input stream* و یک *output stream* ، آن‌ها را برای استفاده‌های بعدی ذخیره می‌کند (برای این کار، لازم است تا فیلدهای جدیدی در کلاس تعریف کنید).

```
public ObjectInputStream getInputStream()
```

این متد، *input stream* سمت سرور را برمی‌گرداند.

```
public ObjectOutputStream getOutputStream()
```

این متد، *output stream* سمت سرور را برمی‌گرداند.

```
public void sendMessage(Message message)
```

این متد با دریافت یک پیام، آن را در *output stream* می‌نویسد.

```
public void close()
```

این متد، *input stream* و *output stream* سمت سرور را می‌بندد.

کلاس `Server` اینترفیس `Runnable` را پیاده‌سازی کرده و وظیفه‌ی مدیریت `ServerSocket` را برعهده دارد. متدهای زیر را در این کلاس پیاده‌سازی کنید:

```
public void start(int port)
```

این متد با دریافت یک پورت، یک `ServerSocket` ایجاد می‌کند، اما قبل از انجام این کار، باید بررسی کند که آیا `ServerSocket` دیگری قبلاً ایجاد شده است یا نه. اگر چنین موردی وجود داشت، باید آن را ببندد و *cache* را خالی کند (برای این کار، می‌توانید فیلدهایی استاتیک در کلاس‌ها تعریف کنید). در نهایت، این متد یک `Thread` با ورودی یک `Server` جدید ساخته و آن را `start` می‌کند تا هندل کردن کلاینت‌ها آغاز شود.

```
public void run()
```

این متد تا زمانی که `ServerSocket` باز است، منتظر اتصال کاربر جدید می‌ماند و به ازای هر کاربر جدید، یک `Thread` ساخته و آن را `start` می‌کند.

## مثال

کلاسی با نام `Test` در فایل‌های اولیه‌ی برنامه قرار دارد که شامل یک متد `main` است. این متد، سرور را در پورت `6379` اجرا کرده، یک ترد برای چاپ کردن ورودی‌های سوکت اجرا کرده و در ترد اصلی، ورودی‌های اسکنر را هندل می‌کند. با اجرای این متد، می‌توان برنامه را تست کرد.

## نکات

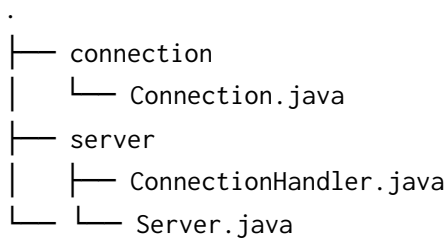
- سرور باید این توانایی را داشته باشد که به صورت هم‌روند به درخواست‌ها پاسخ دهد.
- باید با استفاده از ساختارهای مناسب جلوی *race condition* را بگیرید.
- از ساختمان‌داده‌های مناسب پردازش‌های موازی استفاده کنید.
- تنها ارتباط برنامه‌ی کلاینت و سرور از طریق سوکت خواهد بود و دسترسی مستقیم از کلاس‌ها به یکدیگر (مثلاً دسترسی از کلاس کلاینت به فیلدی در کلاس سرور) به هیچ‌وجه قابل قبول نیست و باعث صفر شدن نمره‌ی شما می‌شود. مثلاً فرض کنید برنامه‌ی سرور و کلاینت در دو سیستم مجزا اجرا می‌شوند.

## راهنمایی‌ها

- برای شروع می‌توانید از این لینک کمک بگیرید.
- برای نگه‌داری داده‌ها می‌توانید از مپ‌ها استفاده کنید. این مپ‌ها از نوع هم‌روند خواهند بود.

## آن‌چه باید آپلود کنید

پس از پیاده‌سازی کلاس‌ها، یک فایل زیپ آپلود کنید که وقتی آن را باز می‌کنیم، با دو پوشه‌ی `connection` و `server` مواجه شویم. درون پوشه‌ی `connection` باید فایل `Connection.java` قرار گرفته باشد و درون پوشه‌ی `server` ، باید دو فایل `Server.java` و `ConnectionHandler.java` قرار گرفته باشد.


.
├── connection
│   └── Connection.java
├── server
│   ├── ConnectionHandler.java
└── └── Server.java