بایت به بایت

- نام طراح: میثم باوی
- سطح سؤال: متوسط
- مبحث: الگوهای طراحی

الگوی طراحی Decorator به ما این اجازه را می‌دهد تا بدون ارث بری و حین اجرای برنامه (runtime)، عملکرد یک شئ را به دلخواه تغییر دهیم. در این سؤال می‌خواهیم با استفاده از این الگو، متد ها و ویژگی های جدید به یک *ByteReader* ساده اضافه کنیم. در مورد این الگو تحقیق کنید (پیشنهاد می‌کنم این مثال را هم ببینید).

پروژهٔ اولیه را از این لینک دانلود کنید.

رابط ByteReader

یک رابط ساده برای خواندن یک بایت داده از فایل، stream و ... است. تابع `read` یک بایت می‌خواند و به جلو می‌رود.

کلاس انتزاعی ByteReaderDecorator

این کلاس، والد هر کلاسی است که می‌خواهد یک *decorator* برای `ByteReader` باشد و به آن متدها و ویژگی های جدید اضافه کند. در سازندهٔ این کلاس، یک شئ `ByteReader` گرفته و در فیلد مربوطه ذخیره می‌شود.
دقت کنید که این کلاس خود رابط `ByteReader` را پیاده سازی می‌کند. رفتار متد `read` به صورت پیش فرض، همان رفتار شئ دریافت شده است و بعداً توسط کلاس های فرزند *override* می‌شود.

آنچه باید پیاده سازی کنید

کلاس ComplementReaderDecorator

- از کلاس ByteReaderDecorator ارث‌بری می‌کند.
- رفتار متد read آن به این صورت است که بایت اول رشته‌ی ورودی را می‌گیرد و بر عکس آن را برمی‌گرداند.
- متد اضافه‌ای ندارد.

```
10001111 -> 01110000
```

کلاس ReverseReaderDecorator

- از کلاس ByteReaderDecorator ارث‌بری می‌کند.
- رفتار متد read آن به این صورت است که برعکس بایت خوانده شده را بر می‌گرداند.
- متد اضافه‌ای ندارد.

```
10001111 -> 11110001
```

کلاس DataReaderDecorator

- از کلاس ByteReaderDecorator ارث‌بری می‌کند.
- رفتار متد read آن همان رفتار پیش فرض در کلاس والد است.

متد های اضافه

متد readChar

```
1 | public char readChar()
```

یک بایت با استفاده از متد read می‌خواند و به کاراکتر اسکی معادل آن تبدیل می‌کند و بر می‌گرداند.

متد readInt

```
1 | public int readInt()
```

چهار بایت با استفاده از متد `read` می‌خواند و به یک مقدار *int* تبدیل می‌کند و بر می‌گرداند. اولین بایت خوانده شده، پرارزش ترین بایت است.

متد `readBoolean`

```
1 | public boolean readBoolean()
```

یک بایت با استفاده از متد `read` می‌خواند و اگر معادل صفر بود `false` و در غیر این صورت `true` بر می‌گرداند.

متد `readLine`

```
1 | public String readLine()
```

تعداد بایت با استفاده از متد `read` می‌خواند تا به کاراکتر `\n` برسد. سپس رشته اسکی معادل بایت هایی که خوانده است را برمی‌گرداند.

- خود کاراکتر `\n` جزو رشته نیست.
- تضمین می‌شود هر خط حتما به یک `\n` ختم شود و تعداد کاراکتر ها حداکثر 30 باشد.

متد `readFully`

```
1 | public void readFully(byte[] b)
```

به اندازهٔ گنجایش آرایهٔ `b` ، با متد `read` از تعداد بایت می‌خواند و در همان ترتیب در آرایه ذخیره می‌کند.

نمونه

بایت به بایت کاراکتر های ورودی را می‌خوانیم و به داده های مورد نظر تبدیل می‌کنیم.

کد

```
DataReaderDecorator reader = new DataReaderDecorator(() -> {
    try {
        return (byte) System.in.read();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return (byte) 0;
});

System.out.println(reader.readInt());
System.out.println(reader.readChar());
System.out.println(reader.readLine());

byte[] bytes = new byte[5];
reader.readFully(bytes);

System.out.println(new String(bytes));
```

ورودی

```
o...=Hello World!
12345
```

خروجی

```
1865297454
=
Hello World!
12345
```

آنچه باید ارسال کنید

سه فایل ReverseReaderDecorator.java و ComplementReaderDecorator.java DataReaderDecorator را بدون هیچ پوشه‌ای در یک فایل زیپ ارسال کنید.

مارپیچ

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- نام طراح: امین ساوه
- سطح: متوسط

در این تمرین, از شما میخواهیم با دریافت عدد صحیح N آرایه ای $N * N$ تشکیل دهید و آن را با اعداد مثبت تا N^2 به صورت مارپیچ و در جهت عقربه های ساعت پر کنید و سپس آن را نمایش دهید.

ورودی

ورودی تنها شامل عدد صحیح N است.

$$1 \leq N \leq 20$$

خروجی

در خروجی آرایه $N * N$ مورد نظر باید چاپ شود.

مثال

ورودی نمونه ۱

```
3
```

خروجی نمونه ۱

1 2 3
8 9 4
7 6 5

ماشین‌حساب رشته‌ای

- محدودیت حافظه: ۲۵۶ مگابایت
- طراح: محمود چوپانی
- سطح سؤال: متوسط
- مبحث: رشته

به دنبال نبود ماشین‌حسابی برای رشته‌ها، از شما خواسته شده چنین چیزی طراحی کنید تا خلأ نبود این برنامه‌ی مهم پر شود.

ماشین‌حساب رشته‌ایِ ما از چندین عملیات مهم پشتیبانی می‌کند:

۱. `to_upper` : این دستور تمام کاراکترهای کوچک یک رشته را بزرگ می‌کند. مثال:

```
input:
to_upper i am mahmood.

output:
I AM MAHMOOD.
```

۲. `to_lower` : برعکس دستور قبل، تمام کاراکترهای بزرگ را کوچک می‌کند. مثال:

```
input:
to_lower i aM MahmooD.

output:
i am mahmood.
```

۳. `camel_case` : **شبیه** فرمت `camelCase` ، یک جمله را به این شکل در می‌آوریم. چند قانون بین خود می‌گذاریم:

- کاراکتر اول هر کلمه‌ای که قبلش کاراکتری غیر از کاراکترهای الفبای لاتین بود، باید بزرگ شود.
- سایر کاراکترهای هر کلمه (غیر از اولین کاراکتر) باید کوچک شوند.
- هر کاراکتری غیر از اعداد و الفبای انگلیسی باید حذف شود.
- کاراکتر اول جمله باید کوچک باشد.

مثال:

```
input:
camel_case Beautiful clouds in sky, will rain soon. at5aM

output:
beautifulCloudsInSkyWillRainSoonAt5Am
```

برای مطالعه بیشتر درباره‌ی `camelCalse` ، در اینترنت جست‌وجو کنید.

نکته: خیلی خوب است که از این قاعده برای نام‌گذاری متغیرهای جاوا استفاده کنید.

۴. `capital` : کاراکتر اول تمام کلمات در صورتی که کوچک بود، بزرگ شود. مثال:

```
input:
capital beautiful clouds in sky, will rain soon.

output:
Beautiful Clouds In Sky, Will Rain Soon.
```

نکته: برای تشخیص هر کلمه به این نکته توجه کنید که بین کلمات کاراکترهای عددی یا کاراکترهای غیر انگلیسی می‌آید.

۵. `encode` : این دستور یک رشته را مطابق الگوی رمزگذاری ما رمزگذاری می‌کند.

الگوی رمزگذاری ما:

- ابتدا رشته برعکس می‌شود.

- سپس کاراکترهایی که ایندکس زوج دارند به ترتیب در کنار هم و کاراکتر های با ایندکس فرد در کنار هم قرار می‌گیرند (ایندکس ها از `0` شروع می‌شوند).
- سپس این دو گروه را با استفاده از جدا کننده `#-#` در کنار هم قرار می‌گیرند (تضمین می‌شود در جمله وارد شده عبارت `#-#` وجود نداشته باشد).

مثال:

```
input:
encode Beautiful Clouds.

output:
.doClftaB#-#sul uiue
```

۶. `decode` : یک جمله‌ی رمزگذاری‌شده مطابق الگوی رمزگذاری ما را رمزشکنی می‌کند (تضمین می‌شود آخر جمله وارد شده *space* نداریم). مثال:

```
input:
decode .doClftaB#-#sul uiue

output:
Beautiful Clouds.
```

۷. `OFF` : با این دستور، اجرای برنامه به پایان می‌رسد.

تضمین می شود دو طرف جمله وارده اسپیس نداشته باشیم.

کتابخانه بهشتی

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- نام طراح: محمدسعید زارع
- سطح سوال: متوسط

سعید که بسیار کثیف کد میزند به دنبال کتابیست که با راهنمایی آن بتواند کد هایش را مرتب بنویسد. استادش به او کتاب clean code اثر رابرت مارتین را معرفی کرده. او که پول کافی برای خرید این کتاب را ندارد به کتابخانه دانشکده مراجعه میکند ولی از بدشانسیش کتابخانه آن کتاب را ندارد. حالا او باید با پای پیاده به کتابخانه تمام دانشکده ها سر بزند. برای جلوگیری از این موضوع، یک سیستم یکپارچه برای کتابخانه های دانشگاه بنویسید که سعید بتواند با یک سرچ سریع و بی دردسر کتاب مورد نظرش را پیدا کند.

برای بهتر درک کردن سوال، یکبار آن را تا انتها بخوانید و سپس شروع به کد زدن کنید.

اینام Type

در این `enum` انواع کتاب های در دسترس نوشته شده است.

۱. SCIENTIFIC
۲. CRIME
۳. FANTASY
۴. HORROR
۵. CLASSICS

کلاس Publisher

کلاس `Publisher` دارای ویژگی های زیر میباشد. `name` نام ناشر و `location` آدرس ناشر

```
private String name;
private String location;
```

کلاس Publisher دارای کانستراکتور زیر میباشد.

```
public Publisher(String name, String location){
    //TODO
}
```

کلاس Book

کلاس Book دارای ویژگی های زیر میباشد. name نام کتاب، type نوع کتاب، publisher ناشر کتاب و ویژگی borrowed نشاندهنده موجود بودن کتاب در کتابخانه است.

```
private String name;
private Type type;
private Publisher publisher;
private boolean borrowed;
```

کانستراکتور کلاس Book به این صورت است.

```
public Book(String name, Publisher publisher, Type type){
    //TODO
}
```

کلاس Member

کلاس Member دارای ویژگی های زیر میباشد. id کد ملی، name نام افراد و books کتاب هایی که امانت گرفته اند.

نکته: تضمین میشود هر کاربر بیشتر از 5 کتاب امانت نگیرد.

```
private String id;
private String name;
private Book[] books;
```

کلاس `Member` دارای کانستراکتور زیر است.

```
public Member(String name, String id){
    //TODO
}
```

کلاس Library

کلاس `Library` دارای ویژگی ها زیر است. `id` شناسه‌ی عددی است که با استفاده از آن می‌توان به آن کتابخانه دسترسی داشت، اولین کتابخانه ای که ساخته می‌شود شناسه‌ی عددی ۱، دومین کتابخانه، شناسه‌ی عددی ۲ و i-امین کتابخانه، شناسه‌ی عددی i دارد. `name` نام کتابخانه، `books` آرایه ای از کتاب های موجود در کتابخانه و `position` موقعیت کتابخانه را نشان میدهد.

```
private int id;
private String name;
private Book[] books;
private int position;
```

دانشگاه شهید بهشتی را به صورت یک خط از خوابگاه به میدان شهید شهریاری در بازه `[-500, 500]` در نظر بگیرد.

کانستراکتور این کلاس به این صورت است.

```
public Library(String name, int position){
    //TODO
```

```
}
```

این کلاس بهتر است حاوی متدهای زیر باشد:

۱. متدی برای جستجوی یک کتاب با نام آن کتاب
۲. متدی برای برگرداندن لیست کتاب های موجود
۳. متدی برای اضافه کردن کتاب
۴. متدی برای برگرداندن لیست کتاب ها بر اساس نوع کتاب

متد `toString` این کلاس را پیاده سازی کنید به این صورت که اگر کتابخانه ای با آیدی 2 و نام Markazi و موقعیت 10 و تعداد کتاب های موجود 200 عدد باشد، رشته زیر برگردانده شود:

```
(2, Markazi, 10, 200)
```

کلاس LibrariesHandler

`LibrariesHandler` مهم ترین کلاس این سوال میباشد و تست ها نیز بر روی متدهای این کلاس انجام خواهد شد پس با دقت متدها را پیاده سازی کنید **و یادتان باشد که امضای توابع دقیقا باید مثل الگویی که در قسمت های بعد می آوریم باشد.**

این کلاس باید از طراحی singleton پیروی کند و تنها نمونه آن از طریق متد getInstance قابل دسترسی باشد.

همچنین شامل آرایه ای از کتابخانه های دانشگاه شهید بهشتی است. (برای راحتی کار فرض میکنیم حداکثر 150 کتابخانه در دانشگاه شهید بهشتی وجود دارد.) و یک آرایه از افراد عضو شده که حداکثر گنجایش 5000 نفر را دارد. (فرض گرفتیم که سیستم یکپارچه است و اگر کسی یک بار ثبتنام کرد میتواند از هر کتابخانه ای که میخواهد کتاب قرض بگیرد.)

متدها

- **متد createLibrary**

این متد یک اسم و یک موقعیت را از ورودی دریافت میکند. اگر کتابخانه ای هم نام با این کتابخانه یا در موقعیت داده شده وجود داشته باشد کتابخانه جدید ساخته نمیشود.

```
public boolean createLibrary(String name, int position){
    //TODO
}
```

- متد addBook

این متد آیدی کتابخانه، نام، ناشر و نوع کتاب را دریافت میکند. و کتاب دریافت شده را به کتابخانه داده شده اضافه میکند.

```
public boolean addBook(int libId, String name, Publisher publisher, Type type){
    //TODO
}
```

- متد addMember

این متد نام و کدملی یک شخص را دریافت میکند. اگر شخصی با این کد ملی قبلا در سیستم وجود داشت ثبتنام انجام نمیشود.

```
public boolean addMember(String name, String id){
    // TODO
}
```

- متد getAllBooks

این متد از ورودی آیدی یک کتابخانه را دریافت میکند و تمام کتاب های موجود در آن را به ترتیبی که اضافه کرده بودیم به صورت زیر برمیگرداند. برای مثال اگر ابتدا کتاب `clean code` و سپس کتاب `the old man and see` و سپس کتاب `java` به لیست کتاب های کتابخانه با آیدی 1 اضافه شوند، خروجی متد به این صورت است.

به اینترهای بین هر خط توجه کنید.

```
1. clean code
2. the old man and see
3. java
```

نکته: اگر کتابخانه ای با این آیدی وجود نداشت، null برگردانید.

امضای این متد به این صورت است.

```
1 | public String getAllBooks(int libId){
2 |     // TODO
3 | }
```

- متد filterByType

این متد آیدی یک کتابخانه و نوع کتاب را از ورودی دریافت میکند و کتاب هایی که در این کتابخانه موجودند را بر اساس تایپ داده شده فیلتر میکند و لیست فیلتر شده را به فرمت زیر بر میگرداند.

```
1. clean code
2. java
3. python object oriented programming
```

امضای این تابع به این صورت است.

```
1 | public String filterByType(int libId, Type type){
2 |     // TODO
3 | }
```

در وصورت وجود نداشتن کتابخانه ای با آیدی داده شده، null برگردانید.

- متد borrow

این متد آیدی شخص و کتابخانه و نام یک کتاب را دریافت میکند و پس از بررسی در صورت عدم مشکل، کتاب قرض داده میشود در غیر اینصورت کاری انجام نمیشود.

```
public boolean borrow(String memberId, int libraryId, String name){
    //TODO
}
```

- **متد returnBook**

این متد نیز همانند متد borrow، آیدی شخص و کتابخانه و نام یک کتاب را دریافت میکند. و پس از بررسی در صورت عدم مشکل، کتاب به کتابخانه برگردانده میشود در غیر اینصورت کاری انجام نمیشود

```
public boolean returnBook(String memberId, int libraryId, String name){
    //TODO
}
```

- **متد size**

این متد تعداد کتابخانه های موجود در سامانه را برمیگرداند.

```
public int size(){
    // TODO
}
```

- **متد findNearestLibraryByPosition**

این متد نام یک کتاب و مختصات فعلی ما را دریافت میکند و نزدیک ترین کتابخانه ای را که این کتاب در آن موجود است به ما برمیگرداند.

اگر این کتاب در هیچ کتابخانه ای وجود نداشت، یک null برگردانده شود

نکته: اگر فاصله دو کتابخانه از ما برابر بود باید نام کتابخانه ای که آیدی کوچکتری دارد برگردانده شود.

```java
public Library findNearestLibraryByPosition(String name, int position){
    // TODO
}
```

- متد findLibrariesHaveBook

این متد نام یک کتاب و مختصات فعلی ما را دریافت میکند و یک لیست از کتابخانه هایی که این کتاب در آن موجود است به فرمت زیر برگرداند. **لیست خروجی باید بر اساس فاصله ما تا کتابخانه از کمترین فاصله تا بیشترین فاصله مرتب شده باشد.**

ابتدا شماره لیست، سپس نام کتابخانه و در اخر فاصله ما تا کتابخانه نمایش داده میشود.

1. Markazi 1
2. Computer 3
3. khabgah 20

امضای این متد به این صورت است.

```java
public String findLibrariesHaveBook(String name, int position){
    //TODO
}
```

- متد restart

این متد کلاس ها را به حالت اولیه برمیگرداند.

```java
public void restart(){
    // TODO
}
```

اگر کد شما به درستی کار کند با دادن ورودی زیر، خروجی زیر نیز باید چاپ شود.

ورودی

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

    LibrariesHandler lh = LibrariesHandler.getInstance();

    System.out.println(lh.createLibrary("Markazi", 100)); // true -> id = 1
    System.out.println(lh.createLibrary("elahiatLib", 100)); // false
    lh.createLibrary("ComputerLib", 0); // true -> id = 2
    lh.createLibrary("KhabgahLib", -300); // true -> id = 3
    System.out.println(lh.createLibrary("KhabgahLib", -200)); // false
    System.out.println("---");

    lh.addBook(2, "clean code", new Publisher("aria", "tehrann"), Type.SCIENTIFI
    lh.addBook(2, "kelile va demne", new Publisher("aria", "tehrann"), Type.FANT
    lh.addBook(2, "dalghak", new Publisher("aria", "tehrann"), Type.CRIME); //tr
    lh.addBook(2, "java", new Publisher("aria", "tehrann"), Type.SCIENTIFIC); //

    System.out.println(lh.addBook(1, "clean code", new Publisher("aria", "tehran
    System.out.println(lh.addBook(5, "sokoot", new Publisher("aria", "tehrann"),
    System.out.println(lh.addBook(1, "binaey", new Publisher("aria", "tehrann"),
    lh.addBook(1, "java", new Publisher("aria", "tehrann"), Type.SCIENTIFIC); //
    System.out.println("---");

    lh.addBook(3, "clean code", new Publisher("aria", "tehrann"), Type.SCIENTIFI
    lh.addBook(3, "eshgh", new Publisher("aria", "tehrann"), Type.FANTASY); // t
    lh.addBook(3, "koori", new Publisher("aria", "tehrann"), Type.CRIME); // tru
    lh.addBook(3, "cpp", new Publisher("aria", "tehrann"), Type.SCIENTIFIC); //

    System.out.println(lh.addMember("saeed", "123456"));
    System.out.println("---");

    System.out.println(lh.getAllBooks(1));
    System.out.println("---");


    System.out.println(lh.getAllBooks(2));
    System.out.println("---");

```

```
38      System.out.println(lh.findNearestLibraryByPosition("clean code", 0)); // Com
39      System.out.println("---");
40
41      System.out.println(lh.findNearestLibraryByPosition("clean code", -200)); //K
42      System.out.println("---");
43
44
45      System.out.println(lh.findLibrariesHaveBook("clean code", 30));
46      System.out.println("---");
47
48      lh.addMember("ali", "456789");lh.borrow("456789", 2, "clean code");System.ou
49      System.out.println("---");
50
51      lh.returnBook("456789", 2, "clean code");System.out.println(lh.findLibraries
52      System.out.println("---");
53
54      System.out.println(lh.filterByType(1, Type.SCIENTIFIC));
55  }
```

خروجی

```
true
false
false
---
true
false
true
---
true
---
1. clean code
2. binaey
3. java
---
1. clean code
2. kelile va demne
```

```
3. dalghak
4. java
---
ComputerLib
---
KhabgahLib
---
1. ComputerLib 30
2. Markazi 70
3. KhabgahLib 330
---
1. Markazi 70
2. KhabgahLib 330
---
1. ComputerLib 30
2. Markazi 70
3. KhabgahLib 330
---
1. clean code
2. java
```

نکات

- در صورت نیاز میتوانید متدها و فیلدهای جدیدی به کلاس ها اضافه کنید.
- تضمین میشود هر کتابخانه تنها یک نسخه از هر کتاب را داشته باشد.

آنچه باید آپلود کنید

فایل Main.java حاوی تمام کلاس های گفته شده را بدون هیچ پکیج بندی آپلود کنید.

سینما

- مباحث تحت پوشش: کار با فایل
- سطح: متوسط

یک سینما به‌تازگی سفارش یک سامانه‌ی فروش بلیت سینما را به شرکت *کدنویس‌گستران شرق به‌جز نیما* داده است. محمدرضا انجام این سفارش را بر عهده گرفته است، اما از آن‌جایی که هنوز کار با فایل را در جاوا یاد نگرفته، از شما می‌خواهیم این سفارش را انجام دهید تا محمدرضا قبل از ددلاین بتواند آن را تحویل دهد.

پیاده‌سازی

داده‌های مربوط به فیلم‌ها از قبل در یک فایل متنی به‌صورت مرتب بر اساس شناسه موجود هستند که مسیر آن در برنامه مشخص می‌شود. نمونه‌ای از این فایل را از این لینک دانلود کنید. بخشی از محتوای این فایل به‌صورت زیر است:

```
1    Toy Story    1995    http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)    Ani
2    GoldenEye    1995    http://us.imdb.com/M/title-exact?GoldenEye%20(1995)    Actio
3    Four Rooms    1995    http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)    T
4    Get Shorty    1995    http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)    A
5    Copycat    1995    http://us.imdb.com/M/title-exact?Copycat%20(1995)    Crime|Dra
6    Shanghai Triad (Yao a yao yao dao waipo qiao)    1995    http://us.imdb.com/Title
7    Twelve Monkeys    1995    http://us.imdb.com/M/title-exact?Twelve%20Monkeys%20(19
8    Babe    1995    http://us.imdb.com/M/title-exact?Babe%20(1995)    Children's|Come
9    Dead Man Walking    1995    http://us.imdb.com/M/title-exact?Dead%20Man%20Walking
10    Richard III    1995    http://us.imdb.com/M/title-exact?Richard%20III%20(1995)
```

اطلاعات هر فیلم در یک سطر از این فایل نوشته شده است. هر سطر شامل ۶ مقدار است که با کاراکتر `\t` از یکدیگر جدا شده‌اند. این مقادیر به‌ترتیب بیانگر موارد زیر هستند:

۱. شناسه‌ی فیلم

۲. عنوان فیلم

۳. سال انتشار فیلم

۴. آدرس صفحه‌ی *IMDb* فیلم

۵. ژانر(های) فیلم که با کاراکتر | از یکدیگر جدا شده‌اند

۶. تعداد بلیت‌های خریداری شده از فیلم

کلاس Movie

از این کلاس برای نگه‌داری اطلاعات یک فیلم استفاده می‌شود. بدنه‌ی این کلاس به‌صورت زیر است:

```
import java.util.*;

public class Movie {
    public final int id;
    public final String title;
    public final int releaseYear;
    public final String IMDbURL;
    public final List<String> genres;
    public int boughtTicketsCount;

    public Movie(int id, String title, int releaseYear, String IMDbURL, List<Str
        this.id = id;
        this.title = title;
        this.releaseYear = releaseYear;
        this.IMDbURL = IMDbURL;
        this.genres = genres;
        this.boughtTicketsCount = boughtTicketsCount;
    }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Movie)) {
            return false;
        }
        Movie m = (Movie) o;
```

```
        return this.id == m.id
                && this.title.equals(m.title)
                && this.releaseYear == m.releaseYear
                && this.IMDbURL.equals(m.IMDbURL)
                && this.genres.equals(m.genres)
                && this.boughtTicketsCount == m.boughtTicketsCount;
    }
}
```

کلاس CinemaManager

این کلاس، وظیفه‌ی مدیریت داده‌های مربوط به سینما را بر عهده دارد. متدهای زیر را در این کلاس پیاده‌سازی کنید:

```
public CinemaManager(String filePath);
```

این متد، کانستراکتور کلاس است که مسیر فایل اطلاعات فیلم‌ها را دریافت می‌کند.

```
public List<Movie> getMovies();
```

این متد، لیستی از همه‌ی فیلم‌ها را برمی‌گرداند.

```
public Movie getById(int id);
```

این متد، فیلمی که شناسه‌ی آن برابر با `id` است را برمی‌گرداند. در صورتی که چنین فیلمی یافت نشد، مقدار `null` را برگردانید.

```
public Movie getByTitle(String title);
```

این متد، فیلمی که عنوان آن برابر با `title` است را برمی‌گرداند. در صورتی که چنین فیلمی یافت نشد، مقدار `null` را برگردانید.

```
public boolean buy(int id);
```

این متد، تعداد بلیت‌های خریداری‌شده‌ی فیلمی که شناسه‌ی آن برابر با `id` است را یک واحد افزایش می‌دهد و مقدار `true` را برمی‌گرداند. اگر چنین فیلمی یافت نشد، مقدار `false` را برگردانید.

```
public boolean add(Movie movie);
```

این متد، فیلم جدیدی را به لیست فیلم‌ها اضافه کرده و مقدار `true` را برمی‌گرداند. اگر شناسه‌ی فیلم جدید تکراری بود، مقدار `false` را برگردانید.

```
public boolean delete(int id);
```

این متد، فیلمی که شناسه‌ی آن برابر با `id` است را حذف کرده و مقدار `true` را برمی‌گرداند. اگر چنین فیلمی یافت نشد، مقدار `false` را برگردانید.

مثال

با اجرای متد زیر (با فرض این که مسیر فایل اطلاعات فیلم‌ها `/home/CinemaManager/Movies.txt` است):

```
public static void main(String[] args) {
    CinemaManager cinemaManager = new CinemaManager("/home/CinemaManager/Movies.
    List<Movie> movies = cinemaManager.getMovies();
    System.out.println(movies.size()); // 60
    Movie movie = cinemaManager.getById(13);
    System.out.println(movie.title); // Mighty Aphrodite
```

```
    int boughtTicketsCount = movie.boughtTicketsCount;
    System.out.println(cinemaManager.buy(movie.id)); // true
    System.out.println(cinemaManager.buy(1000)); // false
    System.out.println(movie.boughtTicketsCount == boughtTicketsCount + 1); // t
    Movie modernTimes = new Movie(
            70,
            "Modern Times",
            1936,
            "https://www.imdb.com/title/tt0027977/",
            Arrays.asList("Comedy", "Classic"),
            103256
    );
    System.out.println(cinemaManager.add(modernTimes)); // true
    movies = cinemaManager.getMovies();
    System.out.println(movies.size()); // 60
    System.out.println(movies.get(movies.size() - 1).title); // Modern Times
    System.out.println(cinemaManager.getByTitle("Modern Times").genres.toString(
    System.out.println(cinemaManager.getById(404)); // null
}
```

خروجی باید به‌صورت زیر باشد:

```
60
Mighty Aphrodite
true
false
true
true
61
Modern Times
[Comedy, Classic]
null
```

نکات

- **تضمین می‌شود** که مسیر فایل ورودی معتبر است.

- تمامی تغییرات انجام‌شده توسط برنامه باید در فایل ذخیره شوند.
- اطلاعات فیلم‌ها پس از اعمال تغییرات نیز باید بر حسب شناسه به‌صورت صعودی مرتب شده باشند.

آن‌چه باید آپلود کنید

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

Distance decoding

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- طراح سوال: مبینا شهبازی

ابتدا پروژه اولیه را از این لینک دانلود کنید. در این سوال از شما می خواهیم به کمک socket programming یک رمزنگار طراحی کنید به طوری که کاربران پیام رمزگذاری شده را برای سرور میفرستند، سرور هم به کمک توابع موجود پیام را رمزگشایی و برای کاربران ارسال میکند. برنامه شامل بخش های زیر است:

کلاس Server:

در این فایل باید تابع start کلاس server را به گونه ای تکمیل کنید که به ازای هر درخواست تردی از کلاس ClientHandler ساخته شود و تابع start آن ترد فراخوانی شود. همچنین تابع run کلاس ClientHandler را به گونه ای باید بازنویسی کنید که هر بار پیش از آنکه client درخواست خود را وارد کند عبارت

```
Enter the encrypted sentence:
Type '**Exit**' to terminate connection.
```

چاپ شود، سپس جواب client را بررسی کند و اگر از فرمت `Strategy name*key*encrypted message` یا `**Exit**` پیروی نمیکرد، پیام `Invalid Input` برای کاربر ارسال شود و استثنایی از نوع `InvalidInputException` با پیام `Oops` پرتاب کند. (پیشنهاد میشود برای تشخیص درستی فرمت درخواست کاربر از regex استفاده کنید.)

- تضمین میشود در پیام رمزگذاری شده تنها از حروف، عدد، `.` ، `,` ، `&` ، `_` ، `'` و `!` استفاده شده.

اگر درخواست کاربر مطابق فرمت اول بود، باتوجه به استراتژی، بخش encrypted message رمزگشایی شود و برای کاربر ارسال شود. و اگر درخواست کاربر `**Exit**` بود عبارت "Connection closed" چاپ شود و ارتباط قطع شود.

واسط DecryptionStrategy:

این واسط دارای یک متد به نام decrypt است:

```
public String decrypt(String message, int key);
```

که شما باید با توجه به توضیحات آن را به سه شکل در کلاسهای زیر طراحی کنید:

۱. الگوریتم MirrorAndShift : در این الگوریتم باید پیام رمزگذاری شده را ابتدا به تعداد کلید ورودی به جلو شیفت دهید سپس پیام حاصل را آینه ای کنید.

ورودی نمونه

```
MirrorAndShift*13*.navobz z'V !lqbolerir vU
```

خروجی نمونه

```
Hi everybody! I'm mobina.
```

۲. الگوریتم AdvancedShift : این متد را طوری پیاده‌سازی کنید که کاراکترهای ورودی با اندیس زوج را به تعداد کلید ورودی به سمت عقب، و کاراکتر با اندیس فرد را به تعداد کلید ورودی به سمت جلو شیفت دهد (فقط حروف a تا z و A تا Z شیفت داده می‌شوند). برای مثال، اگر B را ۲ بار به سمت جلو شیفت دهیم، نتیجه D خواهد بود؛ یا اگر y را ۲ بار به سمت جلو شیفت دهیم، نتیجه a خواهد بود. در صورتی که کارکتری غیر از موارد گفته‌شده بود (مثلا فاصله) آن را تبدیل نکنید و به همان شکل در نتیجه قرار دهید.

ورودی نمونه

```
AdvancedShift*14*omb amfp mamr, moz yodr pm mgn mamr.
```

خروجی نمونه

```
aan mard amad, aan mard ba asb amad.
```

۳. الگوریتم EvenAndOdd : رمزگذاری این الگوریتم به صورت زیر رخ داده: ابتدا رشته برعکس می‌شود. سپس کاراکترهایی که ایندکس زوج دارند به ترتیب در کنار هم و کاراکتر های با ایندکس فرد در کنار هم قرار می‌گیرند. سپس این دو گروه با استفاده از جدا کننده key&key در کنار هم قرار می‌گیرند (تضمین می‌شود در جمله وارد شده عبارت key&key وجود نداشته باشد همچنین key حتما عددی یک رقمی خواهد بود). مثلا اگر key عدد 5 باشد، جداکننده به صورت 5&5 خواهد بود.

ورودی نمونه

```
EvenAndOdd*8*.gnve trvnshuh uhi do wlbnae tgoty p l do M8&8o eaho ee tgottotwsrW.oe ir
```

خروجی نمونه

```
My words fly up, my thoughts remain below. Words without thoughts never to heaven go.
```

پس از تکمیل، فایل های Server, MirrorAndShift, AdvancedShift, EvenAndOdd آنها را به صورت zip آپلود کنید.

چندریختی؟

- نام طراح: سپهر ابراهیمی
- سطح سوال: متوسط

سپهر که مبحث چندریختی یا polymorphism را در جاوا به خوبی درک نکرده بود، بعد از مدتی جستجو به دوستش نیما مراجعه می‌کند تا او این مشکل را برایش حل کند. نیما به جای توضیح مستقیم، برنامه‌ی ناقصی به سپهر می‌دهد تا با کامل کردنش، از نکات چندریختی آگاه شود و استفاده‌ی آن‌ها را ببیند.

مسئله اینجاست که سپهر هنوز نتوانسته این برنامه را طبق خواسته‌ی نیما کامل کند، در نتیجه او از شما می‌خواهد که در این کار به او کمک کنید.

این، برنامه‌ی ناقصی است که نیما به سپهر داده:

```
abstract class AbstractClass{

    final void func(){

        if(/* TODO */){

            System.out.print("not ");

        }

        System.out.println("func here");

    }

    abstract void func1(int a);

    abstract void func2();

    abstract AbstractClass func3(int a);

```

```
}

class Parent extends AbstractClass{

    int a=3;

    // TODO

}

final class Child extends Parent{

    // TODO

}

public class Main {

    public static void main(String[] args) {

        int a = 10;

        // TODO

        reference.func();

        reference.func1(a);

        reference.func2();

        // TODO

        reference.func();

        reference.func1(a);

        reference.func2();

        // TODO

```

```
61          reference.func4();
62
63          reference.func3(10).func4();
64
65      }
66
67  }
```

طبق گفته‌های نیما، تابع `func1` در کلاس `Parent` نصف عدد ورودیش را چاپ می‌کند، اما در کلاس `Child` چهار برابر آن را.

همچنین تابع `func2` در کلاس `Parent` باید عبارت `did nothing` را چاپ کند، اما در کلاس `Child` باید تابع `func` را صدا بزند. نکته‌ی مهم این است که در این کلاس، نهایتا باید عبارت `func here` چاپ شود.

تابع `func3` نیز در کلاس `Parent` یک نمونه (instance) از کلاس `Child` با مقدار `a` برابر با ورودی خودش بسازد و آن را برگرداند. این تابع در کلاس `Child` دست نخورده خواهد ماند.

در نهایت تابع `func4` عبارت `doing the thing` را به تعداد `a` بار تکرار خواهد کرد. هر عبارت با عبارت قبلی یک فاصله دارد و همچنین در انتها نیز باید به خط بعد بروید. توجه کنید که آخرین عبارت بعد از خود فاصله‌ای ندارد و صرفا به خط بعد خواهد رفت. نیما روی این جزئیات حساس است و اگر این مورد رعایت نشود، از سپهر ایراد خواهد گرفت. به خروجی مد نظر نیما توجه کنید. این تابع در کلاس `Child` مقدار `a` را به صورت `this is a: 3` (با فرض برابر بودن a با ۳) چاپ میکند.

نیما همچنین گفته که بعضی نکات را عمداً بیان نکرده تا سپهر خودش به آن‌ها پی ببرد. با کامل کردن تابع main، خروجی مورد نظرش اینچنین خواهد بود:

```
func here
5
did nothing
not func here
40
func here
```

```
doing the thing doing the thing doing the thing
the a is: 10
```

چیزی که باید به سپهر تحویل دهید

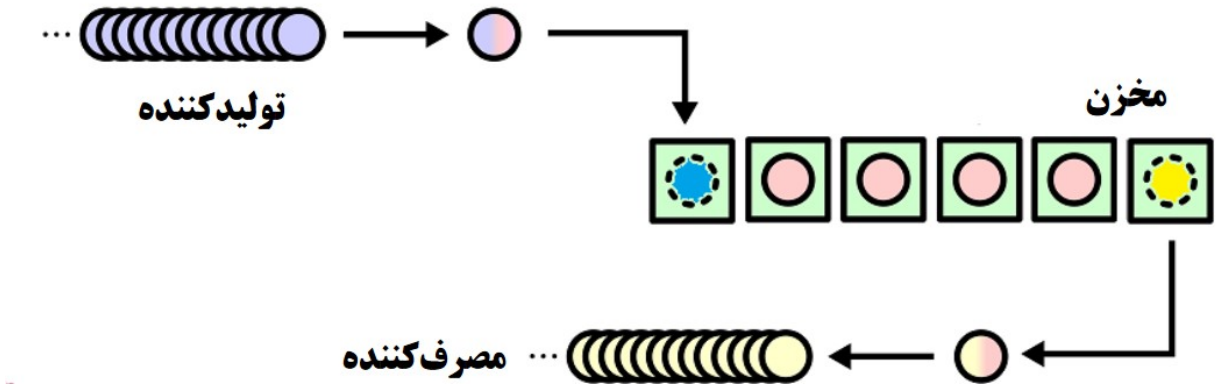
کد تکمیل‌شده‌ایست که نیما به او داده بود. توجه کنید که فقط باید در قسمت‌هایی که با کامنت `TODO` مشخص شده تغییر ایجاد بکنید، وگرنه نیما کد را از سپهر قبول نخواهد کرد. ممکن است برای بعضی از این کامنت‌ها، نیاز به نوشتن بیشتر از یک خط کد شود.

از تولید به مصرف

- مباحث مرتبط: Concurrency, Multi-Threading

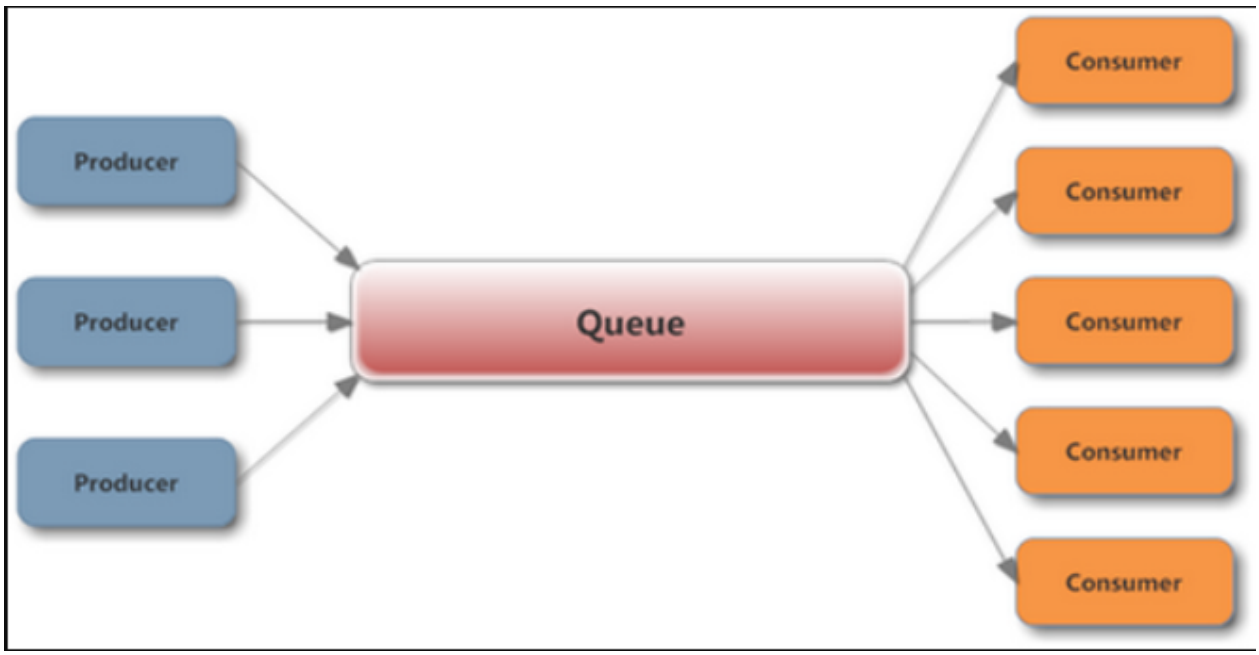
مروری بر مسئله‌ی Producer-Consumer

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 ConcurrentLinkedDeq

    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

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.