صف

در این سوال می‌خواهیم یک صف (Queue) را به صورت جنریک پیاده‌سازی کنیم. برای آشنایی با ساختمان داده‌ی صف، می‌توانید از این لینک استفاده کنید.

جزئیات پروژه

پروژه‌ی اولیه را از این لینک دانلود کنید.

کلاس QueueImpl

این کلاس بیانگر یک پیاده‌سازی از صف بوده و شامل ویژگی‌های زیر است:

- `int size` : این فیلد تعداد المنت‌هایی که در حال حاضر در لیست هستند را نگه‌داری می‌کند.
- `T[] queue` :این فیلد همان صف ما است که به صورت آرایه نگه‌داری می‌شود. سایز آن همواره به اندازه مقدار `capacity` است.
- `int capacity` : این مقدار همواره اندازه آرایه `queue` را نگه‌داری می‌کند. مقدار اولیه‌ی آن ۴ است. در صورتی که آرایه پر شد، باید اندازه `capacity` دو برابر بشود.

متدهای این کلاس را مطابق توضیحات زیر پیاده‌سازی کنید:

```
1 | public boolean add(T element) throws NullElement
```

این متد، `element` را به انتهای آرایه‌ی `queue` اضافه می‌کند. در صورتی که `element` نال باشد، اکسپشن `NullElement` پرتاب می‌شود. اگر در زمان اضافه کردن المنت جدید متوجه شدید که آرایه پر شده، ظرفیت آن را دو برابر کنید. در نهایت مقدار `true` را برگردانید.

```
1 | public boolean offer(T element)
```

این متد دقیقاً مانند متد `add` عمل می‌کند. منتها در صورتی که `element` نال بود، مقدار `false` و در صورتی که عملیات به اتمام رسیده بود مقدار `true` برگردانده می‌شود.

```
public T element() throws EmptyQueue
```

این متد، اولین مقدار صف را بر می‌گرداند. توجه کنید که صرفا آن را بر می‌گرداند و از صف خارج نمی‌کند. در صورتی که هیچ المنتی در صف وجود نداشت، اکسپشن `EmptyQueue` پرتاب می‌شود.

```
public T peek()
```

این متد دقیقاً مانند متد `element` عمل می‌کند. فقط در صورتی که هیچ مقداری در صف وجود نداشت، `null` برمی‌گرداند.

```
public T remove() throws EmptyQueue
```

این متد اولین مقدار آرایه `queue` را پاک کرده و بر می‌گرداند. در صورتی که صف خالی بود، اکسپشن `EmptyQueue` پرتاب می‌شود. نیازی به عوض کردن `capacity` بعد از پاک کردن نیست.

```
public T poll()
```

این متد دقیقا مشابه `remove` عمل می‌کند. با این تفاوت که اگر صف خالی بود، مقدار `null` را بر می‌گرداند.

```
public boolean isEmpty()
```

این متد اگر صف خالی باشد، مقدار `true` و در غیر این‌صورت، مقدار `false` را برمی‌گرداند.

مثال

در صورتی که در متد `main` کد زیر را وارد کنیم:

```
QueueImpl<Integer> q = new QueueImpl<Integer>();
System.out.println(q.peek());
q.add(1);
```

```
4  q.add(2);
5  q.add(3);
6  q.add(4);
7  q.add(5);
8  q.add(6);
9  q.remove();
10 q.poll();
11 System.out.println(q.peek());
12 System.out.println(q.capacity + ", " + q.size);
13 q.add(7);
14 q.add(8);
15 System.out.println(q.size +" "+ q.capacity);
16 System.out.println(q.element());
17 q.remove();
18 q.remove();
19 q.offer(19);
20 System.out.println(q.toString());
```

باید خروجی زیر را ببینیم:

```
null
3
8, 4
6 8
3
[5, 6, 7, 8, 19]
```

آن‌چه باید آپلود کنید

فایل `QueueImpl.java` را آپلود کنید.

چیراکسی؟

در این سوال برای ساده‌تر کردن مدیریت خطاها و مرتب کردن استثناها می‌خواهیم تمام خطاهایی که رخ می‌دهد را با یک قرارداد واحد در کلاسی به نام `ExceptionProxy` ذخیره کنیم. برای این کار به ازای هر خطایی که رخ می‌دهد یک شی از کلاس `ExceptionProxy` می‌سازیم که دارای دو خصوصیت است. این خصوصیات عبارتند از (`e` یک شی از کلاس `ExceptionProxy` است):

- متن استثنای رخ داده. (تبدیل‌شده استثنا به رشته): `e.msg`
- تابعی که باعث ایجاد استثنا شده: `e.function`

فایل Solution.java را دانلود کرده و محتوای آن را ببینید. از شما می‌خواهیم متد `transformException()` را پیاده‌سازی کنید. این متد یک لیست از توابع ورودی می‌گیرد. سپس هر کدام از توابع را صدا می‌کند (توابع بدون آرگومان هستند) و استثناهایی که رخ می‌دهد را با قرارداد بالا به شی‌ای از `ExceptionProxy` تبدیل کرده و در نهایت لیست خطاهای تبدیل‌شده را به همان ترتیب توابع بر می‌گرداند. دقت کنید که اگر تابعی بدون خطا اجرا شد باید یک شی `ExceptionProxy` ساخته و مقدار `msg` آن را با `"OK!"` مقداردهی کنید.

مثال

با اجرای متد `main` در کلاس `Solution` خروجی زیر باید در کنسول چاپ شود:

```
msg: / by zero
function: Devide[1/0]
msg: OK!
function: Devide[1/1]
```

آن‌چه باید آپلود کنید

فایل `Solution.java` را آپلود کنید.

رمزنگار

رمزنگاری نقشی حیاتی در انتقال اطلاعات به‌صورت امن دارد. مصطفی به تازگی به این حوزه علاقه‌مند شده و شروع به مطالعه و تمرین درباره‌ی رمزنگاری کرده است. او می‌خواهد برنامه‌ای شامل الگوریتم‌های مختلف رمزنگاری پیام پیاده‌سازی کند و آن را در *GitHub* منتشر کند. مصطفی اهل نوشتن کد تمیز نیست، اما می‌خواهد از الگوی طراحی *strategy* استفاده کند تا برنامه‌اش ساختار منظمی داشته باشد. بنابراین، از شما خواسته تا برنامه‌ی او را پیاده‌سازی کنید.

جزئیات برنامه

فایل‌های اولیه‌ی برنامه را از این لینک دانلود کنید.

بسته‌ی `exceptions`

این بسته شامل سه استثنا با نام‌های `WrongKeyException` ، `MessageAlreadyEncryptedException` و `MessageAlreadyDecryptedException` است.

دقت کنید که فیلد `serialVersionUID` در استثناها مربوط به serialization است که در ادامه‌ی این ترم با آن آشنا خواهید شد و هیچ تاثیری در حل این سوال ندارد. (مطالعه بیشتر +)

بسته‌ی `strategies`

این بسته شامل الگوریتم‌های رمزنگاری است که باید پیاده‌سازی شوند. اینترفیسی با نام `EncryptionDecryptionStrategy` در این بسته وجود دارد که همه‌ی الگوریتم‌ها باید در کلاس‌هایی پیاده‌سازی شوند که این اینترفیس را `implement` می‌کنند. این اینترفیس به‌صورت زیر است:

```java
package strategies;

public interface EncryptionDecryptionStrategy {
    public String encrypt(String message, int key);

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

همه‌ی الگوریتم‌های رمزنگاری در این برنامه، یک رشته‌ی ورودی و یک کلید محرمانه به‌عنوان ورودی دریافت می‌کنند نتیجه را به‌صورت یک رشته برمی‌گردانند. الگوریتم‌ها را بر اساس توضیحات زیر پیاده‌سازی کنید:

- **الگوریتم `ReverseAndConcatter`**
  - متد `encrypt` این متد را طوری پیاده‌سازی کنید که رشته‌ی ورودی را برعکس کرده، کلید محرمانه را به انتهای آن بچسباند و رشته‌ی رمزنگاری‌شده را برگرداند.
  - متد `decrypt` این متد را طوری پیاده‌سازی کنید که کلید محرمانه را از انتهای رشته حذف کرده، رشته را برعکس کند و رشته‌ی رمزگشایی‌شده را برگرداند. فرض کنید تعداد ارقام کلید هیچ‌گاه بزرگ‌تر از طول رشته‌ی ورودی نیست.
- **الگوریتم `CaesarCipher`**
  - متد `encrypt` این متد را طوری پیاده‌سازی کنید که کاراکترهای ورودی را به تعداد کلید ورودی به سمت جلو شیفت دهد (فرض کنید فقط از حروف `a` تا `z` و `A` تا `Z` استفاده می‌شود). برای مثال، اگر `A` را ۲ بار به سمت جلو شیفت دهیم، نتیجه `C` خواهد بود؛ یا اگر `z` را ۱ 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.
  - متد `decrypt` : این متد را طوری پیاده‌سازی کنید که مشابه متد `encrypt` عمل کند، با این تفاوت که کاراکترها را به سمت عقب شیفت دهد.
- الگوریتم `AdvancedCaesarCipher`
  - متد `encrypt` این متد را مشابه متد `encrypt` الگوریتم `CaesarCipher` پیاده‌سازی کنید، با این تفاوت که کاراکترهای با اندیس زوج به سمت جلو و کاراکترهای با اندیس فرد به سمت عقب شیفت داده شوند (اندیس را از صفر شروع کنید).
  - متد `decrypt` : این متد را طوری پیاده‌سازی کنید که مشابه متد `encrypt` عمل کند، با این تفاوت که کاراکترهای با اندیس زوج را به سمت عقب و کاراکترهای با اندیس فرد را به سمت جلو شیفت دهد.

برای پیاده‌سازی این الگوریتم‌ها می‌توانید (و توصیه می‌شود) از کلاس‌های زیر استفاده کنید.

- کلاس mutable کار با رشته به نام `StringBuilder` مثلا متد `reverse`

- متدهای مختلف خود `String` مثلا `substring`
- متدهای استاتیک کلاس `Character`

کلاس `SecretMessage`

از این کلاس برای جلوگیری از انجام چندباره‌ی عملیات‌های رمزنگاری و رمزگشایی روی پیام‌ها و صحت‌سنجی کلیدهای محرمانه استفاده می‌شود.

- کانستراکتور این کلاس را طوری پیاده‌سازی کنید که یک پیام رمزنگاری‌نشده را دریافت کرده و آن را برای عملیات‌های بعدی رمزنگاری و رمزگشایی نگه‌داری کند. می‌توانید فیلدهای جدیدی در کلاس تعریف کنید.
- متد `getMessage` را طوری پیاده‌سازی کنید که متن فعلی پیام را برگرداند؛ یعنی اگر پیام رمزنگاری شده بود، متن رمزنگاری‌شده و در غیر این‌صورت، متن خام را برگرداند.
- متد `isEncrypted` را طوری پیاده‌سازی کنید که مشخص کند پیام در حال حاضر رمزنگاری شده است یا خیر.
- متد `encrypt` را طوری پیاده‌سازی کنید که با دریافت یک الگوریتم از نوع `EncryptionDecryptionStrategy` و یک کلید، پیام را رمزنگاری کند. اگر پیام از قبل رمزنگاری شده بود، یک استثنا از نوع `MessageAlreadyEncryptedException` پرتاب کنید. اگر مقدار کلید بزرگ‌تر از صفر **نبود**، یک استثنا از نوع `WrongKeyException` پرتاب کنید.
- متد `decrypt` را طوری پیاده‌سازی کنید که با دریافت یک الگوریتم از نوع `EncryptionDecryptionStrategy` و یک کلید، پیام را رمزگشایی کند. اگر پیام از قبل رمزگشایی شده بود، یک استثنا از نوع `MessageAlreadyDecryptedException` پرتاب کنید. اگر مقدار کلید بزرگ‌تر از صفر **نبود**، یک استثنا از نوع `WrongKeyException` پرتاب کنید.

کلاس `EncryptorDecryptor`

از این کلاس برای انجام عملیات رمزنگاری و رمزگشایی در تعداد بالا و با **یک الگوریتم یکسان** استفاده می‌شود.

- کانستراکتور این کلاس را طوری پیاده‌سازی کنید که یک الگوریتم رمزنگاری از نوع `EncryptionDecryptionStrategy` دریافت کرده و آن را برای عملیات‌های بعدی رمزنگاری و

رمزگشایی نگه‌داری کند. می‌توانید فیلدهای جدیدی در کلاس تعریف کنید.

- متد `encrypt(SecretMessage message, int key)` را طوری پیاده‌سازی کنید که با دریافت یک پیام از نوع `SecretMessage` و یک کلید محرمانه، پیام را رمزنگاری کند و متن رمزنگاری‌شده‌ی پیام را برگرداند.
- متد `encrypt(String message, int key)` را طوری پیاده‌سازی کنید که با دریافت یک رشته‌ی پیام و یک کلید محرمانه، رشته‌ی رمزنگاری‌شده را برگرداند.
- متد `decrypt(SecretMessage message, int key)` را طوری پیاده‌سازی کنید که با دریافت یک پیام از نوع `SecretMessage` و یک کلید محرمانه، پیام را رمزگشایی کند و متن رمزگشایی‌شده‌ی پیام را برگرداند.
- متد `decrypt(String message, int key)` را طوری پیاده‌سازی کنید که با دریافت یک رشته‌ی پیام و یک کلید محرمانه، رشته‌ی رمزگشایی‌شده را برگرداند.

توجه: ممکن است تصور کنید که متدهای encrypt/decrypt تکراری هستند و نباید از هر کدام دو تا داشته‌باشید. اما به تفاوتشان دقت کنید. متد decrypt/encrypt ی که رشته ورودی می‌گیرد، فقط عملیات رمزنگاری را روی یک رشته به کمک `encryptorDecryptorStrategy` خود انجام می‌دهد. اما متدهایی که `SecretMessage` دریافت می‌کنند، علاوه بر خود عملیات رمزنگاری، بررسی صحت کلید و وضعیت پیام (که از پیش رمزنگاری شده یا نه) را انجام می‌دهد و برای همین ممکن است استثنا پرتاب کند. (یعنی این متد از متدهای رمزنگاری خود `SecretMessage` استفاده می‌کند)

مثال

پس از پیاده‌سازی برنامه، با اجرای متد `main` موجود در کلاس `Main` :

```java
import exceptions.*;
import strategies.*;

public class Main {
    public static void main(String[] args) {
        EncryptorDecryptor ed1 = new EncryptorDecryptor(
                new ReverseAndConcatter());
        SecretMessage sm = new SecretMessage("Hello from anonymous");
        try {
            System.out.println(ed1.encrypt(sm, 1400));
            System.out.println(sm.getMessage());
```

```
            System.out.println(ed1.encrypt(sm, 1400));
        } catch (MessageAlreadyEncryptedException e) {
            System.out.println("Message already encrypted.");
        } catch (WrongKeyException e) {
            System.out.println("Invalid key.");
        }
        try {
            System.out.println(ed1.decrypt(sm, 1400));
            System.out.println(sm.getMessage());
            System.out.println(ed1.decrypt(sm, 1400));
        } catch (MessageAlreadyDecryptedException e) {
            System.out.println("Message already decrypted.");
        } catch (WrongKeyException e) {
            System.out.println("Invalid key.");
        }

        EncryptorDecryptor ed2 = new EncryptorDecryptor(new CaesarCipher());
        System.out.println(ed2.encrypt("abcd", 2));
        System.out.println(ed2.decrypt("cdef", 2));

        EncryptorDecryptor ed3 = new EncryptorDecryptor(
                new AdvancedCaesarCipher());
        System.out.println(ed3.encrypt("Java", 3));
        System.out.println(ed3.decrypt("Mxyx", 3));
    }
}
```

خروجی برنامه به‌صورت زیر خواهد بود:

```
suomynona morf olleH1400
suomynona morf olleH1400
Message already encrypted.
Hello from anonymous
Hello from anonymous
Message already decrypted.
cdef
abcd
Mxyx
Java
```

آن‌چه باید آپلود کنید

پس از پیاده‌سازی برنامه، یک فایل زیپ آپلود کنید که وقتی آن را باز می‌کنیم، پوشه‌ی strategies و فایل‌های SecretMessage.java و EncryptorDecryptor.java را ببینیم. هم‌چنین، در پوشه‌ی strategies فایل‌های ReverseAndConcatter.java ، CaesarCipher.java و AdvancedCaesarCipher.java را ببینیم. اگر فایل‌های دیگری در فایل زیپ موجود باشند، نادیده گرفته خواهند شد.

```
.
├── strategies
│   ├── AdvancedCaesarCipher.java
│   ├── CaesarCipher.java
│   └── ReverseAndConcatter.java
├── EncryptorDecryptor.java
└── SecretMessage.java
```

موتور جست‌وجوی اشعار

نیما در طی چند ماه گذشته، اشعار خود را در فایل‌های متنی ذخیره کرده است. او هم‌اکنون به‌دنبال ابیاتی است که شامل زیررشته‌ی مشخصی هستند. از آن‌جایی که تعداد این فایل‌ها زیاد است و هم‌چنین فایل‌ها در دایرکتوری‌های مختلفی قرار گرفته‌اند، این کار برای نیما بسیار وقت‌گیر است.

برنامه‌ای برای نیما بنویسید که با استفاده از آن به‌راحتی بتواند اشعار موردنظرش را پیدا کند.

پیاده‌سازی

فایل‌های اولیه‌ی برنامه را از این لینک دانلود کنید.

بسته‌ی `util`

این بسته شامل یک کلاس با نام `FileUtils` است. این کلاس شامل دو متد استاتیک است که باید پیاده‌سازی شوند:

- متد `getFileContents` : این متد با دریافت مسیر یک فایل، محتوای آن را به‌صورت یک رشته برمی‌گرداند.
- متد `getDirectoryFilesList` : این متد با دریافت مسیر یک دایرکتوری، لیستی از مسیر مطلق (*absolute path*) همه‌ی فایل‌های موجود در دایرکتوری و زیردایرکتوری‌های آن را برمی‌گرداند.

بسته‌ی `model`

این بسته شامل یک کلاس با نام `Poem` است. هر شی از کلاس `Poem` بیانگر یک شی است. این کلاس شامل دو فیلد زیر است:

- مسیر فایل متنی شعر (`filePath`)
- متن شعر (`text`)

متدهای زیر از کلاس `Poem` را پیاده‌سازی کنید:

- متد `getFilePath` : این متد، مسیر فایل متنی شعر را برمی‌گرداند.
- متد `getText` : این متد، متن شعر را برمی‌گرداند.

- متد contains : این متد با دریافت یک رشته، در صورتی که متن شعر شامل آن رشته باشد، مقدار true و در غیر این‌صورت، مقدار false را برمی‌گرداند.

بسته‌ی repositories

این بسته شامل یک کلاس با نام PoemRepository است که وظیفه‌ی نگه‌داری اشعار را بر عهده دارد. این کلاس شامل یک Map است که کلیدهای آن مسیر فایل اشعار و مقادیر آن، نمونه‌هایی از کلاس Poem هستند.

متدهای زیر از کلاس PoemRepository را پیاده‌سازی کنید:

- متد addPoem : این متد با دریافت یک شعر، آن را به مخزن اضافه می‌کند.
- متد addPoems : این متد با دریافت لیستی از اشعار، آن‌ها را به مخزن اضافه می‌کند.
- متد addFiles : این متد با دریافت لیستی از مسیر فایل اشعار، نمونه‌هایی از کلاس Poem را ساخته و به مخزن اضافه می‌کند.
- متد getPoems : این متد، نگاشتی از مسیر فایل اشعار به نمونه‌هایی از کلاس Poem را برمی‌گرداند. توجه داشته باشید که هر بار باید یک Map جدید ایجاد شود، یعنی محتوای Map موجود در کلاس را تنها بتوان از طریق متدهای کلاس مخزن تغییر داد.
- متد search : این متد با دریافت یک کلیدواژه به‌صورت یک رشته، نگاشتی از مسیر فایل اشعار به نمونه‌هایی از کلاس Poem به‌طوری که محتوای آن اشعار شامل کلیدواژه‌ی ورودی باشند را برمی‌گرداند. توجه داشته باشید که هر بار نباید از نمونه‌های کلاس Poem کپی بسازید.

مثال

فرض کنید فایل‌های زیر را داریم:

/home/nima/sher1.txt :

```
dar zemestan ta tavani garm sho
dars khan o por ze an azarm sho
```

/home/nima/sher_old/sher174.txt :

```
zemestan shavad pas bahar, ey tabib
bahar por ze sarve chaman, ey habib
```

/home/nima/etc/data.txt :

```
this is not a poem.
```

با اجرای متد `main` موجود در کلاس `Main` :

```java
PoemRepository repo = new PoemRepository(
        FileUtils.getDirectoryFilesList("/home/nima"));
for (String filePath : repo.search("zemestan").keySet()) {
    System.out.println(filePath);
}
```

خروجی برنامه به‌صورت زیر خواهد بود:

```
/home/nima/sher1.txt
/home/nima/sher_old/sher174.txt
```

توجه: مسیر فایل‌ها باید مطلق (*absolute*) باشد. هم‌چنین، ترتیب خطوط خروجی مهم نیست.

آن‌چه باید آپلود کنید

یک فایل *Zip* آپلود کنید که محتوای آن به‌صورت زیر باشد:

```
.
├── model
│   └── Poem.java
├── repositories
│   └── PoemRepository.java
└── util
    └── FileUtils.java
```

اگر فایل دیگری در فایل ارسالی موجود باشد، نادیده گرفته خواهد شد.