هش مپ

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: ساده
- طراح: زهرا عزیزی

در این سوال قصد داریم تا ساختار یک HashMap ساده را با کمک لیست پیوندی پیاده سازی کنیم. توجه کنید که استفاده از داده ساختار های آماده جاوا **پذیرفته نیست** و تمام داده ساختارهای مورد نیاز را خودتان باید پیاده سازی کنید. برای اطلاع بیشتر درباره hash، load factor و rehashing به این لینک مراجعه کنید. (هر چند در این سوال کدهای این بخش نیاز به پیاده سازی ندارند!)

صورت اولیه پروژه را از این لینک دانلود کنید.

کلاس MyHashMap

```
private static final int INITIAL_CAPACITY = 16;
```

اندازه اولیه آرایه مورد استفاده در هش مپ را مشخص می کند. در این سوال اندازه اولیه را ۱۶ در نظر گرفته ایم.

```
private static final double LOAD_FACTOR = 0.75;
```

لود فکتور هش مپ است.

در عملیات اضافه کردن مقادیر جدید در صورتی که تقسیم اندازه فعلی هش مپ بر اندازه آرایه باکت ها بیشتر از لود فکتور شود، نیازمند عملیات rehash هستیم.

```
private Node<K, V>[] buckets;
```

یک آرایه از Nodeهاست. مقادیری که به هش مپ اضافه می شوند در اینجا نگه داری می شوند.

```
private int size;
```

اندازه فعلی هش مپ را نشان می دهد.

```
private static class Node<K, V> {
    K key;
    V value;
    Node<K, V> next;
    Node(K key, V value) {
      this.key = key;
    this.value = value;
    }
}
```

یک کلاس داخلی است که با آن Node را پیاده سازی کرده ایم. (نیاز به تغییر ندارد.)

```
@SuppressWarnings("unchecked")
public MyHashMap() {
    // TODO
}
```

سازنده هش مپ است.

در سازنده، آرایه باکت ها با اندازه اولیه مشخص شده تعریف می شود. همچنین اندازه اولیه هش مپ در سازنده صفر قرار داده می شود.

```
private int hash(K key) {
    return Math.abs(key.hashCode()) % buckets.length;
}
```

تابع هش برای این هش مپ است. (نیاز به تغییر ندارد.)

```
public void put(K key, V value) {
        // TODO

        int index = hash(key);
        Node<K, V> newNode = new Node<>(key, value);

        if(buckets[index] == null)
            buckets[index] = newNode;
        else {
            Node<K, V> curr = buckets[index];
            Node<K, V> prev = null;

            while(curr != null) {
                if(curr.key.equals(key)) {
                    curr.value = value;
                    return;
                }
                prev = curr;
                curr = curr.next;
            }

            prev.next = newNode;
        }

        size++;
 }
```

یک Node جدید با کلید و مقدار داده شده به هش مپ اضافه می کند.

در صورتی که کلید ورودی null باشد، یک exception از نوع IllegalArgumentException با پیام key `cannot be null.` پرتاب می کند.

در صورتی که تقسیم اندازه فعلی هش مپ بر اندازه آرایه باکت ها بزرگتر مساوی لود فکتور شود، تابع rehash صدا زده می شود.

کلید و مقدار داده شده به تابع، داخل یک Node جدید ریخته می شوند. مقدار کلید داده شده hash می شود و به عنوان index قرار داده می شود. اگر در باکت های فعلی، `buckets[index]` خالی بود، Node جدید اینجا قرار داده می شود. در غیر این صورت لیست پیوندی که در این خانه قرار دارد را پیمایش می کنیم تا به اولین جای خالی برسیم و Node جدید را آنجا قرار دهیم.

▼ توضیحات بیشتر

برای پیمایش لیست پیوندی دو اشاره گر از جنس Node<K, V> نیاز دارید:

`curr` : مقدار فعلی در لیست پیوندی را نشان می دهد و در ابتدا به `buckets[index]` اشاره دارد.

`prev` : به مقدار قبلی در لیست اشاره دارد و در ابتدا null است.

پیمایش شما تا زمانی که `curr` مقدار null ندارد باید ادامه پیدا کند و مقادیر `curr` و `prev` باید در هر مرحله بروز شوند.

اگر به جایی رسیدید که کلید آن برابر با کلید ورودی تابع بود، مقدار آن Node را با مقدار ورودی تابع بروز رسانی کنید. اگر لیست کامل پیمایش شد و کلید در لیست فعلی موجود نبود، Node بعد از اشاره گر `prev` را Node جدید بگذارید. توجه کنید که با اضافه شدن یک Node جدید، اندازه هش مپ افزایش می یابد. (اندازه آرایه باکت ها تغییر نمی کند.)

```
public V get(K key) {
    // TODO
}
```

مقدار متناظر به کلید داده شده را بر میگرداند.

در صورتی که کلید ورودی null باشد، یک exception از نوع IllegalArgumentException با پیام key `cannot be null.` پرتاب می کند.

به روش مشابه تابع put لیست پیوندی پیمایش می شود (در اینجا فقط اشاره گر curr کافی است) و در صورتی که به Node با کلید یکسان با کلید ورودی رسیدیم، مقدار متناظر با آن کلید برگردانده می شود. در صورتی که هیچ Node با این کلید موجود نبود null برگردانده می شود.

```
public void remove(K key) {
    // TODO
}
```

گره حاوی کلید داده شده به همراه مقدار متناظر آن را از هش مپ حذف می کند.

در صورتی که کلید ورودی null باشد، یک exception از نوع IllegalArgumentException با پیام key `cannot be null.` پرتاب می کند.

به روش مشابه تابع put لیست پیوندی پیمایش می شود (در اینجا هر دو اشاره گر نیاز است) و در صورتی که به Node دارای کلید یکسان با کلید ورودی رسیدیم، اگر اشاره گر `prev` مقدار null داشت `buckets[index]` را برابر با Node بعدی `curr` قرار می دهیم. در غیر اینصورت گره بعدی `prev` را گره بعدی `curr` قرار می دهیم. توجه کنید که با حذف شدن یک Node، اندازه هش مپ کاهش می یابد. (اندازه آرایه باکت ها تغییر نمی کند.)

```
public boolean containsKey(K key) {
    // TODO;
}
```

وجود یک Node با کلید داده شده در هش مپ را بررسی می کند.

در صورتی که کلید ورودی null باشد، یک exception از نوع IllegalArgumentException با پیام key `cannot be null.` پرتاب می کند.

به روش مشابه تابع put لیست پیوندی پیمایش می شود (در اینجا فقط اشاره گر curr کافی است) و در صورتی که به Node دارای کلید یکسان با کلید ورودی رسیدیم، true برگردانده می شود. در صورتی که هیچ Node با این کلید موجود نبود false برگردانده می شود.

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

}
```

```
}
```

اندازه فعلی هش مپ را برمی گرداند.

```
@SuppressWarnings("unchecked")
private void rehash() {
    Node<K, V>[] oldBuckets = buckets;
    buckets = new Node[oldBuckets.length * 2];
    size = 0;

    for(Node<K, V> node : oldBuckets) {
        while(node != null) {
        put(node.key, node.value);
      node = node.next;
     }
    }
}
```

عملیات rehashing را انجام می دهد.

یک آرایه جدید از باکت ها با اندازه دو برابر تعریف شده و Nodeهای قبلی در آن قرار می گیرند. (نیاز به تغییر ندارد.)

توابعی که با `TODO` مشخص شده اند را تکمیل کنید. فایل MyHashMap.java را به صورت زیپ درآورده و بارگذاری کنید.

خانه هوشمند

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: امیر محمد گنجی زاده

شما مسؤل سیستم هوشمند یک خانه هستید. این خانه دو سیستم (کلاس) **SmartLight** و **SmartThermostat** دارد که باید آنها را کامل کنید. در هر بخش نیز باید حواستان به حالات استثنا باشد.

پروژه اولیه رو از اینجا دانلود کنید.

کلاس SmartLight

فیلد ها :

```
private int brightness;
private int usageCount;
private boolean configurationComplete;
```

که به ترتیب، میزان روشنایی، تعداد دفعات استفاده و یک boolean که نشان دهنده امادگی سیستم برای کار کردن است.

متد ها :

```
public int getBrightness()
```

واضح هست.

```
public void configure()
```

با فراخوانی این متد، دستگاه اماده کار میشود.

```
public void setBrightness(int brightness)
```

این متد میزان روشنایی چراغ را تغییر میدهد.

حالات استثنا :

۱ - اگر قبل از راه اندازی سیستم (configure کردن) از آن استفاده شود، خطای **DeviceNotConfiguredException** رخ میدهد.

۲ - حداکثر تعداد دفعات استفاده از سیستم (فراخوانی متد setBrightness) ۱۰۰ بار است، در غیر این صورت خطای **OveruseException** رخ میدهد.

۳ - روشنایی چراغ عددی بین ۰ تا ۱۰۰ است، در غیر این صورت خطای **InvalidCommandException** رخ میدهد.

کلاس SmartThermostat

فیلد ها :

```
private boolean isOn;
private double temperature;
private boolean requiresSafetyCheck;
```

که به ترتیب، وضعیت روشن یا خاموش بودن، دما و یک boolean که نشان دهنده روشن بودن safe mode است.

متد ها :

```
public double getTemperature()
```

بازم واضحه :).

```
public void turnOn()
```

با فراخوانی این متد، دستگاه روشن میشود.

```
public void turnOff()
```

با فراخوانی این متد، دستگاه خاموش میشود.

```
public void enableSafetyMode()
```

با فراخوانی این متد، safe mode فعال میشود.

```
public void setTemperature(double temperature)
```

این متد دمای محیط را تغییر میدهد.

حالات استثنا :

۱ - اگر هنگام خاموشی سیستم از آن استفاده شود، خطای **DeviceOffException** رخ میدهد.

۲ - اگر safe mode فعال باشد و درخواستی برای تغییر دمای دستگاه، به بالای ۳۰ درجه ایجاد شود، خطای **SafetyException** رخ میدهد.

شما باید برای هر اکسپشن یک کلاس جدا بسازید، به همراه دو کلاس ذکر شده. موفق باشید.

پشت پرده

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: امیر محمد گنجی زاده

شما یک جعبه جادویی پیدا کردید و خیلی مشتاقید که ببینید توش چیه!

ولی باز کردن اون جعبه کار راحتی نیست و شما باید یکسری اطلاعاتی که توی جعبه هست رو قبل از باز کردنش پیدا کنید(یا حتی عوض کنید).

وظیفه شما ساخت کلاس Finder است.

کلاس Finder

متد ها :

```
public String fieldFinder(Object o)
```

در این متد به شما یک شئ داده میشود و شما باید یک رشته شامل اطلاعات فیلد های شئ را برگردانید. هر خط از رشته خروجی شامل اطلاعات یک فیلد است به صورت زیر:

```
name, type, (public / private)
```

که به ترتیب اسم فیلد، تایپ **دقیق** فیلد و نوع دسترسی فیلد است. برای مثال :

```
number, int, public
```

دقت کنید که فیلد ها باید به صورت مرتب شده باشند(فرض کنید هر سطر که اطلاعات یک فلید را نگه میدارد، یک عضو از یه لیست هست، پس ما یه لیست داریم که هر عضوش اطلاعات یه فیلد، از جنس رشته هست، حالا ما لیست را سورت می‌کنیم و بعد خروجی میدهیم).

برای مثال :

```
public Integer b
private double a
```

خروجی شما :

```
a, double, private
b, class java.lang.Integer, public
```

نکته اخر، اینکه انتهای رشته خروجی‌شما باید **'\n'** قرار بدهید.

```
public void match(Object o, int target)
```

این متد به شما یک شئ و یک تارگت ورودی میدهد. یک فیلد به نام number از جنس int درون این شئ قرار گرفته، که شما باید با انجام یک سری عملیات ها، مقدار این فیلد رو برابر با تارگت کنید.

البته شما به هیچ عنوان حق ندارید خودتون مقدار number رو تغییر بدید و باید با استفاده از متد هایی که در همین شي قرار دارد، مقدارش را تغییر بدهید.

دو متد برای راحتی کار شما درون شي وجود دارد :

متدی به نام multiply که مقدار number رو دو برابر میکند.

متدی به نام minus که مقدار number رو یک واحد کاهش میدهد.

دقت کنید که ترتیب استفاده از این دو متد مهم هست و شما اولین بار که از متد دوم استفاده کردید، دیگر نمیتوانید از متد اول استفاده کنید.

و توجه کنید، که شما باید با **کمترین** فراخوانی متد، به خواسه سوال برسید.

مثلا برای رسیدن از ۳ به ۱۰، کمترین عملیاتی که باید انجام شود ۴ تاست (۳ -> ۶ -> ۱۲ -> ۱۱ -> ۱۰).

لازم به ذکره که حق استفاده از کالکشن ها رو ندارید.

مثال :

```java
public class Finder {
    public static String fieldFinder(Object o) {
        // TODO
    }

    public static void match(Object o, int target) throws Except
        // TODO
    }

    static class Test {
        public Integer b;
        private double a;
        public int number;
        public int numberOfOperations;
        public Test(int number) {
            numberOfOperations = 0;
            this.number = number;
        }
        public void multiply() {
            number *= 2;
            numberOfOperations++;
        }
        public void minus() {
            number--;
            numberOfOperations++;
        }
    }

    public static void main(String[] args) throws Exception {
        Test test = new Test(3);
        System.out.println(fieldFinder(test));
        match(test, 10);
        System.out.println(test.numberOfOperations);
    }
}
```

خروجی مورد انتظار :

```
a, double, private
b, class java.lang.Integer, public
```

```
number, int, public
numberOfOperations, int, public

4
```

در نهایت کلاس Finder که حاوی دو متد fieldFinder و match هست رو آپلود کنید.

سیستم مدیریت ICAI

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

همانطور که اطلاع دارید، اولین کنفرانس هوش‌مصنوعی در ایران در دانشگاه شهیدبهشتی تحت عنوان ICAI برگزار شد. یک سیستم مخفی برای مدیریت این رویداد پیاده‌سازی شده‌بود که آراس متوجه می‌شود بخشی از کدهای این سیستم پاک شده‌اند و متاسفانه برای رویدادهای بعدی نمیتوان از این سیستم استفاده کرد. از آنجایی که کدهای داخل این سیستم از داده‌عام (Generics) استفاده کرده‌اند، این سیستم از اهمیت زیادی برخوردار است زیرا می‌توان برای هر رویداد دیگری نیز استفاده کرد. حالا آراس از شما خواسته‌است تا کدهای ناقص این سیستم را در اسرع وقت کامل کنید تا برای رویدادهای بعدی با مشکل مواجه نشیم!!

ابتدا کدهای ناقص را از این لینک دانلود کنید.

توجه: برای فهم بهتر سوال توضیح کلاس‌هایی که نیاز به تغییر ندارند را هم بخوانید تا ابهامات خود را به حداقل رسانده باشید.

کلاس BaseEvent:

این کلاس یک نوع `abstract class` است که کد آن نیاز به تغییر ندارد.

توجه: این کلاس پایه تمام رویدادها است و هر رویداد خاصی از ویژگی‌های این کلاس برای سیستم خود استفاده می‌کند.

فیلدها:

این کلاس صرفا یک فیلد دارد:

- `timestamp` : این فیلد صرفا زمان شروع رویداد را ذخیره می‌کند که در سازنده این کلاس مقدار دهی شده‌است.

سازنده:

```
public BaseEvent() {
    this.timestamp = System.currentTimeMillis();
}
```

همانطور که مشاهده می‌شود داخل این سازنده از متد currentTimeMillis استفاده شده‌است که زمان شروع رویداد را به میلی‌ثانیه برمیگرداند.

متدها:

```
public long getTimestamp() {
    return timestamp;
}
```

این متد یک `getter` است که مقدار زمان شروع رویداد را برمیگرداند.

کلاس Event:

این کلاس برای هر رویداد از نوع `T` ساخته شده‌است.

این کلاس داده‌های هر رویداد را نگه می‌دارد. توجه داشته باشید که هر رویدادی از نوع `T` باید به زمان شروع خود دسترسی داشته باشد و آنرا توسط متد `getter` مربوطه فرخوانی کند(باید `T` نوع درستی از کلاسی که داخل این برنامه وجود دارد را `extends` کند).

```
class Event<T //TODO> {
    private final T data;

    public Event(T data) {
        this.data = data;
    }

    public T getData() {
        return data;


```

```
        }
    }
```

فیلدها:

- این کلاس صرفا یک فیلد `data` دارد که داده آن رویداد را نگه می‌دارد و می‌تواند از نوع دلخواه باشد.

اینترفیس EventListener:

این اینترفیس دارای بدنه کلی زیر است:

```
interface EventListener<T /*TODO*/> {
    void onEvent(Event<T> event);
}
```

که این اینترفیس صرفا یک متد `onEvent` دارد که و اجازه می‌دهد هر کلاسی که نوعی `listener` است آنرا `implement` کند. متد `onEvent` بسته به کلاسی که آنرا استفاده می‌کند می‌تواند کاربردهای متنوعی داشته باشد و اگر جایی از آن استفاده شود کارکرد آن به طور کامل توضیح داده می‌شود(`T` باید یکی از کلاس‌های معرفی شده را `extends` کند. برای اینکار توجه داشته باشید `T` از نوع رویدادهای برگزار شده می‌باشد).

کلاس EventDispatcher:

این کلاس مسئول ثبت افراد ثبت‌نام کننده و انواع رویدادها می‌باشد و از آرایه برای ذخیره‌سازی افراد شرکت‌کننده و رویدادها استفاده می‌کند(استفاده از Collection غیر مجاز است).

فیلدها:

+ `INITIAL_CAPACITY` : ظرفیت تعداد افراد شرکت‌کننده و تعداد انواع رویدادها را نشان می‌دهد.

- `eventTypes` : یک آرایه است که انواع رویدادهایی که شرکت‌کننده‌ها در آن شرکت کرده‌اند را نگه می‌دارد.
- `listeners` : یک آرایه است که افراد شرکت‌کننده را نگهداری می‌کند.

- `size` : تعداد رویدادهای درحال برگزاری را نشان می‌دهد(لزوما با `INITIAL_CAPACITY` برابر نیست).

سازنده:

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

داخل سازنده این کلاس می‌بایست ظرفیت اولیه آرایه‌های `eventTypes` و `listener` مشخص شود. به علاوه می‌بایست اندازه اولیه رویدادهای در حال برگزاری را نشان دهد (در ابتدا هیچ رویدادی در حال برگزاری نیست).

متدها:

متدهای این کلاس به صورت زیر است:

```
public <T /TODO*/> void registerListener(Class<T> eventType, Eve
        //TODO
}
```

این متد یک شرکت‌کننده را ثبت‌نام می‌کند به این شکل که اگر رویدادهای درحال برگزاری برابر با بیشترین تعداد رویدادهای ممکن باشد، درابتدا گنجایش رویدادهای ممکن را بیشتر کرده و سپس به کار خود ادامه می‌دهد.

توجه: برای افزایش ظرفیت تعداد رویدادها، بهتر است بعد از پر شدن گنجایش آرایه‌تان، ظرفیت آنرا دوبرابر کنید. دلیل آنرا می‌توانید از اینجا مطالعه کنید.

پس از اقدام مرحله قبل، این سیستم شرکت‌کننده را به این شکل ثبت‌نام می‌کند که اطلاعات `eventType` و خود `listener` را ذخیره می‌کند.

```
public <T /*TODO*/> void dispatchEvent(Event<T> event) {
    //The body has already completed in file
}
```

این متد با دریافت یک نوع رویداد تمام رویدادهای ذخیره شده تا زمانی که این متد فراخوانی می‌شود را بررسی می‌کند. اگر هر رویداد ذخیره شده‌ای در آرایه مربوطه دقیقا برابر با رویداد مورد نظر باشد، کاربر متناظر با آن رویداد به عنوان عضوی از بخش تدارکات پذیرفته می‌شود (در کلاس‌های پیش‌رو کلاسی معرفی می‌شود که برای پذیرفتن شرکت‌کنندگان به عنوان عضوی از بخش تدارکات ایجاد شده‌است).

توجه: در زمان استفاده از متد `registerListener` به نحوه ثبت‌نام کردن یک کاربر دقت کنید. در این نحوه ثبت‌نام هر عضو از آرایه `listeners` با عضو متناظر خود در آرایه `eventTypes` ارتباط دارد به این شکل که به عنوان مثال `eventTypes[0]` یعنی رویدادی که کاربر `listeners[0]` در آن ثبت‌نام کرده‌است. در واقع این کلاس یک `listener` را صدا می‌زند و در ادامه متد `onEvent` را (که در ادامه توضیح خواهیم داد) برای این `listener` استفاده خواهد کرد و او را در ادامه به عنوان یک عضو از تیم تدارکات میشناسد.

کلاس UserCreatedEvent:

از آنجایی که دانشگاه شهید بهشتی از رویدادهای خلاقانه از طرف کاربران استقبال می‌کند، کاربران میتوانند بعد از ثبت‌نام خود در یک رویداد کلی (که شامل چندین رویداد است) رویداد دلخواه خود را ایجاد کنند و آنرا مدیریت کنند. در این نوع رویدادها که افراد داوطلب مدیر آن هستند، اسم رویداد همان اسم شخص خواهد بود. بنابراین این کلاس صرفا یک فیلد دارد که جلوتر توضیح داده خواهد شد.

فیلدها:

- `username` : این فیلد نام کاربر را نگه می‌دارد که طبق توضیحات بالا اسم رویداد هم خواهد بود.

بدنه این کلاس به شکل زیر است :

```
class UserCreatedEvent //TODO {
    private final String username;

    public UserCreatedEvent(String username) {
        this.username = username;
    }

    public String getUsername() {
        return username;

```

```
    }
}
```

که همانطور که مشاهده می‌کنید در سازنده این کلاس صرفا فیلد نام‌کاربری مقداردهی داده شده‌است و این کلاس صرفا یک `getter` به عنوان متد خود خواهد داشت.

کلاس UserCreatedListener:

بدنه کلی این کلاس به شکل زیر است:

```
class UserCreatedListener //TODO {
    //TODO
}
```

و صرفا پس از استفاده از **متد مربوطه** و override کردن آن، پیام زیر را چاپ کند :

User created: + event.getData().getUsername()

▼ راهنمایی

توجه کنید این کلاس می‌بایست از یک کلاس از نوع listener ارث‌بری کند و متد آنرا override کند.

کلاس SystemEvent:

این کلاس مسئول سیستم موجود در هر رویداد است(توجه کنید این کلاس باید از کلاس مناسب ارث‌بری کند تا به زمان رویداد هم دسترسی داشته باشد).

فیلدها:

- `systemMassage` : صرفا پیام سیستم را نگه می‌دارد. این پیام می‌تواند پیام بعد از خاموش یا روشن کردن آن سیستم باشد.

بدنه کلی متد زیر به صورت زیر است:

```
class SystemEvent /*TODO*/ {
    private final String systemMessage;
```

```
    public SystemEvent(String message) {
        this.systemMessage = message;
    }

    public String getSystemMessage() {
        return systemMessage;
    }
}
```

کلاس SystemEventListener:

هر کاربری که به عنوان مدیر یک رویداد شناخته می‌شود باید برای رویداد خود سیستمی داشته باشد (لپتاپ) و داخل آن امکاناتی باشد تا با آن بتواند رویداد خود را مدیریت کند. این کلاس یک سیستم مجازی برای آن کاربر ایجاد می‌کند و چاپ کردن پیام موردنظر نشان می‌دهد که سیستم مجازی با موفقیت برای کاربر ایجاد شده‌است.

بدنه کلی این کلاس به صورت زیر است:

```
class SystemEventListener //TODO {
    //TODO
}
```

این کلاس، زیر کلاس یک کلاس دیگر است (از آن ارث‌بری می‌کند) و متد آنرا `override` می‌کند.

نکته: در متد `override` شده صرفا پیام زیر چاپ می‌شود:

System Event Occured: + event.getData().getSystemMessage()

`"()System Event Occurred:  + event.getData().getSystemMessage"` که `event` ورودی آن متد `override` شده‌است.

کلاس EventHistoryLogger:

این کلاس برای مدیریت سابقه رویدادها ایجاد شده است و هر رویدادی بعد از برگزاری توسط آرایه‌های این کلاس، نگهداری میشوند تا در صورت دلخواه به اطلاعات آنها دسترسی داشته باشیم.

فیلدها:

- `INITIAL_CAPACITY` : مقدار اولیه در نظر گرفته شده برای نگهداری سابقه رویدادها را نشان می‌دهد.(این فیلد یک تعداد اولیه برای سابقه رویدادها در نظر می‌گیرد و لزوما برابر با تعداد سابقه‌های رویداد موجود در آن لحظه نخواهد بود).
- `eventLog` : یک آرایه برای نگهداری اطلاعات سابقه هر رویداد است که در سازنده این کلاس مقداردهی می‌شود.
- `size` : این فیلد تعداد سوابق رویدادها را تا آن لحظه برمی‌گرداند.

سازنده:

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

توجه کنید در ابتدای فراخوانی این کلاس هیچ سابقه‌ای موجود نیست و باید فیلد مرتبطه در اینجا مقداردهی شود.

به علاوه آرایه تعریف‌شده که فیلد این کلاس محسوب می‌شود، هم باید در این سازنده مقداردهی شود و ظرفیت آن همان مقدار اولیه درنظر گرفته شده برای نگهداری سوابق خواهد بود.

```
    public <T //TODO> void logEvent(Event<T> event) {
        //TODO
    }
```

این متد یک رویداد را می‌گیرد و آنرا برای تهیه سابقه ذخیره می‌کند.

توجه: اگر تعداد رویدادهای حال‌حاضر برای گرفتن سابقه برابر با مقدار اولیه شد، می‌بایست ظرفیت مورد نظر را برای ذخیره سوابق افزایش دهید(باز هم میتوانید ظرفیت آرایه را دوبرابر کنید).

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

این متد تمام سوابق را چاپ می‌کند. نمونه چاپ شدن سوابق در مثال زیر آورده شده‌است.

کلاس Main:

با اجرای بدنه این کلاس:

```
public class Main {
    public static void main(String[] args) {
        EventDispatcher dispatcher = new EventDispatcher();
        EventHistoryLogger logger = new EventHistoryLogger(); //


        dispatcher.registerListener(UserCreatedEvent.class, new


        dispatcher.registerListener(SystemEvent.class, new Syste


        Event<UserCreatedEvent> userEvent = new Event<>(new User
        Event<SystemEvent> systemEvent = new Event<>(new SystemE


        dispatcher.dispatchEvent(userEvent);
        logger.logEvent(userEvent);

        dispatcher.dispatchEvent(systemEvent);
        logger.logEvent(systemEvent);


        logger.printEventHistory();
    }
}
```

باید خروجی زیر چاپ شود:

```
User created: Alice
System Event Occurred: System Shutdown
Event History:
- UserCreatedEvent at 1742331652190
- SystemEvent at 1742331652190
```

توجه: به دلیل استفاده از متد `currentTimeMillis()` اعداد نشان داده شده در دوخط آخر متفاوت خواهند بود.

آنچه که باید آپلود کنید:

باید یک فایل زیپ از کلاس های کامل شده‌ی زیر آپلود کنید.

```
<zip_file_name.zip>
    ├── BaseEvent.java
    ├── Event.java
    ├── EventDispatcher.java
    ├── EventHistoryLogger.java
    ├── EventListener.java
    ├── SystemEvent.java
    ├── SystemEventListener.java
    ├── UserCreatedEvent.java
    └── UserCreatedListener.java
```

فروشگاه استثنایی

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

پروژه اولیه رو از این لینک دانلود کنید.

شما در حال توسعه یک سیستم سفارش‌دهی آنلاین برای یک فروشگاه بزرگ هستید. در این سیستم، کاربران می‌توانند محصولات مختلفی را خریداری کنند و سیستم باید قادر باشد سفارشات را پردازش کرده، هزینه‌ها و ارسال‌ها را محاسبه و انجام دهد. علاوه بر این، سیستم باید قادر به مدیریت انواع مختلف خطاها باشد و پیام‌های خطا را به‌طور دقیق به کاربر نمایش دهد. این سیستم باید به‌گونه‌ای طراحی شود که از اصول برنامه‌نویسی شی‌گرا استفاده کرده و استثناهای مختلف را به‌طور مؤثر مدیریت کند.

بخش 1: طراحی کلاس‌ها و استثناها

۱. **کلاس Order:**

این کلاس باید اطلاعات مربوط به یک سفارش را ذخیره کند. ویژگی‌های این کلاس باید شامل موارد زیر باشد:

- orderId (نوع: String): شناسه یکتا برای سفارش.
- customerId (نوع: String): شناسه یکتا برای مشتری.
- productId (نوع: String): شناسه یکتا برای محصول.
- quantity (نوع: int): تعداد محصول در سفارش.
- pricePerUnit (نوع: double): قیمت هر واحد محصول.
- shippingAddress (نوع: String): آدرس ارسال محصول.
- status (نوع: String): وضعیت سفارش ("در حال پردازش"، "ارسال شده" و یا "پردازش نشده").

همچنین، متدهای زیر باید برای این کلاس پیاده‌سازی شوند:

- سازنده(Constructor) که تمام ویژگی‌ها را به‌طور کامل مقداردهی کند.

- متد validateOrder() که برای بررسی صحت سفارش و تشخیص خطاهای ممکن استفاده شود. در این متد باید خطاهای زیر بررسی شوند:
- اگر quantity منفی باشد، یک استثنای InvalidQuantityException پرتاب کند.
- اگر pricePerUnit منفی باشد، یک استثنای InvalidPriceException پرتاب کند.
- اگر shippingAddress خالی یا null باشد، یک استثنای InvalidShippingAddressException پرتاب کند.

۲. **کلاس OrderProcessor:**

۳. این کلاس مسئول پردازش و مدیریت سفارشات است. متدهای زیر باید در این کلاس پیاده‌سازی شوند:

+** متد processOrder(Order order): این متد باید سفارش را پردازش کند: **

- قبل از پردازش، باید از متد validateOrder() کلاس Order برای بررسی صحت سفارش استفاده شود.
- اگر سفارش معتبر نباشد، متد processOrder باید استثنای InvalidOrderException را پرتاب کند.
- اگر سفارش معتبر باشد، وضعیت آن باید به "در حال پردازش" تغییر کند.
- **متد calculateTotalAmount(Order order):* این متد باید مجموع هزینه سفارش را محاسبه کند. اگر قیمت واحد منفی باشد، استثنای InvalidPriceException باید پرتاب شود.
- **متد shipOrder(Order order):* این متد باید وضعیت سفارش را به "ارسال شده" تغییر دهد. اگر سفارش قبلاً پردازش نشده باشد یا قبلاً ارسال شده باشد، استثنای OrderNotProcessedException باید پرتاب شود.

استثناهای سفارشی :

- InvalidOrderException: هنگامی که یک سفارش نادرست باشد (مثل قیمت منفی یا آدرس خالی).
- InvalidQuantityException: هنگامی که تعداد محصول منفی باشد.
- InvalidPriceException: هنگامی که قیمت واحد منفی باشد.
- InvalidShippingAddressException: هنگامی که آدرس ارسال نادرست باشد.
- OrderNotProcessedException: هنگامی که تلاش می‌شود یک سفارش که پردازش نشده است را ارسال کرد.

شما باید یک فایل Zip شامل یک فایل به نام `ExceptionalStore.java` را آپلود کنید.