# نقشه سرقت پویا

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح سوال: ساده
- طراح: امیررضا یزدان پناه


پروفسور برای سرقت بعدی خود، بانک ملی اسپانیا را مورد هدف قرار داده است، و با توجه به عظمت این هدف باید مطمئن شود که نقشه او بی نقص است. برای اینکار تعدادی طبحکار کهنه کار را به خدمت گرفته است. اما این افراد به طور شانسی انتخاب نشده اند، بلکه با توجه به مهارت و تخصص هر فرد انتخاب شده، اما فارغ از مهارت هایشان، این افراد باید بتوانند در شرایط اضطراری یک سری عملیات هایی که پروفسور از قبل برایشان توضیح داده را انجام بدهند. حالا پروفسور از شما به عنوان یک برنامه نویس خبره می خواهد تا یک برنامه شبیه سازی سرقت بنویسید تا بتواند عملکرد و پویایی افراد را در شرایط بحرانی بررسی کند.

پروژه اولیه را می توانید از این لینک دانلود کنید.

## جزئیات پروژه

### کلاس HeistMember

**پراپرتی ها:**

- `name` : نام افراد

**متد ها:**

- `constructor` : فیلد `name` را مقدار دهی می کند.
- `executePlan` : یک استریگ به صورت زیر برمی گرداند که کار اصلی که همه توی شرایط خاص باید بتوانند انجام دهند را مشخص می کند:

```
[name] is executing the original plan.
```

- `handleEmergency` : یک استرینگ به صورت زیر برمی گرداند که پروتکل اضطراری پیش فرض را مشخص می کند:

```
[name] follows default emergency protocol.
```

- `useTool` : یک استرینگ به صورت زیر برمی گرداند که مشخص می کند شخص از چه ابزاری استفاده می کند:

```
[name] uses [tool].
```

## کلاس Hacker (از HeistMember ارثبری می کند)

**متد ها:**

- `constructor` : نام شخص را مقدار دهی می کند.
- اورراید `executePlan` : یک استرینگ برمی گرداند که کار هکر را مشخص می کند:

```
[name] is hacking the Banco de España servers.
```

- اورراید `handleEmergency` : یک استرینگ برمی گرداند که پروتکل اضطراری مختص هکر را مشخص می کند:

```
[name] is rerouting security feeds to bypass alarms!
```

## کلاس Fighter (از HeistMember ارثبری می کند)

**متد ها:**

- `constructor` : نام شخص را مقدار دهی می کند.
- اورراید `executePlan` : یک استرینگ برمی گرداند که کار جنگنده را مشخص می کند:

```
[name] is securing the hostages.
```

- اورراید `handleEmergency` : یک استرینگ برمی گرداند که پروتکل اضطراری مختص جنگنده را مشخص می کند:

```
[name] is engaging the police with full force!
```

- اورلود `useTool(tool, duration)` : یک استرینگ برمی گرداند که کاربرد جنگنده از ابزار را مشخص می کند:

```
[name] uses [tool] aggressively for [duration] minutes!
```

## کلاس Distractor (از HeistMember ارثبری می کند)

**متد ها:**

- `constructor` : نام شخص را مقدار دهی می کند.
- اورراید `executePlan` : یک استرینگ برمی گرداند که کار گمراه کننده را مشخص می کند:

```
[name] is creating a diversion in the courtyard.
```

## سنجش درستی

در صورتی که کد زیر اجرا شود خروجی شما باید با آن یکسان باشد:

```
public class HeistSimulation {
```

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

        HeistMember[] team = {

                new Hacker("Lisbon"),
                new Fighter("Tokyo"),
                new Distractor("Denver")
          };

          for (HeistMember member : team) {

              System.out.println(member.executePlan());
              System.out.println(member.handleEmergency());

              if (member instanceof Fighter) {

                  System.out.println(member.useTool("gun"));
                  Fighter fighter = (Fighter) member;
                  System.out.println(fighter.useTool("smoke grenad

              } else {

                  System.out.println(member.useTool("walkie-talkie
              }

              System.out.println();
          }
      }
}
```

## خروجی

```
Lisbon is hacking the Banco de España servers.
Lisbon is rerouting security feeds to bypass alarms!
Lisbon uses walkie-talkie.

Tokyo is securing the hostages.
Tokyo is engaging the police with full force!
Tokyo uses gun.
Tokyo uses smoke grenade aggressively for 5 minutes!

Denver is creating a diversion in the courtyard.
```

```
Denver follows default emergency protocol.
Denver uses walkie-talkie.
```

## آنچه باید اپلود کنید:

یک `zip` با ساختار زیر ارسال کنید:

```
<zip_file_name.zip>
    ├── HeistMember.java
    ├── Hacker.java
    ├── Fighter.java
    └── Distractor.java
```

# سیستم پرداخت آنلاین

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

پروژه اولیه رو از این لینک دانلود کنید. در این سوال، شما باید یک سیستم پرداخت آنلاین طراحی کنید که از چندین روش پرداخت پشتیبانی کند (مثل کارت‌های اعتباری، کیف پول‌های دیجیتال و سیستم‌های رمزارز). در این سیستم، تخفیف‌های مختلفی به هر روش پرداخت اعمال خواهد شد.

**وظایف شما:**

۱. **ایجاد اینترفیس‌های مشترک برای روش‌های پرداخت و تخفیف‌ها.**
۲. **پیاده‌سازی روش‌های مختلف پرداخت.**
۳. **استفاده از الگوی Strategy Pattern برای اعمال تخفیف‌های پویا.**

**توضیحات:**

**اینترفیس PaymentMethod:**

- این اینترفیس شامل متدهای pay , refund و getBalance است.

```
interface PaymentMethod {
      void pay(double amount);
      void refund(double amount);
      double getBalance();
      }
```

**اینترفیس DiscountStrategy**

- این اینترفیس باید شامل متد applyDiscount(double amount) باشد که برای هر روش پرداخت، تخفیف مربوطه را اعمال کند.

```
interface DiscountStrategy {
      double applyDiscount(double amount);
```

```
}
```

**راهنمایی:** در پیاده سازی تابع pay باید بررسی شود که مبلغ پرداختی مثبت و کمتر از موجودی حساب باشد.

همچنین تابع refund فقط با ورودی مثبت کار می‌کند.

**پیاده‌سازی کلاس‌های مختلف برای روش‌های پرداخت:**

- برای هر روش پرداخت (مثل کارت اعتباری، کیف پول دیجیتال، و رمزارز) یک کلاس جداگانه بنویسید که از اینترفیس PaymentMethod پیروی کند.
- پرداخت با کارت اعتباری: 5 درصد تخفیف
- پرداخت با کیف پول اینترنتی: 10 درصد تخفیف
- پرداخت با رمزارز: 15 درصد تخفیف
- هر کلاس باید درصد تخفیف متفاوتی را از طریق DiscountStrategy اعمال کند.

۱. **کلاس PaymentSystem برای مدیریت پرداخت‌ها:**

این کلاس از روش‌های مختلف پرداخت و تخفیف‌ها استفاده، و فرایند پرداخت را مدیریت می‌کند.

## مثال

### ورودی نمونه ۱

```
public class PaymentSystem {
    public static void makePayment(PaymentMethod paymentMethod,
        paymentMethod.pay(amount);
    }
    public static void main(String[] args) {
        //different discount strategies
        DiscountStrategy creditCardDiscount = new CreditCardDisc
        DiscountStrategy walletDiscount = new WalletDiscount();
        DiscountStrategy cryptoDiscount = new CryptoDiscount();
        // creating payment methods
        PaymentMethod creditCardPayment = new CreditCardPayment(
        PaymentMethod walletPayment = new WalletPayment(500, wal
        PaymentMethod cryptoPayment = new CryptoPayment(2000, cr
        // payments
        makePayment(creditCardPayment, 100);
```

```
        System.out.println(creditCardPayment.getBalance());
        makePayment(walletPayment, 100);
        System.out.println(walletPayment.getBalance());
        makePayment(cryptoPayment, 100);
        System.out.println(cryptoPayment.getBalance());
    }
```

## خروجی نمونه ۱

```
905.0
410.0
1915.0
```

## ورودی نمونه 2

```
class PaymentSystem {
    public static void makePayment(PaymentMethod paymentMethod,
        paymentMethod.pay(amount);
    }
    public static void main(String[] args) {
        //different discount strategies
        DiscountStrategy creditCardDiscount = new CreditCardDisc
        DiscountStrategy walletDiscount = new WalletDiscount();
        DiscountStrategy cryptoDiscount = new CryptoDiscount();
        // creating payment methods
        PaymentMethod creditCardPayment = new CreditCardPayment(
        PaymentMethod walletPayment = new WalletPayment(500, wal
        PaymentMethod cryptoPayment = new CryptoPayment(2000, cr
        // payments
// پرداخت ۲۰۰ با تخفیف ۵٪
        creditCardPayment.pay(200);
        System.out.println(creditCardPayment.getBalance());

// پرداخت ۱۵۰ با تخفیف ۱۰٪
        walletPayment.pay(150);
        System.out.println(walletPayment.getBalance());

// پرداخت ۳۰۰ با تخفیف ۱۵٪
        cryptoPayment.pay(300);
        System.out.println(cryptoPayment.getBalance());
```

```
26
27          creditCardPayment.pay(2000);
28          System.out.println(creditCardPayment.getBalance());
29
30          creditCardPayment.refund(100);
31          System.out.println(creditCardPayment.getBalance());
32
33          walletPayment.refund(50);
34          System.out.println(walletPayment.getBalance());
35
36          cryptoPayment.refund(200);
37          System.out.println(cryptoPayment.getBalance());
38      }}
```

## خروجی نمونه 2

```
810.0
365.0
1745.0
810.0
910.0
415.0
1945.0
```

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

# داندرمیفلین

- سطح: متوسط
- طراح: زهرا عزیزی

یکی از شرکت های رقیب *Dunder Mifflin* توانسته است به کد برنامه های مدیریتی کارکنان شعبه *Scranton* این شرکت دست یابد. اما آن ها بعد از بررسی کدها متوجه شده اند که بخشی از کدها در فرآیند انتقال از بین رفته اند. این شرکت شما را به عنوان یک برنامه نویس خبره استخدام کرده است تا با توجه به توضیحات هر بخش، کدهای ناقص را تکمیل کنید!


صورت اولیه پروژه را از این لینک دانلود کنید.

## ساختار پروژه:

```
├── AbstractEmployee.java
├── Employee.java
├── Manager.java
├── PaperType.java
├── RoleLevel.java
└── Worker.java
```

# توضیحات فایل ها

## PaperType:

یک enum است که حاوی انواع مختلف کاغذ تولید شده در *Dunder Mifflin* می باشد. (نیاز به تغییر ندارد.)

```
public enum PaperType { RECYCLED, GLOSSY, CARDSTOCK, NEWSPRINT,
```

## RoleLevel

یک enum است که رتبه هر کارمند این شرکت را مشخص می کند. ترتیب رتبه هر کارمند اهمیت دارد. به عنوان مثال INTERN پایین ترین رتبه و DIRECTOR بالاترین رتبه را دارد. (نیاز به تغییر ندارد.)

```
public enum RoleLevel { INTERN, WORKER, SUPERVISOR, MANAGER, DIR
```

## Employee

یک interface است که کلاس AbstractEmployee آن را پیاده سازی می کند. (نیاز به تغییر ندارد.)

```
public interface Employee {
    String work();
    double calculateSalary();
    String promote();
    String demote();
    String changeRole(String newRole);
    String getName();
}
```

## AbstractEmployee

اینترفیس Employee را پیاده سازی می کند.

```java
public abstract class AbstractEmployee implements Employee{
    protected final String name;
    protected RoleLevel level;
```

```java
public AbstractEmployee(String name, RoleLevel level) {
        // TODO
    }
```

هر کارمند در این شرکت نام و رتبه مشخص دارد که در سازنده باید مقدار دهی شوند.

```java
    @Override
    public String getName() {
        // TODO
    }
```

اسم کارمند را بر می گرداند.

```java
    @Override
    public String changeRole(String newRole) {
        // TODO
    }
```

نقش یک کارمند را تغییر می دهد. با توجه به حرف اول نقش جدید (استفاده از a و an در جای مناسب) صرفا پیام تغییر نقش را با فرمت زیر بر می گرداند.

```
{name} is now {a/an} {newRole}.
```

```java
    @Override
    public String promote() {
        // TODO
    }
```

رتبه کارمند را در صورت امکان افزایش می دهد. اگر رتبه فعلی DIRECTOR باشد، امکان ترفیع رتبه وجود نداشته و پیامی با فرمت زیر برگردانده می شود.

```
{name} is already at highest level.
```

در صورت ترفیع رتبه، پیامی با فرمت زیر برگردانده می شود و `level` کارمند تغییر می کند. برای محاسبه رتبه بعدی که کارمند می تواند کسب کند باید از values و ordinal در enum ها استفاده کنید. (به ترتیب رتبه ها در RoleLevel دقت کنید.)

```
{name} has been promoted to {level}!
```

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

رتبه کارمند را در صورت امکان کاهش می دهد. اگر رتبه فعلی INTERN باشد، امکان تنزیل رتبه وجود نداشته و پیامی با فرمت زیر برگردانده می شود.

```
{name} is already at lowest level.
```

در صورت تنزیل رتبه، پیامی با فرمت زیر برگردانده می شود و `level` کارمند تغییر می کند. برای محاسبه رتبه قبلی که کارمند می تواند کسب کند باید از values و ordinal در enum ها استفاده کنید. (به ترتیب رتبه ها در RoleLevel دقت کنید.)

```
{name} has been demoted to {level}.
```

## Manager

نشاندهنده یک مدیر است.

```
public class Manager extends AbstractEmployee {
    private double baseSalary;
    private Worker[] team = new Worker[3];
    private int teamCount = 0;
```

مدیر از کلاس AbstractEmployee ارثبری می کند. هر مدیر یک حقوق پایه و یک تیم حداکثر سه نفره دارد که آن ها را مدیریت می کند. از teamCount برای نگه داشتن تعداد اعضای تیم مدیر استفاده می شود.

```
1     public Manager(String name, double baseSalary) {
2         // TODO
3     }
```

در سازنده باید از سازنده کلاس والد (AbstractEmployee) برای مقدار دهی اسم و رتبه استفاده شود. (رتبه مدیر MANAGER است.) همچنین حقوق پایه باید مقدار دهی شود.

```
1      public void addTeamMember(Worker w) {
2         // TODO
3     }
```

یک کارمند جدید را در صورتی که ظرفیت تیم پر نشده باشد ( سه نفر تکمیل نشده باشند) به تیم اضافه می کند.

```
1     @Override
2     public String work() {
3         // TODO
4     }
```

یک پیام حاوی اسم مدیر و افرادی که در تیم مدیر هستند با فرمت زیر بر می گرداند.

```
{name} manages: {worker 1 name}, {worker 2 name}, {worker 3 name}
```

توجه کنید که اگر تیم سه نفره نباشد، باید صرفا به تعداد اعضای فعلی، اسامی نوشته شوند. بعد از آخرین اسم نباید از کاما استفاده شود.

```
1     @Override
2     public double calculateSalary() {
3         // TODO
4     }
```

حقوق مدیر ۲۰ درصد از حقوق پایه بیشتر است. علاوه بر آن برای هر یک از اعضای تیم مدیر، مدیر ۱۰ درصد بیشتر از حقوق هر عضو را نیز دریافت می کند. به عبارتی حقوق مدیر شامل ۲۰ درصد بیشتر از حقوق پایه و ۱۰ درصد بیشتر از حقوق هر یک از اعضای تیمش است که این مقادیر باهم جمع زده شده و به عنوان حقوق مدیر بازگردانده می شود.

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

در صورت ترفیع گرفتن مدیر، حقوق پایه او ۳۰ درصد افزایش پیدا می کند. در اینجا باید از متود promote کلاس والد هم استفاده شود.

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

در صورت تنزیل مدیر، حقوق پایه او ۱۵ درصد کاهش پیدا می کند. در اینجا باید از متود demote کلاس والد هم استفاده شود.

## Worker

نشانده یک کارمند عادی است.

```
public class Worker extends AbstractEmployee {
    private double hourlyRate;
    private int hoursWorked;
    private final PaperType paperType;
```

کارمند از کلاس AbstractEmployee ارثبری می کند. هر کارمند نرخ حقوق مشخص به ازای ساعت کار، میزان ساعات کار کرده و نوع مشخصی از کاغذ که روی آن کار می کند، دارد.

```
    public Worker(String name, double hourlyRate, int hoursWorke
        // TODO
```

```
3 |      }
```

در سازنده باید از سازنده کلاس والد (AbstractEmployee) برای مقدار دهی اسم و رتبه استفاده شود. (رتبه کارمند WORKER است.) همچنین باید نرخ حقوق مشخص به ازای ساعت کار، میزان ساعات کار کرده و نوع مشخصی از کاغذ که رو کارمند روی آن کار می کند مقدار دهی شوند.

```
1 |      @Override
2 |      public String work() {
3 |          // TODO
4 |      }
```

کار کردن کارمند را نشان می دهد. به ازای هر بار فراخوانی این متود، باید یک ساعت به ساعات کار کرده اضافه شود. همچنین رشته ای با فرمت زیر باید برگردانده شود.

```
{name} worked 1 hour on {paperType}. Total: {hoursWorked} hours.
```

```
1 |      @Override
2 |      public double calculateSalary() {
3 |          // TODO
4 |      }
```

میزان حقوق پایه کارمند از ضرب نرخ حقوق به ازای ساعت کاری در میزان ساعات کار کرده به دست می آید. بسته به نوع کاغذی که کارمند مسئول آن است، یک ضریب به حقوق پایه اضافه می شود. ضرایب برای هر کاغذ به صورت زیر محاسبه می شوند:

```
RECYCLED:   1.0
GLOSSY:     1.1
CARDSTOCK:  1.2
NEWSPRINT:  1.3
PARCHMENT:  1.4
```

به عنوان مثال فردی که روی CARDSTOCK کار می کند، بعد از محاسبه حقوق پایه، ۱.۲ برابر حقوق پایه را دریافت می کند.

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

در صورت ترفیع گرفتن کارمند، نرخ حقوق به ازای ساعت کاری او (hourlyRate) ۲۰ درصد افزایش پیدا می کند. در اینجا باید از متود promote کلاس والد هم استفاده شود.

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

در صورت تنزیل کارمند، نرخ حقوق به ازای ساعت کاری او ۱۰ درصد کاهش پیدا می کند. در اینجا باید از متود demote کلاس والد هم استفاده شود.

## تست

در صورت اجرای کد زیر، خروجی باید تطابق داشته باشد.

```
public class Main {
    public static void main(String[] args) {
        Worker w1 = new Worker("Pam Beesly", 10, 19, PaperType.R
        Worker w2 = new Worker("Jim Halpert", 20, 10, PaperType.

        Manager m = new Manager("Michael Scott", 3000);
        m.addTeamMember(w1);
        m.addTeamMember(w2);

        System.out.println(w1.work() + " Salary: $" + w1.calcula
        System.out.println(w2.work() + " Salary: $" + w2.calcula
        System.out.println(m.work() + " Salary: $" + m.calculate
    }
}
```

خروجی:

```
Pam Beesly worked 1 hour on RECYCLED. Total: 20 hours. Salary: $200.0
Jim Halpert worked 1 hour on CARDSTOCK. Total: 11 hours. Salary: $264
Michael Scott manages: Pam Beesly, Jim Halpert Salary: $3646.4
```

## آنچه باید بارگذاری کنید:

متود هایی که با `//TODO` علامت گذاری شده اند را تکمیل کنید.

به فاصله بین کلمات و علائم نگارشی در دستورات توجه داشته باشید.

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

# اسنپ! فود

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: متوسط
- طراح: احسان حبیب آگهی

در یک روز معمولی در دفتر شرکت فناوری "کدویزارد"، شما به عنوان یک توسعه‌دهنده ارشد، پروژه‌ای مرموز دریافت می‌کنید. مدیرتان با چهره‌ای جدی وارد می‌شود و می‌گوید:"ما یک مجموعه کد و فایل ناقص از برنامه اسنپ‌فود دریافت کرده‌ایم. این کدها بخشی از یک ماژول حیاتی هستند که به دلایلی ناشناخته ناقص یا حذف شده‌اند. ماموریت شما این است که این کدها را تحلیل کنید، بخش‌های گم‌شده را بازسازی کنید و برنامه را به حالت اولیه بازگردانید."

کد ها را از اینجا دانلود کنید

بخش هایی از کد که کامل نیستند یا خطاهایی دارند که مانع اجرای صحیح برنامه میشوند را اصلاح کنید. هر جا نیاز به اصلاح باشد در کامنت های فایل توضیح داده شده - نیازی به تغییر سایر بخش ها نیست.

## توضیحات (به ترتیب)

### OrderObserver

همه چیز از اینجا شروع میشه! از ابتدا یک کلاس CustomerNotifier و RestaurantNotifier ساخته می‌شود.

### FoodOrder.java

به ازای هر سفارش یک instance از آن ساخته می‌شود. کانستراکتور سفارش بر اساس آیدی و قیمت پایه و فاصله تعیین می‌شوند.

- بخش های ناقص را کامل کنید. در constructor مقدار اولیه status را `PENDING` بگذارید.

- یک آرایه (یا لیست) اضافه کنید که از کلاس observer بتوان نمونه هایی از این کلاس مانند restaurant و client را افزود
- در addObserver() شی داده شده را به آرایه (لیست) اضافه کنید.
- در notifyObservers() به تمام observer ها اعلان دهید. (از طریق update)

در مراحل تست پس از ساخت شی سفارش (یکی از وارث های FoodOrder) کلاینت (مشتری) و رستوران مربوطه را به آن add کنید

## OrderProcessor.java

در مراحل تست پس از ساخت شی از این کلاس سفارش را با استفاده از processOrder() پیگیری کنید. - اینجا تست تمام می‌شود (:

## PricingPlan.java

قیمت نهایی بر اساس قیمت پایه سفارش به علاوه هزینه ارسال محاسبه می شود. هزینه سفارش نیز مجددا بر اساس یک قیمت پایه به علاوه یک مقدار کارمزد به ازای هر کیلومتر مسافت محاسبه می شود.برای محاسبه هزینه ارسال از دستور زیر استفاده کنید

| پلن | قیمت پایه | هزینه به ازای هر کیلومتر |
|---|---|---|
| Standard Delivery | $5 | $0.5 |
| Express Delivery | $8 | $0.7 |
| Long Distance Delivery >10km | $10 | $1.0 |
| Long Distance Delivery <=10km | $6 | $0.5 |

در سایر فایل ها هر آنچه نیاز به تغییر باشد در کامنت ها ذکر شده است.

## Main.java

میتوانید در این فایل برنامه ای که ساختید را تست کنید. این بخش اختیاری است

نمونه ای از یک تست:

```java
OrderObserver customer = new CustomerNotifier();

OrderObserver restaurant = new RestaurantNotifier();

FoodOrder standardOrder = new StandardOrder("012345", 100, 5);

standardOrder.addObserver(customer);

standardOrder.addObserver(restaurant);

OrderProcessor<FoodOrder> processor = new OrderProcessor<>();

processor.processOrder(standardOrder);

FoodOrder expressOrder = new ExpressOrder("067890", 200, 8);

expressOrder.addObserver(customer);

expressOrder.addObserver(restaurant);

processor.processOrder(expressOrder);

FoodOrder longDistanceOrder = new LongDistanceOrder("054321", 15

longDistanceOrder.addObserver(customer);

longDistanceOrder.addObserver(restaurant);

processor.processOrder(longDistanceOrder);
```

## Submission

یک فایل زیپ file_name.zip که شامل کلاس های زیر (تمامی کلاس ها) باشد آپلود کنید:

```
<file_name.zip>

├── ExpressOrder.java

├── FoodOrder.java
```

```
├── LongDistanceOrder.java
├── Main.java - (optional)
├── OrderFactory.java
├── OrderObserver.java
├── OrderProcessor.java
├── OrderStatus.java
├── PricingPlan.java
└── StandardOrder.java
```

# Lord Of The Rings

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: سخت
- طراح: مهرسا سمیع‌زاده

در این بازی، Gollum ، Frodo و Sauron برای پیدا کردن Ring پر قدرت، درمیان نفرین های Middle-Earth، به رقابت می‌پردازند.

## جزئیات برنامه

ابتدا پروژه اولیه را این لینک دانلود کنید.

این بازی شامل بازیکن، و سه شخصیت Sauron ، Gollum و Frodo است ، که در Middle-Earth صورت می‌گیرد. بازیکن ها هرکدام پس از انتخاب شخصیت خود به دنبال حلقه در هزارتوی بازی هستند. نوع حرکت هر کاراکتر متفاوت است.

هر بازیکن تنها می‌تواند در نوبت خودش یک حرکت انجام بدهد. و ترتیب نوبت بازیکنان به ترتیب ادد شدن آنها به بازی بستگی دارد. در مسیر Doom یا نفرین ها باعث می‌شوند که بازیکن دور بعد، نتواند بازی کند . و Luck یا شانس های مسیر نیز به معنای این اند که بازیکن پس از رسیدن به این شانس یک حرکت دیگر نیز می‌تواند انجام دهد.

اگر پس از انجام حرکت توسط یک بازیکن، خانه ای که به آن وارد می‌شود پر باشد(توسط بازیکنی دیگر)، حرکت او خنثی می‌شود و به خانه‌ی قبلی خود برمیگردد، اما از hint بازیکنی که در آن خانه قرار دارد بهره مند می‌شود. توجه داشته باشید که با اینکه حرکت نکرده است و به خانه‌ی قبلی خود باز می‌گردد، اما از نوبت خود استفاده کرده است.

هزارتو را شبیه ماتریس مربعی، حداقل 5*5، در نظر بگیرید. هر بازیکن در شروع برنامه در خانه (0-0) قرار دارد. بازیکنان ممکن است در مسیر به خوش‌شانسی یا نفرین دچار شوند. توجه داشته باشید، نحوه قرار گیری نفرین و شانس ها از الگوریتم مشخصی تبعیت می‌کند. و حلقه در خانه‌ی مرکزی قرار دارد.

برای درک بهتر به تصویر زیر، از هزارتوی 9*9 توجه کنید. ردیف های‌فرد، و زوج مختص هر مانع اند. اختلاف هر دو مانع، در هر ردیف ثابت و برابر با سه خانه است.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | LUCK | | | | LUCK | | |
| DOOM | | | | DOOM | | | | DOOM |
| | | LUCK | | | | LUCK | | |
| DOOM | | | | DOOM | | | | DOOM |
| | | LUCK | | RING | | LUCK | | |
| DOOM | | | | DOOM | | | | DOOM |
| | | LUCK | | | | LUCK | | |
| DOOM | | | | DOOM | | | | DOOM |
| START | | LUCK | | | | LUCK | | |

برای کلاس‌ها getter و setter های مناسب، پیاده‌سازی کنید.

## کلاس Player

که دارای فیلدهای زیر است.

```
1 private String playerName;
2 private Character character;
3 private String location;
4 private int x,y;
5 private boolean doomed,lucky;
```

استرینگ location به فرمت x-y ذخیره می‌شود. سازنده‌ی کلاس به صورت زیر است.

```
public Player(String playerName){
//TODO
}
```

متدهای این کلاس:

```
public String makeMove(String direction){
//TODO
}

public String getHint(){
//TODO
};
```

این دو متد صرفا، متدهای مربوط را از کاراکترشان ریترن می‌کنند.

## کلاس انتزاعی Character

هر سه شخصیت از این کلاس ازث‌بری می‌کنند. که دارای فیلد زیر است.

۱. ویژگی player که از جنس Player است

۲. ویژگی‌های horizontal و vertical که از نوع int هستند و مشخص کننده نوع حرکت عمودی و افقی کاراکتر هستند.

۳. ویژگی middleEarth از نوع MiddleEarth

سازنده‌ی این کلاس به صورت زیر است.

```
public Character(MiddleEarth middleEarth,Player player){
//TODO
}
```

این کلاس شامل متد های زیر است.

```
public abstract String move(String direction);
public abstract String getHint();
```

هر شخصیت نوع حرکت متفاوتی دارد . و همچنین هینت گرفتن برای موقعیت نسبی بازیکن نسبت به حلقه نیز برای هر شخصیت با توانایی ها متفاوت، متفاوت است.

## کلاس های Sauron ، Frodo ، Gollum

این کلاس‌ها از کلاس انتزاعی Character ارث‌بری می‌کنند و فقط همان سازنده را صدا میزنند.

در این کلاس ها شما نیاز دارید که متد های move و getHint را Override کنید.

```
@Override
public String move(String direction){
//TODO
};
@Override
public String getHint(){
//TODO
};
```

*تحوه move هر کاراکتر:*

- به صورت افقی یک خانه می‌تواند جابجا شود و به صورت عمودی نیز یک خانه :Gollum
- یک خانه افقی، دو خانه عمودی :Frodo
- یک خانه عمودی، دو خانه افقی :Sauron

یعنی با دریافت دستور `right` که یک حرکت افقی است، Gollum یک خانه، Frodo دو خانه و Sauron نیز دو خانه به سمت راست می‌رود.

پس از فراخوانی این متد، مواردی که حرکتی صورت نمیگیرد:

- در صورت نفرین بودن `YOU WERE DOOMED` ، برگردانده می‌شود.
- در صورتی که نوبت بازیکن نباشد `IT IS NOT YOUR TURN` برگردانده، می‌شود.
- درصورتی که مختصات مقصد، توسط بازیکنی دیگر پر بود، نوبت بازیکن استفاده می‌شود و هینت ان بازیکن، با توجه به لوکیشن و کاراکتراش، ریترن میشود.
- درصورتی که مختصات مقصد خارج از مرز هزارتو بود، از نوبت استفاده میشود و `<playeName> stays in their location` برگردانده می‌شود.

و اگر حرکت صورت می‌گیرد:

- مختصات مقصد، حاوی شانس بود، پس از انجام حرکت `YOU GOT LUCKY` برگردانده می‌شود.
- و در غیر این موارد، و حرکت موفقیت‌آمیز بازیکن، `<playerName> moved successfully` برگردانده می‌شود.

*تحوه هینت گرفتن هر کاراکتر:* اگر فاصله یک خانه بود `TOO CLOSE!` ، حداکثر سه خانه `ALMOST` و در غیر این صورت `KEEP TRYING` برگردانده شود.

- Gollum: (خانه‌های سمت راست) گالم تنها می‌تواند روبرویش را ببیند،
- Frodo: (خانه‌های سمت چپ) می‌تواند پشت سرش را چک کند،
- Sauron: می‌تواند خانه های بالاتر را چک کند

## کلاس MiddleEarth

این کلاس روند بازی را کنترل می‌کند. دارای فیلد زیر می‌باشد.

۱. ویژگی mazeSize از جنس int (که طول و عرض زمین ما را نشان می‌دهد، نه مساحت را)
۲. ویژگی players که آرایه ای از جنس []Player است.
۳. ویژگی currentTurn که از جنس int است و برای کنترل روند نوبت بازیکنان استفاده می‌شود.
۴. ویژگی obstacle که آرایه ای از جنس []String است و نفرین و شانس های مسیر را ذخیره می‌کند.(به صورت `x-y:Doom` و `x-y:Luck` )

سازنده‌ی این کلاس به صورت:(تضمین می‌شود سایز ورودی، حتما عددی فرد و حداقل 5 است)

```
1 public MiddleEarth(int mazeSize){
2 //TODO
3 }
```

متد های این کلاس:

```
1 public void setObstacles(int mazeSize){
2 //TODO
3 }
```

در این متد DOOM و LUCK در خانه های مورد نظرما ست می‌شوند.(با استفاده از آرایه obstacle) می‌توانید با توجه به تصویر ابتدای سوال، الگوی این موانع را پیدا کنید.

```
public boolean checkGameEnd(){
//TODO
}
```

این متد در صورت اتمام بازی و پیدا شدن حلقه توسط یک بازیکن، `Player <player name> found the ring` را چاپ می‌کند و true را برمی‌گرداند.

```
public boolean addPlayer(Player player){
//TODO
}
```

برای اینکه بازیکنان بتوانند بازی کنند، باید آنهارا در بازی ادد کنیم و برای اینکار از این متد و آرایه مورد نیاز در پراپرتی این کلاس استفاده می‌کنیم.

```
public Player getCurrentPlayer(){
//TODO
}
public void nextTurn() {
//TODO
}
```

این دو متد کمک می‌کنند که سیستم رعایت نوبت بازیکنان رعایت شود. متد اول بازیکنی که نوبتش است را برمی‌گرداند و متد دوم بازیکن بعدی را مشخص می‌کند و currentTurn را آپدیت می‌کند.

▼ تفاوت makeMove و move

توجه داشته باشید که هر پلیر با متد makeMove درخواست حرکت می‌کند، و این متد بیشتر مانند متدی کنترلی عمل می‌کند تا مطمئن شود نوبت ها یا نفرین بودن یا نبودن ها رعایت شده و در واقع روند حرکت بازیکن را کنترل می‌کند. در حالی که متد move بیشتر بر نوع حرکت و جابه‌جایی نهایی بازیکنان نظارت دارد.

توجه داشته باشید که در صورت نیاز می‌تواند از مقدار بازگردانده شده از متد move، در متد makeMove استفاده کنید.

درواقع به عبارتی makeMove درخواست حرکت را چک می‌کند، که آیا بازیکن ما شرایط حرکت را دارد، و متد move حرکت را انجام می‌دهد و نتیجه حرکت را گزارش می‌دهد.

## مثال

به گونه‌ای پیاده سازی کنید که با اجرای **Main** زیر:

```
public class Main {
    public static void main(String[] args) {
        MiddleEarth middleEarth = new MiddleEarth(9);
        Player frodo = new Player("Frodo");
        Player gollum = new Player("Gollum");
        Player sauron = new Player("Sauron");

        Character frodoCharacter = new Frodo(middleEarth, frodo)
        Character gollumCharacter = new Gollum(middleEarth, goll
        Character sauronCharacter = new Sauron(middleEarth, saur

        frodo.setCharacter(frodoCharacter);
        gollum.setCharacter(gollumCharacter);
        sauron.setCharacter(sauronCharacter);

        middleEarth.addPlayer(gollum);
        middleEarth.addPlayer(frodo);
        middleEarth.addPlayer(sauron);

        System.out.println(gollum.makeMove("right"));
        System.out.println(frodo.getHint());
        System.out.println(gollum.makeMove("right"));
        System.out.println(frodo.makeMove("up"));
        System.out.println(sauron.makeMove("up"));
        System.out.println(gollum.makeMove("up"));
        System.out.println(frodo.makeMove("down"));
        System.out.println(sauron.makeMove("left"));
        System.out.println(gollum.makeMove("up"));
        System.out.println(middleEarth.getCurrentPlayer().getPla
        System.out.println(gollum.makeMove("right"));
        System.out.println(sauron.getHint());
    }
}
```

خروجی به این شکل باشد.

```
Gollum moved successfully
KEEP TRYING
IT IS NOT YOUR TURN
Frodo moved successfully
Sauron moved successfully
Gollum moved successfully
Frodo moved successfully
YOU WERE DOOMED
Gollum moved successfully
Frodo
IT IS NOT YOUR TURN
KEEP TRYING
```

## آنچه که باید آپلود کنید:

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

```
<zip_file_name.zip>
    ├── Player.java
    ├── Character.java
    ├── Gollum.java
    ├── Frodo.java
    ├── Sauron.java
    └── MiddleEarth.java
```