کلاس خصوصی

- نام طراح: عرفان مشیری
- سطح سوال: ساده

دانشکده کامپیوتر دانشگاه شهید بهشتی به علت کسب درآمد بیشتر تصمیم گرفته است تا برای دانشجویان کلاس‌های خصوصی با اساتید ترتیب دهد! تا اینگونه بخشی از کسری بودجه خود را جبران کرده و بتوانند برای بچه‌ها لابی بسازد. بدین منظور، دانشجویان سال بالایی قطعه کد زیر را پیاده سازی کرده‌اند تا مسئولین بتوانند این کلاس‌ها را ثبت و مدیریت کنند.

ابتدا فایل‌های اولیه پروژه رو از این لینک دانلود کنید

برای نمایش اساتید و دانشجویان از کلاس‌های Teacherو Student استفاده شده که هردو از کلاس Person ارث‌بری میکنند و **تنها دارای یک ویژگی private به نام name** میباشند:

```
class Person {
    public Person(String name) {
        this.name = name;
    }

    private final String name;

    public String getName() {
        return name;
    }
}

class Teacher extends Person {
    public Teacher(String name) {
        super(name);
    }
}

class Student extends Person {
```

```
    public Student(String name) {
        super(name);
    }
```

برای نمایش کلاس‌ها ابتدا از یک interface با نام Class استفاده شده است که دارای یکسری متد میباشد:

```
interface Class {
    int getExamMark();

    int getFee();

    int getNumOfSessions();

    int getHeldSessions();

    boolean[] getPresence();

    void setRating(int rate);

    String toString();
}
```

همچنین یک کلاس abstract به نام AbstractClass تعدادی property به آن اضافه کرده و درواقع آن را کامل‌تر کرده است *(implement)* :

```
abstract class AbstractClass implements Class {

    public AbstractClass(Teacher t, Student s, int fee, int numOfSessions) {}

    protected Teacher teacher;
    protected Student student;

    protected int fee;
    protected int numOfSessions;
    protected int heldSessions;
    protected boolean[] presence;  //size is numOfSessions

```

```
    protected int examMark;
    protected int teacherRating;
    protected int studentRating;
}
```

- هرکلاس یک استاد و یک دانشجو دارد تا اعضای کلاس را مشخص کند
- fee نشان دهنده قیمت کلاس است
- numOfSessions تعداد کل جلسات را نشان میدهد
- heldSessions تعداد جلسات برگزار شده تا الان را نشان میدهد
- presence یک آرایه از جنس boolean به اندازه تعداد کل جلسات است که هر خانه n آن نشان میدهد جلسه n+1 دانشجو حضور داشته است یا خیر
- examMark نمره دانشجو در آخر ترم را نشان میدهد
- teacherRating امتیاز داده شده به کلاس توسط استاد است(یک تا ۵)
- studentRating امیتاز داده شده به کلاس توسط دانشجو است(یک تا پنج)

ولی همانطور که میدانید کلاس abstract قابل نمونه‌سازی نیست. حال بعنوان دانشجویان جدید این دانشکده نوبت شماست تا مهارت‌های خود را ثابت کرده و این کلاس را پیاده سازی کنید. اما از آنجا که دانشجویان یا اساتید با توجه به نوعشان ویژگی‌ها و دسترسی‌های متفاوتی دارند. شما باید ۲ کلاس متفاوت بنویسید که از این کلاس abstract ارث‌بری میکنند.

کلاس‌ها با نام‌های **StudentClass** و **TeacherClass** دارای یکسری عملکردهای متفاوت و یکسری عملکرد شبیه هستند

- در ابتدا، هردو آنها constructorهایی مشابه کلاس AbstractClass دارند که باید پیاده سازی شوند. (فراموش نکنید که آرایه presence را هم new کنید)

متدهای getter مشترک

هردو کلاس باید متدهای getter برای fee, numOfSessions, heldSessions, presence, rating, examMark را **بطور یکسان** پیاده سازی کنند. بدین منظور با افزودن get به ابتدای نام هریک متدشان را میسازیم و

ویژگی مربوطه را بعنوان خروجی برمیگردانیم. مانند نمونه زیر:

```
1 | public int getNumOfSessions() { return numOfSessions; }
```

متدهای setter مخصوص استاد

یکسری از متدها مختص استاد هستند و باید تنها در کلاس مربوطه پیاده سازی شوند.

- استاد در متد setHeldSessions با دادن یک ورودی که شماره جلسه است، متغیر heldSessions را اگر درحال حاضر کوچکتر از ورودی بود، به عدد جدید آبدیت میکند و در غیراین صورت تغییری در آن ایجاد نمیکند.
- استاد همچنین باید متد setPresence را پیاده سازی کند. این متد با دریافت یک عدد که شماره جلسه است، در آرایه حضور و غیاب خانه مربوطه را true میکند. همچنین در نهایت متد setHeldSessions را با همان شماره جلسه صدا میزند تا تعداد جلسات برگزار شده تا الان آبدیت شود.
- در پایان ترم استاد با صدا زدن متد setExamMark به دانشجو نمره میدهد و true برمیگرداند. ولی اگر تعداد جلسات برگزار شده تا الان کمتر از تعداد کل جلسات بود نمره ثبت نشده و false برگردانده میشود.

متدهای setter مشترک

ولی تعدادی از متدها در هردو کلاس وجود دارند و باید بصورت متفاوت پیاده سازی شوند.

- متد set_rating برای هردو پیاده سازی میشود. اگر استاد این متد را صدا زد، متغیر teacher_rating مقداردهی میشود و در حالت دیگر متغیر student_rating.

درنهایت متد *toString* نیز که با آن آشنایی دارید برای هردو کلاس بدین نوع پیاده سازی میشود که برای استاد نام شاگرد که جز ویژگی‌های کلاس student است را برمیگرداند. و در کلاس شاگرد نام استاد را که از ویژگی‌های کلاس teacher است.

تمام کلاس‌های داده شده در فایل‌های اولیه پروژه را به همراه کلاس‌های پیاده سازی شده در یک فایل به نام Main.java ذخیره کرده و ارسال نمایید

انتگرال چندگانه

- نام طراح: عرفان مشیری
- سطح سوال: ساده

میخواهیم کلاس‌هایی طراحی کنیم که انتگرال را محاسبه میکنند. و آنجا که برنامه نویسی امکانات فوق‌العاده‌ای به ما میدهد تصمیم داریم انتگرال را تا ۳ مرحله محاسبه کنیم (انتگرال ۳گانه).

هرکلاس انتگرال به محاسبه انتگرال خود میپردازد. و برای جلوگیری از کدزنی اضافه کلاس‌ها از یکدیگر ارث بری میکنند، چون عملا کارکرد هرکدام شبیه دیگری است :)

اما نگران نباشید، از آنجا که این نوع انتگرال خیالی است، فقط برای محاسبه نتیجه کافی است تا به **ابتدای** عبارتی که میخوهیم انتگرالش را محاسبه کنیم % اضافه شود. مثلا انتگرال اول و دوم x به ترتیب برابر است با

> %x
>
> %%x

در زیر کلاس های ۳گانه انتگرال را مشاهده میکنید:

```
1  class Integral1 {
2
3      private String result;
4
5      public String getResult() {
6          return result;
7      }
8
9      public Integral1(String s) {
10     // TODO
11     }
12
13     public void calIntegral() {
14         result = "%" + result;
```

```
    }

}

class Integral2 extends Integral1 {
    public Integral2(String s) {
    // TODO
    }
}

class Integral3 extends Integral2 {
    public Integral3(String s) {
    // TODO
    }
}
```

از آنجا که کلاس‌های انتگرال بسیار باهوش هستند، خود با یکبار ساخته شدن (صدا زدن سازنده‌اش) و دریافت ورودی، انتگرال را تا مرحله موردنظر گرفته و در result ذخیره میکند.

شما باید constructorهای این سه کلاس (بخش‌های *TODO*) را به گونه‌ای پیاده سازی کنید که با یکبار صدا زدن هر انتگرال نتیجه آن عبارت در result ذخیره شود.

به این مثال توجه کنید:

```
Integral3 integral3 = new Integral3("(2f-3)");
// expecting: integral3.getResult() = "%%%(2f-3)"

Integral2 integral2 = new Integral2("76");
// expecting: integral2.getResult() = "%%76"
```

دقت کنید نیاز به چاپ کردن چیزی نیست. همچنین **هیچ تابع یا متغیر دیگری به کلاس‌ها اضافه نکنید.**

در نهایت ۳ کلاس کامل شده را در یک فایل با نام Integral.java ذخیره کرده و آپلود کنید.

گرم و سرد

- نام طراح: میثم باوی
- سطح سؤال: آسان

یک شرکت لوازم خانگی در حال معرفی برند جدیدی از محصولات خود است. این محصولات با استفاده از برق، آب را گرم یا سرد می کنند. مدیر فنی شرکت از شما خواسته است برنامه ای بر مبنای مفاهیم شیءگرایی بنویسید که بعداً بتوان با کمترین تغییر، در باقی محصولات استفاده کرد. ویژگی اصلی این برنامه، محاسبهٔ هزینهٔ آب و برق دستگاه، با استفاده از داده های ورودی مثل دمای آب است.

پروژهٔ اولیه را از این لینک دانلود کنید.

کلاس انتزاعی Thermostat

این کلاس، کلاس والد برنامهٔ مخصوص یک دستگاه است. وظیفهٔ یک Thermostat، رساندن دمای محیط به دمای دلخواه (desired) است.

متد `setDesiredDegreeCelsius`

```
1 | public void setDesiredDegreeCelsius(double degreeCelsius)
```

این متد با گرفتن یک عدد، فیلد دمای مورد نظر کاربر را مقداردهی می کند. واحد دمای ورودی **درجهٔ سلسیوس** است.

متد `setDesiredDegreeFahrenheit`

```
1 | public void setDesiredDegreeFahrenheit(double degreeFahrenheit);
```

این متد با گرفتن یک عدد، فیلد دمای مورد نظر کاربر را مقداردهی می کند. واحد دمای ورودی **درجهٔ فارنهایت** است. نکته: تنها یک فیلد برای دمای مورد نظر کاربر وجود دارد و لازم است با تبدیل یکی از دماها به دیگری، این فیلد را مقدار دهی کنید.

متد `getDesiredDegreeCelsius`

```
1 | public double getDesiredDegreeCelsius();
```

این متد مقدار دمای مورد نظر کاربر را در واحد **درجهٔ سلسیوس** برمیگرداند.

متد `setWaterDegreeCelsius`

```
1 | public void setWaterDegreeCelsius(double degreeCelsius);
```

این متد، دمای آب ورودی به دستگاه را در فیلد مربوطه مقداردهی میکند.

متد `use`

```
1 | public void use(double liter);
```

این متد هنگامی صدا زده میشود که حجم مشخصی از آب از دستگاه خارج و مصرف میشود. هر دستگاه با توجه به ویژگی هایش، دمای آب را تغییر می دهد و برای این کار **انرژی** مصرف میکند. میزان آب و انرژی مصرفی، در این متد، به یک پیاده سازی از `BillCalculator` داده می شود.

کلاس Heater

این کلاس، کلاس انتزاعی `Thermostat` را پیاده سازی میکند. ویژگی های دستگاه Heater از این قبیل است:

- نمیتواند آب ورودی را سردتر کند؛ اگر دمای قابل تنظیم (desired) کمتر از دمای آب باشد، نرخ انرژی مصرفی M خواهد بود.
- بیشترین دمای **قابل تنظیم (desired)**، **80** درجهٔ سلسیوس است.
- میزان مصرف انرژی **به ازای هر لیتر** مصرفی، از فرمول زیر محاسبه میشود.

$$(desiredTemperature - currentWaterTemperature) \times Coefficient$$

- اگر فرمول بالا، عددی کمتر از M بدهد، نرخ انرژی مصرفی M خواهد بود.
- دمای قابل تنظیم (desired)، **وابسته به دمای آب نیست** و کاربر هر مقداری از 80 به پایین را باید بتواند وارد کند.

سازنده

```
1 | public Heater(BillCalculator billCalculator, double coefficient, double minimum)
```

- پارامتر `coefficient` ضریب مثبت محاسبهٔ انرژی مصرفی است.
- پارامتر `minimum` ، حداقل نرخ انرژی مصرفی است.
- پارامتر `billCalculator` ، برای محاسبهٔ قبض آب و برق است.

کلاس Cooler

این کلاس، کلاس انتزاعی `Thermostat` را پیاده سازی میکند. ویژگی های دستگاه Cooler از این قبیل است:

- نمیتواند آب ورودی را گرمتر کند؛ اگر دمای قابل تنظیم (desired) بیشتر از دمای آب باشد، نرخ انرژی مصرفی 0 خواهد بود.
- کمترین دمای **قابل تنظیم (desired)**، **10** درجهٔ سلسیوس است.
- میزان مصرف انرژی **به ازای هر لیتر** مصرفی، از فرمول زیر محاسبه میشود.
- دستگاه دمای آب را به میزان **محدودی** میتواند تغییر دهد.

$$(desiredTemperature - currentWaterTemperature)^2 \times Coefficient$$

- اگر فرمول بالا، عددی کمتر از 0 بدهد، نرخ انرژی مصرفی 0 خواهد بود.
- دمای قابل تنظیم (desired)، **وابسته به دمای آب نیست** و کاربر هر مقداری از 10 به بالا را باید بتواند وارد کند.

سازنده

```
1 | public Cooler(BillCalculator billCalculator, double coefficient, double coolingR
```

- پارامتر `coefficient` ضریب مثبت محاسبهٔ انرژی مصرفی است.
- پارامتر `coolingRange` ، یک عدد مثبت و برابر با حداکثر میزان تغییر دمای آب (در واحد درجهٔ سلسیوس) توسط دستگاه است. مثلاً اگر برابر با 20 باشد، و دستگاه بخواهد آب را از 40 به 15 درجهٔ سلسیوس تغییر دهد، تنها میتواند آب را تا 20 درجهٔ سلسیوس خنک کند. این مقدار در نرخ انرژی مصرفی **تأثیر دارد**.
- پارامتر `billCalculator` ، برای محاسبهٔ قبض آب و برق است.

واسط `BillCalculator`

یک واسط برای کلاس هایی است که با گرفتن مقادیر انرژی و برق مصرفی، هزینهٔ نهایی را محاسبه میکنند.

متد `addEnergy`

```
1 | public void addEnergy(double energyConsumed)
```

میزان انرژی مصرفی را می گیرد و به مجموع مقادیر قبلی اضافه میکند.

متد `addWater`

```
public void addWater(double waterConsumed)
```

میزان آب مصرفی را می گیرد و به مجموع مقادیر قبلی اضافه میکند.

متد getBill

```
public double getBill()
```

مجموع هزینهٔ آب و انرژی مصرفی را بر اساس نرخ مشخصی محاسبه میکند. نرخ محاسبه، بسته به پیاده سازی این واسط دارد.

متد reset

```
public void reset()
```

مجموع مقادیر قبلی انرژی و آب را 0 میکند.

کلاس SummerBillCalculator

این کلاس، واسط `BillCalculator` را پیاده سازی میکند. نرخ محاسبهٔ در این کلاس، مربوط به فصل تابستان و به صورت زیر است:

- 10 واحد هزینه به ازای هر واحد انرژی مصرفی
- 20 واحد هزینه به ازای هر لیتر از 3000 لیتر آب اولیه، و 60 واحد هزینه به ازای باقی مصرف (برای مثال، هزینه برای 4000 لیتر آب مصرفی، 120000 است)

نمونه

```
public class Main {
    public static void main(String[] args) {
        BillCalculator summerBill = new SummerBillCalculator();
```

```
        Thermostat heater = new Heater(summerBill, 2.2, 30);

        heater.setWaterDegreeCelsius(30);
        heater.setDesiredDegreeCelsius(70);
        System.out.println(heater.getDesiredDegreeCelsius());
        heater.use(30);
        heater.use(20);
        System.out.println(summerBill.getBill());
        System.out.println();

        heater.setWaterDegreeCelsius(30);
        heater.setDesiredDegreeCelsius(90);
        System.out.println(heater.getDesiredDegreeCelsius());
        heater.use(15);
        System.out.println(summerBill.getBill());
        System.out.println();

        heater.setDesiredDegreeFahrenheit(153);
        System.out.println(heater.getDesiredDegreeCelsius());
        heater.use(10);
        System.out.println(summerBill.getBill());
        summerBill.reset();
        System.out.println();

        heater.setDesiredDegreeCelsius(30);
        System.out.println(heater.getDesiredDegreeCelsius());
        heater.use(10);
        System.out.println(summerBill.getBill());
        System.out.println();
    }
}
```

خروجی

```
70.0
45000.0

80.0
```

```
61800.0

67.22222222222223
70188.88888888889

30.0
3200.0
```

(کمتر از 4 رقم اعشار تفاوت در اعداد اعشاری به دست آمده ایرادی ندارد)

آنچه باید آپلود کنید

فایل های `Thermostat.java` و `Cooler.java` و `Heater.java` و `SummerBillCalculator.java` را بدون هیچگونه پوشه بندی، مستقیم زیپ و ارسال کنید.

مافیا

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- نام طراح: مهرسا عرب زاده
- سطح سوال: متوسط

پیشنهاد میشود ابتدا یک بار سوال را به طور کامل بخوانید سپس اقدام به کدزنی کنید.

مهرسا به شدت به بازی مافیا علاقه مند است. او که به تازگی برنامه نویسی یاد گرفته است میخواهد با استفاده از آن یک بازی مافیا طراحی کند اما نمی داند از کجا شروع کند. در این سوال قصد داریم یک پیاده سازی اولیه از بخش کوچکی از بازی مافیای کلاسیک را انجام دهیم تا مهرسا در فرصتی مناسب آن را گسترش داده و بخش های باقی مانده ی آن را پیاده سازی کند.

ابتدا پروژه اولیه را از این لینک دانلود کنید.

در این بازی در زمان شب، مافیا به مرگ یک بازیکن رای می دهند و نهایتا با رای گادفادر، رییس مافیا، یک بازیکن کشته میشود. همچنین یک دکتر وجود دارد که هرشب یک نفر را نجات میدهد، و اگر شخص نجات داده شده توسط دکتر، با شخص کشته شده توسط گادفادر یکسان باشد؛ آن شخص زنده می ماند.

در زمان روز، بازیکنان به مرگ یک بازیکن رای می دهند و بازیکنی که رای بیشتری داشته باشد کشته میشود، اگه رای چند بازیکن برابر و از همه بیشتر باشد، در رای گیری روزانه کسی کشته نمی شود.

همچنین بازی دارای یک خدا می باشد که به بازی نظارت میکند و از هویت تمام بازیکنان و اشخاص کشته شده آگاه است.

کلاس انتزاعی Player

این کلاس دارای فیلدهای زیر است:

۱. ویژگی name از جنس String

۲. ویژگی voted از جنس String

که name نام بازیکن و voted نام شخصی که بازیکن در رای گیری روزانه به او رای می دهد را نگه میدارد.

این کلاس دارای یک constructor است که ویژگی name را مقداردهی می کند.

همچنین دارای یک متد با امضای زیر است:

```
public String vote(){
}
```

که نام شخصی که بازیکن در رای گیری روزانه به او رای داده است(voted) را برمیگرداند.

همچنین این کلاس دارای یک متد انتزاعی با امضای زیر نیز می باشد:

```
public abstract Player action();
```

کلاس Mafia

این کلاس از کلاس Player ارث بری می کند و دارای فیلد زیر است:

- ویژگی nightVote از جنس Player

که بازیکنی که مافیا در رای گیری شبانه به مرگ او رای می دهند را نگه میدارد.

پیاده سازی متد action در این کلاس پس از تعریف کلاس Godfather توضیح داده می شود.

کلاس Godfather

این کلاس یک singleton است که از کلاس Mafia ارث بری می کند و دارای فیلدهای زیر است:

۱. ویژگی استاتیک godfather از جنس Godfather

۲. ویژگی finalVote از جنس Player

۳. ویژگی استاتیک names از جنس String[]

۴. ویژگی استاتیک size از جنس int

که finalVote بازیکنی که گادفادر در شب به مرگش رای میدهد و names نام کسانی که سایر اعضای مافیا به مرگش رای داده بودند را نگه میدارد و size اندازه ی آرایه ی names می باشد.

- تضمین می شود بیش از 10 مافیا در بازی نباشند.

پس تمامی رای های سایر اعضای مافیا به اطلاع گادفادر می رسد و در آرایه ی names نگه داری می شود.

نکته: در مورد singleton سرچ کنید.

این کلاس دارای متدهای زیر است:

```
1 public static Godfather godfather(){
2 }
```

که فیلد godfather موجود در کلاس را برمیگرداند.

```
1 public static String[] addVote(String name){
2 }
```

که name موجود در آرگومانش را به آرایه ی names اضافه میکند.

در اینجا پیاده سازی متد action در کلاس Mafia توضیح داده می شود.

این متد نام بازیکن nightVote که یک فیلد در کلاس Mafia بود را به کمک متد addVote پیاده سازی شده در کلاس Godfather به آرایه ی names موجود در کلاس Godfather اضافه می کند. در واقع این متد وظیفه ی انتقال اطلاعات رای گیری شبانه ی مافیا به گادفادر را دارد و بازیکن nightVote را نیز برمیگرداند.

متد action در کلاس Godfather نیز Override میشود.

این متد بازیکن finalVote را از آرایه ی بازیکنان بازی(در کلاس God که در ادامه تعریف شده است موجود است.) حذف میکند و همین بازیکن را نیز برمیگرداند.

کلاس Villager

این کلاس نیز از کلاس Player ارث بری میکند ولی متد action آن کاری انجام نمی دهد و مقدار null را برمیگرداند.

کلاس Doctor

این کلاس نیز یک singleton است که از کلاس Villager ارث بری می کند و دارای فیلدهای زیر است:

۱. ویژگی استاتیک doctor از جنس Doctor
۲. ویژگی saved از جنس Player

که Saved بازیکنی است که دکتر در شب نجات می دهد.

این کلاس دارای متدهای زیر است:

```
1 public static Doctor doctor(){
2 }
```

که فیلد doctor موجود در کلاس را برمیگرداند.

متد action را در این کلاس به گونه ای Override کنید که اگر بازیکن نجات یافته توسط دکتر با بازیکن کشته شده توسط گادفادر یکسان باشند، (ملاک یکسان بودن بازیکنان نام آن هاست.) آن بازیکن مجددا به آرایه ی بازیکنان بازی اضافه شود.

کلاس God

این کلاس دارای فیلدهای زیر است:

۱. ویژگی استاتیک players از جنس Player[]
۲. ویژگی استاتیک size از جنس int

که players لیست بازیکنان بازی و size تعداد آن ها را نگه میدارد.

- تضمین می شود بیش از 100 بازیکن نداشته باشیم.

این کلاس دارای متدهای زیر است:

```
public static String killedAtDay(){
}
```

این متد رای روزانه ی تمام بازیکنان را جمع آوری میکند و اگر شخصی در روز کشته شود، او را از لیست بازیکنان حذف کرده و نام او را برمیگرداند؛ همچنین اگر کسی در روز کشته نشود null برمیگرداند.

```
public static String playerTypes(){
}
```

این متد نام تمام بازیکنان و نقش آن ها را به شکل یک String با فرم زیر برمیگرداند:

```
name: role
```

که این فرمت دارای size خط می باشد.(در هر خط نام و نقش یک بازیکن می آید.)

مثال

پروژه را به گونه ای پیاده سازی کنید که با اجرای Main زیر:

```
public class Main {
    public static void main(String[] args) {
        Mafia x=new Mafia("x");
        Mafia y=new Mafia("y");
        Godfather g=Godfather.godfather();
        g.setName("g");
        Player a=new Villager("a");
        Player b=new Villager("b");
        Doctor d=Doctor.doctor();
        d.setName("d");
        Player[] players = {g, x, y, d, a, b};
```

```
        God.setPlayers(players);
        God.setSize(players.length);

        x.setNightVote(a);
        y.setNightVote(b);
        x.action();
        y.action();
        for(String s: g.getNames()){
            if(s!=null){
                    System.out.print(s+" ");
            }
        }
        System.out.println();

        g.setFinalVote(a);
        g.action();
        System.out.println(God.playerTypes());

        d.setSaved(a);
        d.action();
        System.out.println(God.playerTypes());

        x.setVoted("a");
        y.setVoted("a");
        g.setVoted("b");
        a.setVoted("b");
        b.setVoted("x");
        d.setVoted("y");
        God.killedAtDay();
        System.out.println(God.playerTypes());

        b.setVoted("a");
        God.killedAtDay();
        System.out.println(God.playerTypes());
    }
}
```

خروجی به شکل زیر باشد:

```
a b

g: Godfather
x: Mafia
y: Mafia
d: Doctor
b: Villager

g: Godfather
x: Mafia
y: Mafia
d: Doctor
b: Villager
a: Villager

g: Godfather
x: Mafia
y: Mafia
d: Doctor
b: Villager
a: Villager

g: Godfather
x: Mafia
y: Mafia
d: Doctor
b: Villager
```

آنچه باید آپلود کنید

یک فایل Zip آپلود کنید که هنگامی که آن را باز می کنیم یک فایل God.java در آن موجود باشد که شامل تمامی کلاس های پیاده سازی شده توسط شماست و کلاس God نیز در آن public است.