جنگ ستارگان

- سطح سوال : متوسط
- طراح سوال : شایان حقیقت

ابتدا پروژه اولیه را این لینک دانلود کنید.

در دنیای **جنگ ستارگان (Star Wars)**، سه نوع شخصیت مهم وجود دارد که به نبرد با دشمنان و حفظ تعادل در کهکشان پرداخته و دارای توانایی‌های خاص خود هستند:

۱. **جدای**: جدای‌ها دروازه‌بانان صلح و عدالت در کهکشان هستند. آنها دارای قدرت‌های ذهنی و مهارت‌های نبرد بالا هستند. هنگام حمله به دشمنان، توانایی آن‌ها برای مقاومت در برابر آسیب‌های وارده بسیار بالا است، اما هنگام استفاده از قدرت‌های ویژه‌شان، سرعتشان کاهش می‌یابد.
۲. **سیث**: سیث‌ها در جستجوی قدرت مطلق هستند و به نابودی مخالفان و گسترش سلطه خود می‌پردازند. قدرت آنها به طور مستقیم به میزان آسیبی که می‌خورند بستگی دارد و با استفاده از تقویت‌ها، قدرت آنها به سرعت افزایش می‌یابد.
۳. **مستقل**: شخصیت‌های مستقل در دنیای جنگ ستارگان به‌طور ویژه به خود متکی هستند و به هیچ جناح خاصی تعلق ندارند. آنها هیچ‌گاه آسیب نمی‌بینند و تقویت‌ها یا توانایی‌های ویژه‌ای از محیط نمی‌پذیرند.

1. کلاس `StarWarsCharacter`

این کلاس **انتزاعی** پایه‌ای برای تمام شخصیت‌های جنگ ستارگان است و شامل ویژگی‌ها و متدهای عمومی است که تمامی شخصیت‌ها از آن‌ها استفاده می‌کنند.

- **ویژگی‌ها:**
  - `int strength` : قدرت شخصیت. قدرت از صفر پایین تر نخواهد آمد .
  - `int speed` : سرعت شخصیت. سرعت از صد نیز بیشتر نخواهد شد .
  - مقادیر به صورتی تعریف شوند که از سایر کلاس ها نیز قابل دسترسی باشند .

```
int strength;

```

```
int speed;
```

- **سازنده:**
  - مقداردهی اولیه `strength` به 50 و `speed` به 30.
- **متدهای عمومی:**
  - `String type()` : نوع شخصیت را برمی‌گرداند (پیش‌فرض `"Unknown Character"`).
  - `String strength()` : قدرت فعلی شخصیت را برمی‌گرداند.
  - `String speed()` : سرعت فعلی شخصیت را برمی‌گرداند.
  - `String attack(int damage)` : باید در کلاس‌های فرزند پیاده‌سازی شود. پیاده‌سازی پیش‌فرض `"ERROR"` را برمی‌گرداند.
  - `String boost(int boostValue)` : باید در کلاس‌های فرزند پیاده‌سازی شود. پیاده‌سازی پیش‌فرض `"ERROR"` را برمی‌گرداند.
  - `void setStrength(int strength)` : تنظیم قدرت شخصیت.
  - `void setSpeed(int speed)` : تنظیم سرعت شخصیت.
  - `int getStrength()` : دریافت قدرت شخصیت.

```
String type() {}
String strength() {}
String speed() {}
String attack(int damage) {}
String boost(int boostValue) {}
public void setStrength(int strength) {}
public void setSpeed(int speed) {}
public int getStrength() {}
public int getSpeed() {}
```

2. کلاس `Jedi`

```
public class Jedi // TODO
{
    // TODO Your implementation here
}
```

این کلاس برای شخصیت‌های **جدای** طراحی شده است و ویژگی‌های خاص آنها را پیاده‌سازی می‌کند.

- **متد `type()`** :
  - بازمی‌گرداند: `"I AM JEDI"`
- **متد `attack(int damage)`** :
  - قدرت شخصیت را به میزان دو برابر آسیب ورودی کاهش می‌دهد.
  - به این صورت که قدرت جدید برابر است با `getStrength() - (2 * damage)` و مقدار منفی را صفر می‌کند.

```
return "My strength is " + getStrength();
```

- **متد `boost(int boostValue)`** :
  - سرعت شخصیت را به میزان ورودی کاهش یافته با 5 افزایش می‌دهد.
  - به این صورت که سرعت جدید برابر است با `getSpeed() + (boostValue - 5)` و مقدار بیشتر از 100 را به 100 محدود می‌کند.

```
return "My speed is " + getSpeed();
```

3. کلاس `Sith`

این کلاس برای شخصیت‌های **سیث** طراحی شده است و ویژگی‌های خاص آنها را پیاده‌سازی می‌کند.

```
public class Sith // TODO
{
    // TODO Your implementation here
}
```

- **متد `type()`** :
  - بازمی‌گرداند: `"I AM SITH"`
- **متد `attack(int damage)`** :
  - قدرت شخصیت را به میزان آسیب ورودی کاهش می‌دهد.
  - به این صورت که قدرت جدید برابر است با `getStrength() - damage` و مقدار منفی را صفر می‌کند.

- **متد `boost(int boostValue)`** :
  - سرعت شخصیت را به میزان ورودی افزایش یافته با 5 افزایش می‌دهد.
  - به این صورت که سرعت جدید برابر است با `getSpeed() + (boostValue + 5)` و مقدار بیشتر از 100 را به 100 محدود می‌کند.

4. کلاس `Independent`

این کلاس برای شخصیت‌های **مستقل** طراحی شده است و ویژگی‌های خاص آنها را پیاده‌سازی می‌کند.

- **متد `type()`** :
  - بازمی‌گرداند: `"I AM INDEPENDENT"`
- **متد `attack(int damage)`** :
  - هیچ‌گونه تغییر در قدرت ایجاد نمی‌کند و `"ERROR"` را برمی‌گرداند.
- **متد `boost(int boostValue)`** :
  - هیچ‌گونه تغییر در سرعت ایجاد نمی‌کند و `"ERROR"` را برمی‌گرداند.

می توانید پیاده سازی خود را با توجه به کلاس تست ارزیابی کنید .

آن‌چه باید آپلود کنید

ساختار فایل `zip` ارسالی باید به صورت زیر باشد:

```
<zip_file_name.zip>
    ├── Sith.java
    ├── Jedi.java
    ├── Independent.java
    └── StarWarsCharacter.java
```

Lord Of The Rings

- سطح: سخت
- طراح: مهرسا سمیع‌زاده

آرمیتا که کودکیش را با Lord Of The Rings سپری کرده، حالا به فکر افتاده که چگونه دینش را به این شاهکار، که کودکیش را از بیهودگی نجات داده، ادا کند. او که دیگر آرام و قرار ندارد، تصمیم گرفت تا دست بکار شود و برای جبران، او نیز نوری به زندگی ها بتاباند و بازیی طراحی کند که 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.

		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
}
```

این متد حرکت بازیکنان را کنترل می‌کند. در صورت نفرین بودن `YOU WERE DOOMED` ، در صورتی که نوبت بازیکن نباشد `IT IS NOT YOUR TURN` و اگر به خانه شانس رسیدند `YOU GOT LUCKY` برگردانده می‌شود. اگر بازیکن شرایط انتخاب حرکت را داشت، اما به دلیل پر بودن خانه مقصد، یا خارج از مرز هزارتو بودن، بازیکن مجبور به بازگشت به لوکیشن قبلی خود می‌شود و `<playerName> stays in their location` برگردانده می‌شود. اما در صورتی که حرکت موفقیت آمیز باشد `<playerName> moved successfully` برگردانده می‌شود.

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
};
```

نحوه move هر کاراکتر:

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

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 انجام حرکت، نتیجه برگردانده می‌شود، که در واقع سه حالت دارد:

- درصورتی که مختصات مقصد، توسط بازیکنی دیگر پر بود، حرکت صورت نمیگیرد (اما نوبت بازیکن استفاده می‌شود) و `<playeName> stays in their location` برگردانده می‌شود.
- اگر 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.

نحوه 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: گالم تنها می‌تواند روبرویش را ببیند
- Frodo: 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.

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

```
public MiddleEarth(int mazeSize){
//TODO
}
```

متد های این کلاس:

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

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:

```java
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);
middleEarth.addPlayer(gollum);
middleEarth.addPlayer(frodo);                          middleEarth.a
System.out.println(gollum.makeMove("right"));
System.out.println(frodo.getCharacterHint());
System.out.println(gollum.makeMove("right"));//not your turn
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().getPlayerName(
System.out.println(gollum.makeMove("right"));
System.out.println(sauron.getCharacterHint());
}
  }
```

خروجی به این شکل باشد.

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

Javagram (The Messenger)

- سطح : متوسط
- طراح : آریا زریاب

جزئیات برنامه

ابتدا پروژه اولیه را از این لینک دانلود کنید.

این تمرین یک سیستم پیام‌رسان را شبیه‌سازی می‌کند که در آن کاربران می‌توانند پیام‌هایی را با استفاده از پیام‌رسان‌های مختلف ارسال و دریافت کنند. سیستم شامل سه پیام‌رسان است: **تلگرام**، **واتساپ** و **اینستاگرام**. کاربران می‌توانند پیام‌ها را از طریق هر یک از این پیام‌رسان‌ها به دیگر کاربران ارسال کنند و سیستم پیام‌ها را با تایم‌استمپ‌ها و وضعیت تحویل ردیابی می‌کند.

▼ ساختار فایل پروژه

```
<zip_file_name.zip>
      ├── Messenger.java
      ├── Telegram.java
      ├── Whatsapp.java
      ├── Instagram.java
      ├── Message.java
      ├── User.java
      └── MessageStatus.java
```

اینترفیس `Messenger`

این اینترفیس نوع پیام رسان را مشخص میکند.

۱. متد `sendMessage` : پیام را به گیرنده ارسال می‌کند.

```java
void sendMessage(String message, User receiver);
```

۲. متد `receiveMessage` : دریافت پیام از فرستنده را مدیریت می‌کند

```
String receiveMessage();
```

۳. متد `getMessengerName` : نام پیام‌رسان را برمی‌گرداند.

```
String getMessengerName();
```

۴. متد `setLastMessage` : آخرین متن مسیج پیام رسان را آپدیت میکند.

```
public void setLastMessage(String message);
```

کلاس Telegram

- این کلاس اینترفیس `Messenger` را `implement` میکند.

پراپرتی ها

۱. `lastMessage` : آخرین متن پیام را نگه میدارد و از جنس `String` و سطح دسترسی `private` میباشد

متود ها

۲. `sendMessage` : باید متن زیر را *چاپ* کند.

`Message sent via Telegram: + message`

▼ مثال

به طور مثال اگر متن پیام `hi` باشد خروجی `Message sent via Telegram: hi` میباشد

۲. `receiveMessage` : باید متن زیر را *برگرداند*.

`Message received via Telegram: + lastMessage`

۳. `getMessengerName` : نام کلاس که همان `Telegram` است را *برمیگرداند*.

کلاس Whatsapp

- این کلاس اینترفیس `Messenger` را `implement` میکند.

پراپرتی ها

۱. `lastMessage` : آخرین متن پیام را نگه میدارد و از جنس `String` و سطح دسترسی `private` میباشد

متود ها

۲. `sendMessage` : باید متن زیر را *چاپ* کند.

```
Message sent via Whatsapp: + message
```

▼ مثال

به طور مثال اگر متن پیام `hi` باشد خروجی `Message sent via Whatsapp: hi` میباشد

۲. `receiveMessage` : باید متن زیر را *برگرداند*.

```
Message received via Whatsapp: + lastMessage
```

۳. `getMessengerName` : نام کلاس که همان `Whatsapp` است را *برمیگرداند*.

کلاس Instagram

- این کلاس اینترفیس `Messenger` را `implement` میکند.

پراپرتی ها

۱. `lastMessage` : آخرین متن پیام را نگه میدارد و از جنس `String` و سطح دسترسی `private` میباشد

متود ها

۲. `sendMessage` : باید متن زیر را *چاپ* کند.

```
Message sent via Instagram: + message
```

▼ مثال

به طور مثال اگر متن پیام `hi` باشد خروجی `Message sent via Instagram: hi` میباشد

۲. `receiveMessage` : باید متن زیر را *برگرداند*.

```
Message received via Instagram: + lastMessage
```

۳. `getMessengerName` : نام کلاس که همان `Instagram` است را *برمیگرداند*.

کلاس `Message`

هر پیام را تعریف میکند.

پراپرتی ها

۱. `content` : متن پیام را نگه میدارد که از جنس `String` است.
۲. `sender` : فرستنده پیام که از جنس `User` است.
۳. `receiver` : گیرنده پیام که از جنس `User` است.
۴. `messenger` : نوع پیام رسان که از جنس `Messenger` است.
۵. `sentTimestamp` : زمان فرستادن پیام که از جنس `LocalDateTime` است.
۶. `deliveredTimestamp` : زمان رسیدن پیام به گیرنده که از جنس `LocalDateTime` است.
۷. `readTimestamp` : زمان دیدن پیام توسط گیرنده که از جنس `LocalDateTime` است.
۸. `status` : وضعیت پیام که از جنس `MessageStatus` است.

تمام پراپرتی ها سطح دسترسی `private` را دارند.

کانستراکتور

این کلاس تنها یک متود سازنده دارد که به ترتیب مقادیر `content` و `sender` و `receiver` و `messenger` را ورودی میگیرد و آن هارا مقداردهی میکند، همچنین زمان فرستادن پیام و وضعیت آن را مشخص میکند.

متود ها

۱. `markAsDelivered` : پیام را به وضعیت `DELIVERED` تغییر میدهد.

۲. `markAsRead` : پیام را به وضعیت `READ` تغییر میدهد.

توجه : متود `toString` را تغییر ندهید.

کلاس `User`

پراپرتی ها

۱. `name` : نام کاربر را نگه میدارد که از جنس `String` است.
۲. `sentMessages` : تمام مسیج های ارسال شده کاربر را نگه داری میکند و از جنس `List<Message>` است.
۳. `receivedMessages` : تمام مسیج های دریافت شده کاربر را نگه داری میکند و از جنس `List<Message>` است.

تمام پراپرتی ها سطح دسترسی `private` را دارند.

کانستراکتور

این کلاس تنها یک متود سازنده دارد که `name` را ورودی میگیرد و آنرا مقداردهی میکند.

متود ها

۱. `send` : مسیجی را با پیام رسان دلخواه به گیرنده دلخواه میفرستد.
۲. `receive` : کاربر مسیج را دریافت میکند.
۳. `readMessage` : کاربر مسیج ورودی گرفته شده را میخواند. (صرفا وضعیت پیام را آپدیت کنید)

توجه : فقط قسمت های `//TODO` ۳ متود بالا را کامل کنید.

۴. `deleteSentMessage` : با استفاده از این متود کاربرمیتواند مسیج ورودی گرفته شده را از لیست مسیج های ارسالی اش پاک کند.
۵. `deleteReceivedMessage` : با استفاده از این متود کاربرمیتواند مسیج ورودی گرفته شده را از لیست مسیج های دریافتی اش پاک کند.

در ۳ متود بالا نیازی به چک کردن وجود داشتن پیام در لیست پیام های کاربر نیست، تضمین میشود که پیام در لیست وجود دارد.

▼ مثال

برای مثال با اجرای main زیر:

```java
public static void main(String[] args) {
    Messenger telegram = new Telegram();
        User aria = new User("Aria");
        User arman = new User("Arman");

        aria.send("Hello Arman!", arman, telegram);
    }
```

خروجی به صورت زیر است:

```
Aria: Sending "Hello Arman!" to Arman via Telegram...
Message sent via Telegram: Hello Arman!
Arman: Received "Hello Arman!"
```

کلاس MessageStatus

از جنس ENUM است که وضعیت های پیام را نگه میدارد.

```java
 public enum MessageStatus {
    SENT,
    DELIVERED,
    READ
}
```

نکات

- شما اجازه‌ی اضافه کردن پراپرتی دیگری غیر از پراپرتی‌های خواسته‌ی سوال ندارید.
- گتر و ستر های مورد نیاز را پیاده سازی کنید.

▼ نحوه نام گذاری متود های گتر و ستر

نامگذاری باید به شکل Camel case باشد. به طور مثال برای نامگذاری متد *setter* و *getter* فیلدی به نام name به ترتیب باید به صورت setName و getName نامگذاری شود.

▼ نحوه استفاده از کلاس LocalDateTime

برای مطالعه متدها و نحوه‌ی استفاده از این کلاس می‌توانید به اینجا مراجعه کنید..

آنچه باید آپلود کنید

ساختار فایل `zip` ارسالی باید به صورت زیر باشد:

```
<zip_file_name.zip>
    ├── Messenger.java
    ├── Telegram.java
    ├── Whatsapp.java
    ├── Instagram.java
    ├── Message.java
    ├── User.java
    └── MessageStatus.java
```

مدیریت اتوبوس

- سطح: متوسط
- طراح: نیلا چناری

شما قرار است یک سیستم مدیریت برای اتوبوس های شهر بنویسید.

ابتدا پروژه اولیه را این لینک دانلود کنید.

- نکات کلی:
  - به اسم متد ها و پراپرتی ها دقت کرده و عین نمونه ها بنویسید.
  - برای تمام پراپرتی ها **گتر و ستر های مورد نیاز** را به صورت کمل کیس بگذارید.
  - به کانستراکتور ها و مقادیری که باید در ورودیشان گرفته شود یا همانجا مقداردهی شوند نیز توجه کنید.

اینترفیس `Vehicle`

اینترفیس ماشین که برخی متد های مشترک بین تمام ماشین های موجود را دارد.

```
interface Vehicle:

String start();
String stop();

int getCapacity();
String getEnergySource();
String getLicensePlate();
String getModel();
int getYear()
```

- توضیحات (به ترتیب):
  - نمایش رشته ای از وضعیت ماشین در حال شروع به حرکت.
  - نمایش رشته ای از وضعیت ماشین در حال توقف.
  - نشان دادن ظرفیت.

- نشان دادن نوع انرژی ماشین (برق یا بنزین یا ...).
- نشان دهنده شماره پلاک
- نشان دهنده مدل
- نشان دهنده سال خریداری شده

کلاس انتزاعی Bus

همانطور که میدانیم یک نوع ماشین اتوبوس است. که باید به صورت یک انتزاع پیاده سازی شود.

```
class Bus:

String licensePlate
String model
int year
int capacity
int numberOfDoors
String energySource

String replenishEnergy()
```

- توضیحات (به ترتیب):
  - شماره پلاک
  - مدل ماشین
  - سال خریداری شدن
  - ظرفیت
  - تعداد در ها
  - نوع سوخت (غیر قابل تغییر و حتما موقع ساخت شی ست شود)
  - متد replenishEnergy() برای شارژ کردن یا پر کردن باک است که در این کلاس پیاده سازی ای ندارد.
- متد های start و stop را به این صورت پیاده سازی میکند.

```
start()
stop()
```

با فراخوانی این متد باید رشته .the bus is trying to start برگردانده شود. با فراخوانی این متد باید رشته .the bus is trying to stop برگردانده شود.

- همچنین **همه متدهایی** از اینترفیس که به آن مربوط میشود و برای تمام ساب-کلاس هایش یکسان است را پیاده سازی میکند.

دو نوع اتوبوس سوختی و برقی داریم. که هر نوع از اتوبوس هم چند تایپ ماشین دارند. پس این دو کلاس برقی و سوختی abstract هستند و vehicle را پیاده سازی میکنند.

کلاس انتزاعی ElectricBus

```
class ElectricBus:

boolean hasAutopilotMode
int batteryCapacity
String chargingStationLocation
boolean autoPilotMode

boolean enableAutopilotMode()
boolean disableAutopilotMode()
String recharge()
```

- توضیحات (به ترتیب):
  - ایا قابلیت خودران دارد یا خیر
  - ظرفیت باطری
  - محل شارژ
  - آیا در حال حال حاضر ماشین در حالت خودران قرار دارد یا خیر
  - روشن کردن اتوپایلت (طبیعتا در صورت داشتن قابلیت). در صورت موفقیت آمیز بودن عملیات true و در غیر این صورت false برگردانید.
  - خاموش کردن اتوپایلت (طبیعتا در صورت داشتن قابلیت). در صورت موفقیت آمیز بودن عملیات true و در غیر این صورت false برگردانید.
  - (اگر اتوپایلت روشن بود و باز متد enable فراخوانی شد نیز باید true برگردانید، همچنین برای خاموش)

- اگر دستگاه در محل شارژ بود رشته bus is charging و در غیر این صورت bus is not charging را برگردانید.
- این متد در ایت کلاس نباید هیچ پیاده سازی داشته باشد. با فراخوانی متد recharge روی هر شی، در صورت قرار داشتن اتوبوس در محل شارژ رشته ای مانند نمونه برگردانده شود، و در غیر این صورت رشته bus is not charging.
- نمونه: #tFullClassName# is charging. (به تست کیس ها نگاه کنید)

```
replenishEnergy()
```

این کلاس متد بالا را پیاده سازی میکند و در آن شارژ ماشین را صدا میزند.

کلاس انتزاعی DieselBus

```
class DieselBus:

int fuelTankCapacity
String gasStationLocation
boolean hasTicketMachine;

String refuel()
```

- توضیحات (به ترتیب):
  - ظرفیت باک
  - محل پمپ بنزین
  - ایا دستگاه کارت بلیط دارد
    - این فیلد به صورت دیفالت مقدار دهی میشود و یوزر نباید بتواند به صورت دستی مقدارش را عوض کند.
    - اگر سال خرید برای آن مشخص شده باشد:
      - فقط اتوبوس های بعد از سال 2000 ای قابلیت را دارند.
    - اگر سال خرید برای آن مشخص نشده باشد به صورت دیفالت اتوبوس این قابلیت را ندارد.

- متد refuel در این کلاس نباید هیچ پیاده سازی داشته باشد. با فراخوانی متد refuel روی هر شی، در صورت قرار داشتن اتوبوس در پمپ بنزین باک را پر کرده و رشته ای برگردانید. توضیحات بیشتر در کلاس فرزند داده میشود.

```
replenishEnergy()
```

این کلاس متد بالا را پیاده سازی میکند و در آن پرکردن باک اتوبوس را صدا میزند.

کلاس CityElectricBus

```
class CityElectricBus:

String initialStop
String destinationStop
int numberOfStops
```

- توضیحات (به ترتیب): توجه: تمام این سه فیلد حتما باید در کانستراکتور مقدار دهی شوند
  - ایستگاه شروع
  - ایستگاه پایان
  - تعداد کل ایستگاه ها
    - باید این آپشن وجود داشته باشد که در صورت تمایل مقداری برای این فیلد وارد نشود و به صورت دیفالت 30 مقدار دهی شود.
- متدهای دیگر:

```
start()
stop()
```

- اتوبوس فقط وقتی میتواند شروع به حرکت کند که ایستگاه شروع برای آن مشخص شده باشد.
  - اگر بتواند شروع به حرکت کند، علاوه بر رشته تعریف شده در کلاس پدر، ایستگاه شروع را نیز برمیگرداند.
    - نمونه: the bus is trying to start. starting form stop1.
  - اگر نتواند شروع به حرکت کند رشته زیر را برمیگرداند:

    - city electric bus can not start without initial stop.
- اتوبوس فقط وقتی میتواند ایست کند که برای آن ایستگاه پایانی مشخص شده باشد.
    - اگر بتواند توقف کند، علاوه بر رشته تعریف شده در کلاس پدر، ایستگاه پایانی را نیز برمیگرداند.
        - نمونه: the bus is trying to stop. stopping at stop2.
    - اگر نتواند شروع به حرکت کند رشته زیر را برمیگرداند:
        - city electric bus can not stop without destination stop.

کلاس LuxuryElectricBus

```
class LuxuryElectricBus:

boolean hasMusicSystem
boolean hasWifi
hasRestaurant
boolean musicPlaying
boolean wifiAvailable
String[] foods

void createRestaurant(...)
boolean addFood(String foodName)

void playMusic()
void stopPlayingMusic()
void disableWifi()
void enableWifi()
```

- توضیحات (به ترتیب):
    - داشتن قابلیت پخش موزیک (تغییر ناپذیر و حتما ست شود)
    - داشتن قابلیت وایفای (تغییر ناپذیر و حتما ست شود)
    - داشتن رستوران (تغییر پذیر، اما حتما در کانستراکتور ست شود)
    - نشان دهنده در حال پخش بودن موزیک در حال حاضر
    - نشان دهنده روشن بودن وایفای در حال حاضر
    - لیست غذا ها
    - متد enableRestaurant():

- اگر اتوبوس رستوران دارد لازم نیست کاری انجام دهید.
- اگر رستوران ندارد این قابلیت را برای آن true کنید و لیست غذا برای آن بسازید:
- اگر کاربر بخواهد در ورودی متد تعداد غذا ها را وارد کند باید این آپشن را داشته باشد.
- اگر وارد نکرد تعداد غذا ها را ۱۰ بگیرید.

- متد addFood در صورت وجود داشتن رستوران و پر نبودن لیست غذا ها، آن را به foods اضافه میکند و true برمیگرداند.
- چهار متد آخر در صورت اینکه اتوبوس از قابلیت پشتیبانی کند، آن را روشن یا خاموش میکنند.

- نکات دیگر:
  - دسترسی به فیلد های wifiAailable , musicPlaying را طوری بگذارید که فقط از طریق ۴ متد مخصوصوششان بتوان آنها را کنترل کرد.
  - در کانستراکتور باید آپشن اینکه کاربر همراه با داشتن رستوران بخواهد تعداد غذا ها را مشخص کند نیز وجود داشته باشد. اما به صورت دیفالت اگر hasRestaurant false بود نباید آرایه food ساخته شود، و اگر true بود و مقداری برای آن وارد نشده بود آرایه ای به طول 10 ساخته شود.

```
start()
stop()
```

- اتوبوس موقع start علاوه رشته های پدر وضعیت خود را نیز نشان میدهد:
  - نمونه:
- اتوبوس موقع ایست موسیقی و وایفای خود را خاموش میکند و رشته luxury electric bus stopped. را برمیگرداند.

کلاس CityDieselBus

```
class CityDieselBus

int currentFuelLevel
```

- توضیحات (به ترتیب):

- سطح پری باک
  - (بین ۰ تا ظرفیت باک)
  - به صورت دیفالت ۰ باشد.
  - همچنین کاربر این آپشن را داشته باشد که موقع ساخت شی این مقدار را اعلام کند.

```
refuel()
refuel(int fuelAmount)
```

- این متد را از کلاس پدر به صورت زیر پیاده سازی میکند
  - اگر در پمپ بنزین نباشد:
    - city diesel bus is not refueling.
  - اگر در پمپ بنزین باشد باک را کامل پر میکند و رشته ای مانند نمونه برمیگرداند
    - city diesel bus is refueled with XXX liters of diesel.
    - XXX مقدار بنزین زده شده است
- همچنین یک آپشن به کاربر میدهد که مقدار بنزین را مشخص کند.
  - اگر پمپ بنزین نداشته باشد مانند توضیحات پیش رشته برمیگرداند.
  - اگر مقدار درخواستی بیشتر از ظرفیت باشد:
    - city diesel bus cannot be refueled more than XXX liters.
    - که XXX ظرفیت باک است
  - در غیر این صورت مانند توضیحات قبل مقدار بنزین زده شده در رشته برمیگرداند.

```
start()
stop()
```

- در استارت علاوه بر رشته پدر مقدر بنزین را نیز نشان میدهد.
  - نمونه: the bus is trying to start. and its gas level currently is XXX.

نحوه ساخت کانستراکتور ها: به ترتیب تعریف شده در سوال پراپرتی های لازم را در کانستراکتور ها قرار دهید. (اول پراپرتی های کلاس پدر و ...)

مثال

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

```
public class Main {
    public static void main(String[] args) {
        Bus bus = new CityDieselBus("Diesel", 100, 70);
        System.out.println(bus.start());
        bus.setLicensePlate("1234");
        System.out.println(bus.getLicensePlate());
        System.out.println(bus.replenishEnergy());
        ((DieselBus) bus).setGasStationLocation("gas station 1")
        System.out.println(((CityDieselBus) bus).refuel(20));
        System.out.println(bus.replenishEnergy());

        Vehicle cityElectricBus = new CityElectricBus("Urabn Ele
        System.out.println(((CityElectricBus)cityElectricBus).ge
        System.out.println(cityElectricBus.start());
        System.out.println(cityElectricBus.stop());
        System.out.println(((CityElectricBus) cityElectricBus).e
        ((ElectricBus)cityElectricBus).setChargingStation("charg
        System.out.println(((Bus) cityElectricBus).replenishEner
    }
}
```

خروجی به این شکل باشد.

```
the bus is trying to start. and its gas level currently is 70.
1234
city diesel bus is not refueling.
city diesel bus is refueled with 20 liters.
city diesel bus is refueled with 10 liters of diesel.
30
the bus is trying to start. starting from velenjak.
the bus is trying to stop. stopping at parkway.
false
city electric bus is charging.
```

آنچه که باید آپلود کنید:

باید یک فایل زیپ از کلاس های کامل شده آپلود کنید.

غریبه‌ها

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

جزئیات برنامه

ابتدا پروژه اولیه را این لینک دانلود کنید.

دکتر نفاریو با استفاده از تلپورت خود به دنیای واقعی سفر کرد. او همیشه به دنبال تجربه‌های تازه است. به همین دلیل او با استفاده از دستگاه مخصوص خود دو انسان عادی را به انسان‌هایی عجیب تبدیل کرد که عقاید متفاوتی دارند.

اینترفیس StrangePerson :

این اینترفیس دارای متدهای زیر می‌باشد:

```
1 int strangeCalculate(int number1,int number2);
2 String response(String input);
3 String belief(int number);
```

`strangeCalculate` : انسان‌های عجیب همیشه دوست دارند محاسباتی ویژه روی دو عدد داشته باشند.

`belief` : آنها عقیده مشخص خود را دارند اما یک چیز مشخص است، عقاید آنها بر اساس اعداد است.

`response` : انسان‌های عجیب پاسخ‌های معینی برای سوالات و صحبت‌های شما دارند.

کلاس Jack:

دکتر نفاریو دو دسته انسان عجیب تولید می‌کند. دسته اول که اسمشان Jack است و دسته دوم که اسمشان Kevin است.

توجه: این کلاس‌ها دارای فیلد نیستند.

متدها:

```
@Override
public int strangeCalculate(int number1, int number2;
```

انسان‌هایی که اسمشان `Jack` است همیشه دوست دارند ب.م.م دو عدد را محاسبه کنند. بنابراین آنها هر دو عددی که کنارهم ببینند را ب.م.م می‌گیرند و مقدار ب.م.م را برمی‌گردانند.

```
@Override
public String response(String input);
```

اگر شما بخواهید با یک نمونه `Jack` صحبت کنید، او فقط نگاه می‌کند که جمله شما دارای کلمه $hello$ است یا خیر فارغ از اینکه به چه‌شکلی نوشته شود ($HELLO$ یا $hello$ یا $HeLlo$ یا حالت‌های دیگر). اگر جمله شما دارای این کلمه باشد، در پاسخ به شما `hey` را برمی‌گرداند. در غیر اینصورت `goodbye` را برمی‌گرداند.

```
@Override
public String belief(int number);
```

عقاید یک نمونه از نوع `Jack` به این بستگی دارد که عددی که به آن توجه می‌کند اول است یا خیر. اگر اول باشد عقیده او این است: `seven is my lucky number` . در غیر اینصورت عقیده او این است: `nine is my lucky number`.

کلاس Kevin:

متدها:

```
@Override
public int strangeCalculate(int number1, int number2);
```

یک نمونه انسان از نوع `Kevin` عادت دارد بعد از دیدن دو عدد کنار هم، جمع ارقام آن دو عدد را حساب کند و حاصل نهایی را اعلام کند.

▼ مثال

به طور مثال اگر اعداد 107 و 51 باشند مقدار 14 را اعلام می‌کند.

```
@Override
public String response(String input);
```

یک نمونه انسان از نوع `Kevin` صرفا طول رشته ای که به او می‌دهید را نگاه می‌کند. اگر طول آن برابر ۹ یا بیشتر باشد در پاسخ به شما مقدار `perfect` را برمی‌گرداند. در غیر اینصورت مقدار `weird` را برمی‌گرداند.

```
@Override
public String belief(int number);
```

عقیده یک نمونه انسان از نوع `Kevin` به این است که آیا عدد توانی از ۲ است یا خیر. اگر بود عقیده او `two is my lucky number` است. در غیر اینصورت عقیده او `two is not my lucky number` است.

آنچه باید آپلود کنید

ساختار فایل `zip` ارسالی باید به صورت زیر باشد:

```
<zip_file_name.zip>
    ├── StrangePerson.java
    ├── Jack.java
    ├── Kevin.java
```