رفلکشن

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

پروفسور علی که ید طولایی در فیزیک کوآنتوم دارد، به تازگی با مفهوم clone آشنا شده است. او می‌داند که اگر دستگاهی اختراع کند که بتواند به ازای هر ورودی، عین همان را خروجی دهد، عملاً به تکنولوژی clone انسان دست پیدا کرده است. حال علی این کار خطیر را به شما سپرده است و می‌داند که شما حتماً از پس آن برمی‌آیید.

توضیحات پیاده‌سازی

در ابتدا این template را دریافت کنید. شما در این template باید متد clone در کلاس DeepCopy را کامل کنید. این متد یک object ورودی می‌گیرد و یک object همانند آن که کپی شده‌ی آن است را خروجی می‌دهد. توجه کنید که مقدار هر یک از متغیرهای آن باید عیناً کپی شوند همچنین هر متغیر ممکن است خود یک object دیگر باشد. ما تنها تضمین‌های زیر را داریم و به جز این موارد تضمین دیگری نداریم:

- هیچ کدام از objectها هیچ interfaceی را implement نمی‌کنند.
- هیچ کدام از متدها final نیستند.
- تمام کلاس‌ها حتماً یک constructor بدون پارامتر دارند.
- در کلاس همانند A امکان ندارد که یک متغیر از کلاس B باشد و در کلاس B هم یک متغیر از کلاس A باشد.

در نهایت در template داده شده، یک سری unit test وجود دارند که پاس شدن تمام آن‌ها لازم و کافی است.

تک رستوران دکور شده

در این سوال قصد داریم که کد یک رستوران را پیاده‌سازی کنیم. در این رستوران مقداری مواد اولیه داریم و می‌توانیم سفارش‌های مختلفی از طرف مشتریان بگیریم. دقت کنید که تنها یک رستوران داریم (به عبارت دیگر باید از دیزاین پترن مناسب استفاده کنید تا از ایجاد چند نمونه از این کلاس جلوگیری کند). هدف اصلی در این سوال بررسی سفارشات مشتریان و تحویل آنها در صورت توانایی آماده کردن آنها، و در غیر این صورت، رد کردن می‌باشد.

رستوران می‌تواند ساندویچ یا پیتزا سرو کند و پیتزاهای آن سایز کوچک، متوسط و بزرگ دارند. علاوه بر آن هر ساندویچ و پیتزا مواد خاصی در خود دارند. مواد موجود در این سوال به همراه قیمت آنها در جدول زیر آمده‌اند:

Ingredient	Price
Bacon	2.2
Basil	1.2
Bread	0.8
Cheese	1
Chicken	2.4
Corn	1
Dough	-
Egg	1.9
Fries	3.6
Garlic	3.6

Ingredient	Price
Hamburger	2.8
Jalapeno	4
Lettuce	1.8
Mushroom	1.6
Olive	1.6
Onion	3.5
Pepper	1.8
Pepperoni	3
Pickles	2.8
Salami	1.5
Sauce	1
Tomato	3.2
Tuna	2.8

برای آماده‌سازی یک پیتزا به یک خمیر (Dough) و برای آماده‌سازی ساندویچ به دو عدد نان (Bread) نیاز است. خمیر پیتزا به این دلیل قیمت ندارد که قیمت آن روی پیتزا محاسبه می‌شود و مشتری نمی‌تواند در سفارش خود درخواست خمیر اضافی کند. بقیه مواد می‌توانند توسط مشتری به عنوان محتوای اضافی به غذا اضافه شوند. همچنین هزینه پایه یک ساندویچ ۲ و یک پیتزای کوچک، متوسط و بزرگ به ترتیب ۲، ۴ و ۵ می‌باشد.

شکل سفارشات به فرمت زیر خواهد بود:

```
[[[Small|Medium|Large] Pizza] | Sandwich] [[and? also? with? extra?] [I
```

میان هر دو ماده اضافی می‌تواند هر کدام از حروف ربط نوشته شده در بالا به کار رود (ممکن است بیشتر از یک حرف ربط به کار رود اما حداقل یک حرف ربط استفاده می‌شود). پس از دریافت هر سفارش باید بررسی کنیم که مواد لازم برای آماده‌سازی سفارش در انبار رستوران موجود است یا خیر. اگر بود، سفارش آماده می‌شود و در غیر این صورت سفارش رد می‌شود. پیام‌های مربوطه به کاربر بابت انجام و رد سفارش مطابق زیر است:

```
Order Completed!
```

```
Order Dismissed!
```

در نهایت، باید درآمد حاصل از انجام همه سفارشات رستوران را پرینت کنید.

دقت کنید که برای پیاده‌سازی مواد مختلف باید از دیزاین پترن **دکوریتور** استفاده کنید.

ورودی

در خط اول ورودی ۲۴ عدد دریافت خواهید کرد. عدد اول نشان‌دهنده تعداد سفارشات و ۲۳ عدد بعدی، تعداد هرکدام از مواد موجود به ترتیب جدول داده شده را نشان می‌دهد.

در N خط بعدی در هر خط یک سفارش داده می‌شود.

خروجی

در خروجی باید پاسخ مربوط به هر سفارش را پرینت کنید. در خط آخر هم باید درآمد کلی را پرینت کنید:

```
The final profit is: [profit with one decimal point]
```

ورودی نمونه ۱

```
3 0 0 4 0 0 0 1 3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Sandwich with Egg and also Egg and extra Fries
Sandwich with Fries and also with extra Mushroom
Large Pizza with extra Fries and Egg
```

خروجی نمونه ۱

```
Order Completed!
Order Dismissed!
Order Completed!
The final profit is: 19.9
```

چیزی که باید آپلود بکنید

فایل ارسالی .zip شما باید ویژگی های زیر را داشته باشد:

- برای ارسال پاسخ های خود، تمامِ محتویاتِ پوشه‌ی src خود را در یک فایل زیپ قرار دهید و آن فایل زیپ را ارسال کنید. برای مثال فایلِ زیپِ شما باید ساختاری مانند زیر داشته باشد:

```
├── Class1.java
├── Class2.java
├── Main.java
├── Package1
|   ├── Class3.java
|   └── Class4.java
└── Package2
    ├── Class5.java
    └── Class6.java
```

- همان گونه که در دیاگرام بالا نشان داده شده است، entry point شما باید نام Main.java را داشته باشد یعنی این فایل درون هیچ پوشه (package) دیگری نباشد.

حالت‌های مختلف نتیجه‌ی جاج به شرح زیر است:

```
❌ compileTest -> Your code does not compile (check imports?)
❌ judgeProject[i] | ❌ runtimeCheck[i] -> Your code throws runtime exc
❌ judgeProject[i] | ✅ runtimeCheck[i] -> Your code produces wrong ans
✅ judgeProject[i] | ✅ runtimeCheck[i] -> GG
```

دقت کنید فقط زمانی نمره‌ی یک تست را می‌گیرید که تست `judgeProject` PASS شود.

در صورتی که کد شما کلا کامپایل نمی‌شود به کمک دکمه‌ی اجرای تست نمونه می‌توانید دلیل آن را مشاهده کنید.

دیزاین

- محدودیت زمان: 4 ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

فرض کنید شما یک شهر هوشمند دارید، روزانه در این شهر تعدادی ربات تولید می‌شوند. این ربات‌ها از سه نوع هستند و هر یک، یک مسئولیت خاص بر عهده دارند.

ربات نوع اول ربات نگهبان است که کار او محافظت از شهر است. هر ربات نگهبان با دستور زیر تولید می‌شود:

```
create security robot -id -power -live
```

که ورودی `power` قدرت آن ربات و ورودی `live` تعداد جان‌های آن ربات است.

ربات نوع دوم ربات حامل است که کار او حمل‌ونقل در شهر است. هر ربات حامل با دستور زیر تولید می‌شود:

```
create delivery robot -id -vehicle
```

که ورودی `vehicle` وسیله نقلیه آن ربات است.

ربات نوع سوم ربات رفتگر است که وظیفه او تمیزی شهر است. هر ربات رفتگر با دستور زیر تولید می‌شود:

```
create cleaning robot -id -num_task -areas
```

که ورودی `num_task` تعداد `task` هایی است که باید انجام دهد و `areas` شماره محدوده‌هایی است که مجاز است تمیز کند.

شما برای این‌که ربات‌های خود را مدیریت کنید از دستورات زیر استفاده می‌کنید:

```
deliver robot -id
```

با این دستور ربات با `id` گفته شده یک بسته را می‌فرستد و در خروجی پیغام زیر چاپ می‌شود:

```
delivery robot id sent the pocket by -vehicle
```

که `vehicle` وسیله نقلیه ربات است.

اگر `id` گفته شده در بین ربات‌های حامل وجود نداشت، می‌بایست خطای زیر را خروجی دهید:

```
Invalid robot id
```

```
clean robot -id -area
```

با این دستور در صورتی که شماره `area` در لیست شماره محدوده‌های مجاز ربات با `id` مشخص شده باشد، این محدوده را تمیز می‌کند.

توجه کنید که اگر تعداد تسک‌های این ربات صفر باشد، او بازنشسته شده است و خطای زیر چاپ می‌شود:

```
this robot is retired
```

اگر محدوده داده شده در محدوده‌های مجاز ربات نبود، خطای زیر چاپ می‌شود:

```
invalid area
```

اگر id گفته شده در بین ربات‌های رفتگر وجود نداشت، خطای زیر را چاپ کنید:

```
Invalid robot id
```

بعد از اینکه این ربات تمیزکاری را انجام داد، یکی از تعداد تسک‌هایی که باید انجام دهد کم می‌شود و پیغام زیر چاپ می‌شود:

```
cleaning robot -id cleaned the area
```

که id شماره id ربات است.

```
perform task robot -id
```

این دستور باعث می‌شود ربات نگهبان با id داده شده تسک اصلی خود که monitoring است را انجام دهد و می‌بایست پیغام زیر چاپ شود:

```
security robot is monitoring
```

اگر id گفته شده در بین ربات‌های نگهبان وجود نداشت، خطای زیر را چاپ کنید:

```
Invalid robot id
```

```
attack robot -id -enemyPower
```

این دستور باعث می‌شود ربات نگهبان با id داده شده عملیات خود را آغاز کند. قدرت دشمن در ورودی داده شده است و همان enemyPower است. اگر power ربات نگهبان بیشتر یا مساوی رنک دشمن باشد، پیروز خواهد شد. در غیر این صورت، یکی از تعداد جان‌های این ربات کم خواهد شد. اگر جان این ربات به صفر برسد، می‌میرد و از لیست ربات‌های موجود حذف می‌شود.

اگر حمله پیروز شود، می‌بایست پیغام زیر را چاپ کنید:

```
attack was successful
```

و اگر پیروز نشود، می‌بایست پیغام زیر را چاپ کنید:

```
attack was unsuccessful
```

اگر `id` گفته شده در بین ربات‌های نگهبان وجود نداشت، خطای زیر را چاپ کنید:

```
Invalid robot id
```

```
get all robots
```

با این دستور لیست تمام ربات‌ها با فرمت زیر به ترتیب id خروجی داده می‌شود:

```
robot -id -kind
```

که `kind` نوع ربات از بین سه نوع گفته شده است.

در صورتی که دستوری ناشناخته بود پیغام زیر را چاپ کنید:

```
Unknown command
```

با دستور `end` برنامه خاتمه می‌یابد.

دقت کنید در پیاده‌سازی این تمرین باید از Factory Method Design Pattern استفاده کنید و به پاسخ‌های خارج از این Design Pattern نمره‌ای تعلق نمی‌گیرد. شما باید حتما سه کلاس `CleaningRobot` , `DeliveryRobot` , `SecurityRobot` و همچنین سه کلاس `factory` برای ساخت هر یک از این ربات‌ها داشته باشید.

مثال

ورودی نمونه ۱

```
create security robot 22 70 3
create delivery robot 23 car
create cleaning robot 24 4 5 6 7
deliver robot 23
clean robot 24 5
attack robot 22 65
perform task robot 22
attack robot 22 75
deliver robot 23
clean robot 24 6
clean robot 24 7
attack robot 22 35
attack robot 22 45
clean robot 24 8
clean robot 24 7
clean robot 24 6
attack robot 22 75
attack robot 22 75
get all robots
end
```

خروجی نمونه ۱

```
delivery robot 23 sent the pocket by car
cleaning robot 24 cleaned the area
attack was successful
security robot is monitoring
attack was unsuccessful
delivery robot 23 sent the pocket by car
cleaning robot 24 cleaned the area
cleaning robot 24 cleaned the area
attack was successful
attack was successful
invalid area
cleaning robot 24 cleaned the area
this robot is retired
attack was unsuccessful
attack was unsuccessful
```

```
robot 23 delivery
robot 24 cleaning
```