Reflection

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: 50 مگابایت

مامور انگلیش پس از ناکامی‌هایی که در ماموریت‌های اخیر خود داشته است به بخش فنی منتقل شده است. در پروژه‌ای که برای هک دستگاه‌های طرف مقابل طراحی شده است نیاز به کلاسی است که قدرت نفوذ به کلاس‌های دیگر داشته باشد. مامور انگلیش مسئول نوشتن بخش اصلی این پروژه شده است. کلاسی که او باید آنرا بنویسد بعدا در کنار دیگر فایلها قرار خواهد گرفت تا پروژه تکمیل شود.

او برای انجام این کار باید کلاس Agent را بنویسد اما متوجه حرکت‌های شرورانه‌ای شده و می‌خواهد با آنها مقابله کند و از شما درخواست کرده تا این کلاس را برای او بفرستید. این کلاس مامور نفوذ به اشیاء و کلاس‌هاییست که به او اطلاع داده می‌شود. این کلاس نیازی به متد main ندارد و تنها باید متدهای خواسته شده را پیاده‌سازی کند. این مامور (همان کلاس Agent) باید قابلیت‌های زیر را داشته باشد:

- پیدا کردن نام متدهای کلاسِ یک شیء که به مامور داده می‌شود
- دسترسی به فیلد اعلام شده
- صدا زدن یک متد با استفاده از نام و آرگومان‌های داده شده به مامور

- ساخت یک شیء تازه
- تخلیه اطلاعاتی یک شیء
- ساخت یک کپی از شیء داده شده به مامور

> شدیدا پیشنهاد می‌شود تا از جاوا ۸ استفاده کنید :)

در ادامه امضای متدهایی که باید در این کلاس وجود داشته باشد و کاری که انجام می‌دهند شرح داده شده است:

پیدا کردن نام متدهای کلاسِ یک شیء 🕵🕵

```
1 public List<String> getMethodNames(Object object)
```

این متد نام متدهایی که مربوط به کلاسِ شیء object است را به صورت یک لیست برمی‌گرداند.

دسترسی به فیلد 🤝

```
1 public Object getFieldContent(Object object, String fieldName) throws
```

این متد باید محتویات فیلدی با نام fieldName را در شیء object به ما برگرداند.

- اگر چنین فیلدی در کلاس شیء مورد نظر وجود نداشت یک java.lang.NoSuchFieldException را throw می‌کند.

```
1 public void setFieldContent(Object object, String fieldName, Object co
```

این متد باید محتویات فیلدی با نام fieldName را در شیء object عوض کند به شرطی که نوع content و نوع فیلد مورد نظر دقیقا یکی باشد در غیر اینصورت هیچ کاری نباید انجام شود.

- اگر چنین فیلدی در کلاس شیء مورد نظر وجود نداشت یک java.lang.NoSuchFieldException را throw می‌کند.

صدا زدن یک متد ☎

```
public Object call(Object object, String methodName, Object[] paramete
```

این متد، متدی با نام methodName شیء object را با پارامترهای parameters صدا می‌زند. پارامترها به ترتیب آرگومان‌های تابع در کد آن کلاس هستند. خروجی این متد، خروجی متد صدازده‌شده است.

- اگر متدی با این نام وجود نداشت و یا اینکه نوع پارامترها با پارامترهای آن متد همخوانی نداشت یک java.lang.NoSuchMethodException را throw می‌کند.

ساخت یک شیء تازه 👶

```
public Object createANewObject(String fullClassName, Object[] initials
```

این متد نام یک کلاس را به صورت کامل (همراه با نام پکیج، برای مثال src.model.User) را می‌گیرد. initials یک آرایه از اشیاء است که باید از آنها برای ساخت شیء تازه در کانستراکتور آن کلاس استفاده کنید.

- اگر کلاسی با این نام وجود نداشت یک java.lang.ClassNotFoundException را throw می‌کند.
- اگر در کلاس کانستراکتوری موجود نباشد که با initials همخوانی داشته باشد یک java.lang.NoSuchMethodException را throw می‌کند.

تخلیه اطلاعاتی یک شیء 👮‍♂️👮‍♀️

```
public String debrief(Object object)
```

مامور با استفاده از این متد می‌تواند به اطلاعات حیاتی کلاس مربوط به object دست پیدا کند. خروجی این دستور یک رشته است که باید به صورت زیر باشد:

```
Name: <Class Name>
Package: <Package Name>
No. of Constructors: <#Constructors of Class>
===
Fields:
<?public|private|protected> <?static> <?final> <Type of Field> <Name of Fie
...
```

```
(<Count of Fields> fields)
===
Methods:
<Return Type> <Method Name>(<Parameter Types>...)
(<Count of Methods> methods)
```

برای مثال نوشته زیر می‌تواند خروجی این متد باشد:

```
Name: TestClass
Package: this.is.a.package
No. of Constructors: 5
===
Fields:
public static Integer field1
private int field2
double field3
(3 fields)
===
Methods:
Integer doSomthing(Object, String, Integer, TestClass)
void doSomthingElse()
(2 methods)
```

توجه کنید که ترتیب متدها و فیلدها (با هر سطح دسترسی) باید ترتیب الفبایی باشد. همینطور تعداد تمام کانستراکتورها با هر سطح دسترسی آورده شود. اگر برای مثال کلاسی هیچ متدی نداشت نتیجه قسمت متدها به صورت زیر خواهد بود:

```
Methods:
(0 methods)
```

کپی کردن یک شیء

```
1 | public Object clone(Object toClone) throws Exception
```

این متد باید یک کپی از شیء toClone که به آن داده می‌شود بسازد. تضمین می‌شود که toClone (و بقیه کلاس‌های احتمالی که در این فرایند ممکن است به آن برخورد کنید :)) متعلق به کلاسی است که کانستراکتوری بدون پارامتر دارد و یا اگر پارامترهای آن هیچ‌مقدار (یا صفر برای پارامترهای Primitive)

باشد مشکلی رخ نخواهد داد. همینطور تضمین می‌شود که اشیائی که به این کلاس پاس داده می‌شوند به نسبت ساده هستند.

بازگشت پیام‌رسان

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

پس از آن‌که مدیر بخش فنی از غیبت مامور انگلیش آگاه شد، جلسه‌ای اضطراری تشکیل داد و او را از آن بخش هم بیرون کرد. وقتی مامور انگلیش این موضوع را متوجه شد، تصمیم گرفت تا با نشان دادن شگردی در برنامه‌نویسی، مدیر را تحت تاثیر قرار دهد و او را متقاعد کند تا با بازگشتش موافقت کند.

او کمی با خود فکر کرد و به یاد آورد که یک بار به طور کاملا اتفاقی شنیده بود که مدیر بخش فنی علاقه‌ی زیادی به ساده‌سازی و خوانایی کد دارد و annotation ویژگی مورد علاقه‌ی او در جاواست! به همین دلیل مامور انگلیش از شما می‌خواهد پیام‌رسانی که برای او نوشتید را به گونه‌ای تغییر دهید که تعریف تگ‌ها در کد با استفاده از annotationها انجام شود و دیگر آن بخش صرفا با ifهای متعدد پیاده‌سازی نشده باشد. مدیر بخش فنی حتما مجذوب این ویژگی خواهد شد!

در این سوال شما باید annotationهای خواسته شده را به شکل زیر (یا به صورتی مشابه) در نظر بگیرید و parser مربوط به آن‌ها را طراحی و پیاده‌سازی کنید.

انوتیشن Instruction

می‌توانیم دستورات برنامه را با استفاده از این انوتیشن تعریف کنیم.

```
@Instruction(name = "<name>", description = "<description>")
InstructionHandler instructionHandler;
```

انوتیشن Label

این انوتیشن می‌تواند تگ‌هایی که صرفا بود و نبودشان مهم است و مقدار نمی‌گیرند (مثل `login`) را تعریف کند.

```
@Label(name = "<name>", description = "<description>")
Boolean labelIsPresent;
```

انوتیشن Attribute

تگ‌هایی که مقدار می‌گیرند (مثل `port`) را با این انوتیشن تعریف می‌کنیم.

```
@Attribute(name = "<name>", description = "<description>")
String attributeName;
```

دقت کنید که بخش name صرفا حاوی نام خواهد بود و پیشوند `--` در تگ‌ها خود به خود در parser در نظر گرفته می‌شود.

هنگامی که کاربر دستوری را وارد کرد و تگ‌های مربوط به آن به نحوی ناقص یا نادرست بودند، برنامه باید لیست تگ‌های مربوط به آن دستور را به صورت زیر نمایش دهد.

```
<instruction name> (help):
<name> : <description>
<name> : <description>
...
```

نکته ۱: دقت کنید که تغییرات برنامه صرفا در پیاده‌سازی اعمال می‌شود و نمود خارجی برنامه و آنچه کاربر می‌بیند همچنان مانند قبل خواهد بود. تنها تفاوت، اضافه شدن راهنمایی برای تگ‌ها در صورت وارد کردن تگ نادرست است.

نکته ۲: جزئیات طراحی و تغییراتی که اعمال می‌کنید بر عهده‌ی شماست و تا زمانی که موارد گفته شده در بالا رعایت شوند و کارکرد لازم پیاده‌سازی شود، دست شما در نحوه‌ی طراحی و پیاده‌سازی parser و انوتیشن‌ها باز است.

نکته ۳: یک نمونه از به کار بردن انوتیشن برای دریافت دستورات CLI (که مشابه خواست این سوال است) در اینجا قابل مشاهده است. برای به دست آوردن دید بهتر و آشنایی بیشتر با این موضوع، مطالعه و درک آن شدیدا توصیه می‌شود.