باغ وحش کارتونی

- سطح سوال: ساده

ریک و مورتی در یکی از آزمایش‌هایشان، فضا و زمان را به صورت عجیبی در هم پیچیدند. آن‌ها در این حادثه، موفق به اضافه کردن شخصیت‌های انیمیشن‌های محبوب ما به دنیای واقعی شدند. مشکل اینجاست که به علت پیچیدگی فضا و زمان (و احتمالا ابعاد دیگر)، تمام موجوداتی که به دنیای ما اضافه شدند، اصیل نیستند.

بعد از مهار کردن شرایط، همه‌ی شخصیت‌ها را به باغ وحشی هدایت کردیم تا بتوانیم شخصیت‌های اصیل و شخصیت‌های دیگر را از هم جدا کنیم و برای برگرداندن آن‌ها به دنیای خودشان، تصمیمات لازم را بگیریم. در حال حاضر موجودات حاضر در باغ وحش به ۴ دسته تقسیم می‌شوند:

۱. شخصیت‌های اصیل، که دقیقا همان شخصیت‌های کارتونی مجبوب ما هستند.
۲. شخصیت‌های تقلبی، که خود را به جای شخصیت‌های اصیل جا زده‌اند اما در واقع موجوداتی خبیث با نقشه‌های شیطانی‌اند.
۳. شخصیت‌های گم‌شده (در ذهن خود)، شخصیت‌های اصیلی بوده‌اند که در این حادثه، در ذهن خود گم شده‌اند و باید به آن‌ها شخصیت اصلی خود را یادآوری کنیم.
۴. دیگر موجودات، از جمله حیوانات عادی‌ای که قبلا در باغ وحش زندگی می‌کردند.

نحوه‌ی بررسی موجودات به این صورت است که ابتدا از هر موجود، نام او را می‌پرسیم و در لیستی قرار می‌دهیم. سپس به صورت تصادفی نام این موجودات را در بلندگوی باغ وحش اعلام می‌کنیم و جوابی که از آن موجود دریافت می‌کنیم را یادداشت می‌نماییم. نکته اینجاست که شخصیت‌های اصیل، پاسخ‌های منحصر به فرد دارند که از روی این پاسخ، می‌توان آن‌ها را شناسایی کرد.

شخصیت‌های اصیل نام درست و پاسخ منحصر به فرد خود را دارند. شخصیت‌های تقلبی خود را با نام یکی از شخصیت‌های اصیل معرفی می‌کنند اما پاسخ آن‌ها، اشتباه خواهد بود. شخصیت‌های گم‌شده پاسخ درستی خواهند داد، ولی نام خود را فراموش کرده و فکر می‌کنند کس دیگری هستند.

علاوه بر این اطلاعات، با بررسی (تماشای) انیمیشن‌های شخصیت‌های اصیل، به پاسخ منحصر به فرد آن‌ها پی برده‌ایم و آن‌ها را با شما در این لینک به اشتراک می‌گذاریم.

شما به عنوان طرفدار این شخصیت‌های محبوب، وظیفه‌ی اتوماتیک کردن پروسه‌ی دسته‌بندی موجودات باغ وحش را با استفاده از زبان جاوا و ویژگی‌های شیءگرایی آن دارید. فایل Zoo.java برای شما آماده شده و انواع موجودات حاضر در باغ وحش دارای کلاسی مختص خود هستند. در ورودی، ابتدا تعداد موجوداتی که اطلاعات آن‌ها را به شما خواهیم داد، اعلام میکنیم. برای ساده شدن کار شما، این مقدار حداکثر ۱۰ خواهد بود.

در ادامه، در هر ۲ خط، ابتدا نام موجود و سپس پاسخ آن را در خطوط جداگانه اعلام خواهیم کرد. شما در خروجی باید تعداد موجودات حاضر در هر یک از ۴ دسته‌ی اعلام شده را بیان کنید. برای روشن‌تر شدن نحوه‌ی ورودی دادن و خروجی گرفتن، به مثال زیر دقت کنید:

ورودی:

```
4
Baby Shark
Baaaby shark dodo, dodo dodo
Baby Shark
MaaaMaaa shark dodo, dodo dodo
Lost Shark
Baaaby shark dodo, dodo dodo
Shark
*Shark sounds*
```

خروجی:

```
Number of real characters: 1
Number of fake characters: 1
Number of lost characters: 1
Others: 1
```

توضیح:

ابتدا عدد ۴ وارد شده، که یعنی اطلاعات ۴ موجود در ادامه خواهد آمد. سپس در ۸ خط، ابتدا نام و سپس جواب هر موجود آمده. اولین موجود شخصیت اصیل Baby Shark است که نام آن کاملا درست بوده و پاسخش نیز همان پاسخ منحصر به فرد خودش است. پس این موجود را به عنوان یک کارکتر واقعی انیمیشنی طبقه‌بندی خواهیم کرد.

در خطوط ۴ و ۵، نام Baby Shark به درستی بیان شده اما این موجود جواب اشتباهی داده و در واقع از جواب مادر این شخصیت استفاده کرده است. در هر صورت پاسخ او اشتباه بوده و این موجود به عنوان یک شخصیت تقلبی شناسایی می‌شود.

خط ۶ و خط ۷ نمایان‌گر یک شخصیت گم‌شده هستند. این موجود پاسخ منحصر به فرد Baby Shark را داده و همین برای ما کافیست تا مطمئن شویم شخصیتی تقلبی نیست. اما نام خود را فراموش کرده و خودش را به عنوان Lost Shark معرفی کرده است. پس این موجود در ذهن خود گم شده و نیاز به کمک دارد.

۲ خط آخر نیز نام و پاسخ یک کوسه‌ی عادی اعلام شده که در دسته‌بندی others قرار می‌گیرد.

توجه: در این سوال تنها مجاز به استفاده از یک آرایه از جنس `Creature` هستید و آرایه‌ی دیگری مجاز نیست.

جست و جو درمورد Anonymous Class در جاوا توصیه می‌شود. این لینک نیز مفید خواهد بود.

در فایل `Zoo.java` چیزی نباید تغییر کند، تنها یک فایل در کنار آن ایجاد کنید و از کلاس‌های موجود در این فایل استفاده کنید. در نهایت فایل اصلی خود را آپلود کنید. جاج این سوال به دلیل استفاده از junit و بررسی ورودی و خروجی کد شما، مقداری بیشتر زمان می‌برد که جای نگرانی نیست :)

نام فایل آپلودی و کلاس اصلی شما باید Main باشد.

چندریختی؟

- سطح سوال: متوسط

سپهر که مبحث چندریختی یا polymorphism را در جاوا به خوبی درک نکرده بود، بعد از مدتی جستجو به دوستش نیما مراجعه می‌کند تا او این مشکل را برایش حل کند. نیما به جای توضیح مستقیم، برنامه‌ی ناقصی به سپهر می‌دهد تا با کامل کردنش، از نکات چندریختی آگاه شود و استفاده‌ی آن‌ها را ببیند.

مسئله اینجاست که سپهر هنوز نتوانسته این برنامه را طبق خواسته‌ی نیما کامل کند، در نتیجه او از شما می‌خواهد که در این کار به او کمک کنید.

این، برنامه‌ی ناقصی است که نیما به سپهر داده:

```
abstract class AbstractClass{

    final void func(){

        if(/* TODO */){

            System.out.print("not ");

        }

        System.out.println("func here");

    }

    abstract void func1(int a);

    abstract void func2();

    abstract AbstractClass func3(int a);

}

class Parent extends AbstractClass{
```

```
24
25      int a=3;
26
27      // TODO
28
29  }
30
31  final class Child extends Parent{
32
33      // TODO
34
35  }
36
37  public class Main {
38
39      public static void main(String[] args) {
40
41          int a = 10;
42
43          // TODO
44
45          reference.func();
46
47          reference.func1(a);
48
49          reference.func2();
50
51          // TODO
52
53          reference.func();
54
55          reference.func1(a);
56
57          reference.func2();
58
59          // TODO
60
61          reference.func4();
62
63          reference.func3(10).func4();
64
65      }
66
```

```
67 |
    }
```

طبق گفته‌های نیما، تابع `func1` در کلاس `Parent` نصف عدد ورودیش را چاپ می‌کند، اما در کلاس `Child` چهار برابر آن را.

همچنین تابع `func2` در کلاس `Parent` باید عبارت `did nothing` را چاپ کند، اما در کلاس `Child` باید تابع `func` را صدا بزند. نکته‌ی مهم این است که در این کلاس، نهایتا باید عبارت `func here` چاپ شود.

تابع `func3` نیز در کلاس `Parent` یک نمونه (instance) از کلاس `Child` با مقدار `a` برابر با ورودی خودش بسازد و آن را برگرداند. این تابع در کلاس `Child` دست نخورده خواهد ماند.

در نهایت تابع `func4` عبارت `doing the thing` را به تعداد `a` بار تکرار خواهد کرد. هر عبارت با عبارت قبلی یک فاصله دارد و همچنین در انتها نیز باید به خط بعد بروید. توجه کنید که آخرین عبارت بعد از خود فاصله‌ای ندارد و صرفا به خط بعد خواهد رفت. نیما روی این جزئیات حساس است و اگر این مورد رعایت نشود، از سپهر ایراد خواهد گرفت. به خروجی مد نظر نیما توجه کنید.

این تابع در کلاس `Child` مقدار `a` را به صورت `the a is: 3` (با فرض برابر بودن a با ۳) چاپ میکند.

نیما همچنین گفته که بعضی نکات را عمداً بیان نکرده تا سپهر خودش به آن‌ها پی ببرد. با کامل کردن تابع main، خروجی مورد نظرش اینچنین خواهد بود:

```
func here
5
did nothing
not func here
40
func here
doing the thing doing the thing doing the thing
the a is: 10
```

چیزی که باید به سپهر تحویل دهید

کد تکمیل‌شده‌ایست که نیما به او داده بود. توجه کنید که فقط باید در قسمت‌هایی که با کامنت `TODO` مشخص شده تغییر ایجاد بکنید، وگرنه نیما کد را از سپهر قبول نخواهد کرد. ممکن است برای بعضی از این کامنت‌ها، نیاز به نوشتن بیشتر از یک خط کد شود.

سازگارسازی

- سطح: ساده تا متوسط
- مباحث تحت پوشش: اینترفیس و برخی متدهای `Integer`

نیما از کودکی دوست داشت همه چیز سر جای خود و در مرتب‌ترین حالت ممکن باشند. اکنون که با برنامه‌نویسی آشنا شده است نیز دوست دارد اعداد را مرتب کند. او برنامه‌ای نوشته است که با استفاده از آن بتوان اعداد را بر اساس موارد مختلفی مرتب کرد. در حال حاضر، او دو روش برای مرتب‌سازی اعداد پیاده‌سازی کرده است. یکی از آن‌ها اعداد را به‌ترتیب غیرنزولی و دیگری، اعداد را به‌ترتیب غیرصعودی مرتب می‌کند. محمدرضا که خفن الگوریتمی است، اخیراً یک برنامه نوشته که اعداد را به‌ترتیب غیرنزولیِ تعداد بیت‌های `1` معادلِ باینری اعداد مرتب می‌کند. نیما می‌خواهد از برنامه‌ی محمدرضا در برنامه‌ی خود استفاده کند، اما می‌خواهد ساختار کد برنامه‌اش همچنان تمیز بماند. نیما در مورد الگوی طراحی adapter مطلبی خوانده بنابراین می‌خواهد از این الگو در برنامه‌ی خود استفاده کند. به نیما در انجام این کار کمک کنید.

جزئیات برنامه

فایل‌های اولیه‌ی برنامه را از این لینک دانلود کنید.

اینترفیس `NumberSorter`

این اینترفیس شامل یک متد `void` با نام `sort` است که آرایه‌ای از اعداد دریافت می‌کند و اعداد را مرتب می‌کند.

```
1 public interface NumberSorter {
2     public void sort(Integer[] numbers);
3 }
```

دو کلاس `NonIncreasingSorter` و `NonDecreasingSorter` نیز در حال حاضر در برنامه پیاده‌سازی شده‌اند که اینترفیس `NumberSorter` را `implement` می‌کنند.

توجه کنید که در پیاده‌سازی این مرتب‌سازی‌ها نیما از کلاس‌های خود جاوا استفاده کرده، همانطور که احتمالا خودتان هم حدس زده‌اید، خط زیر آرایه را به شکل غیرصعودی مرتب می‌کند.

```
Arrays.sort(numbers, Collections.reverseOrder());
```

همچنین جالب است بدانید کلاس `Arrays` یکسری متد کمکی روی آرایه‌های معمولی تعریف شده از جمله سورت و خیلی اوقات استفاده از این متدهای به جای پیاده‌سازی مجدد، جلوی کد تکراری را می‌گیرد. در این لینک می‌توانید چند نمونه استفاده از این کلاس را ببینید.

کلاس `SorterBySetBitsCount`

این کلاس، همان برنامه‌ی محمدرضاست که شامل متد `sortBySetBitsCount` است. این متد، آرایه‌ای از رشته‌های باینری دریافت می‌کند و آن‌ها را بر اساس تعداد بیت‌های `1` به‌صورت غیرنزولی مرتب می‌کند.

مشکل اینجاست که محمدرضا با قواعد کد تمیز نیما کد نزده، یعنی کلاسش اینترفیس `NumberSorter` را پیاده‌سازی نمی‌کند و به همین سبب امکان استفاده مستقیم از کلاس محمدرضا با کلاس‌های نیما وجود ندارد. برای همین منظور از یک کلاس adapter برای سازگارسازی استفاده می‌کنیم.

کلاس `SetBitsCountSorterAdapter`

این کلاس در برنامه موجود نیست و باید توسط شما پیاده‌سازی شود. این کلاس باید اینترفیس `NumberSorter` را `implement` کند.

- کانستراکتور این کلاس را طوری پیاده‌سازی کنید که نمونه‌ای از کلاس `SorterBySetBitsCount` دریافت کند تا برای مرتب‌سازی اعداد از آن استفاده کند.
- متد `sort` این کلاس را طوری پیاده‌سازی کنید که از کلاس `SorterBySetBitsCount` برای مرتب‌سازی اعداد استفاده کند، یعنی اعداد را به معادل باینری تبدیل کرده، مرتب کند و آن‌ها را مجدداً به مبنای ۱۰ برگرداند.

مثال

پس از پیاده‌سازی کلاس `SetBitsCountSorterAdapter` ، با اجرای متد `main` موجود در کلاس `Main` :

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        NumberSorter sorter = new SetBitsCountSorterAdapter(
                new SorterBySetBitsCount());
        Integer[] numbers = new Integer[] { 1, 2, 3, 4, 5, 6, 7, 8 };
        sorter.sort(numbers);
        System.out.println(Arrays.toString(numbers));
    }
}
```

خروجی برنامه به‌صورت زیر خواهد بود:

```
[1, 2, 4, 8, 3, 5, 6, 7]
```

آن‌چه باید آپلود کنید

پس از پیاده‌سازی کلاس `SetBitsCountSorterAdapter`، یک فایل با نام `SetBitsCountSorterAdapter.java` آپلود کنید.

مافیا

- 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:

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

که فیلد godfather موجود در کلاس را برمیگرداند.

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

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:

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

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.

```java
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:

```java
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);
```

```
27          g.action();
28          System.out.println(God.playerTypes());
29
30          d.setSaved(a);
31          d.action();
32          System.out.println(God.playerTypes());
33
34          x.setVoted("a");
35          y.setVoted("a");
36          g.setVoted("b");
37          a.setVoted("b");
38          b.setVoted("x");
39          d.setVoted("y");
40          God.killedAtDay();
41          System.out.println(God.playerTypes());
42
43          b.setVoted("a");
44          God.killedAtDay();
45          System.out.println(God.playerTypes());
46      }
47  }
```

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.

Stream

- سطح سوال: سخت

در زبان جاوا یک کلاس stream وجود دارد که با استفاده از آن میتوان عملیات مختلفی را بر روی انواعی از داده ها مانند آرایه پیاده سازی کرد. به طور مثال میتوانیم تمام اعضای آن را باهم جمع کنیم، تمام اشیا را به یک ویژگی خاص از آن شی نسبت دهیم و ... .

در این سوال میخواهیم امکانی نسبتا شبیه به stream را برای شی فرضی Instance پیاده سازی کنیم.

ابتدا پروژه اولیه را از این لینک دانلود کنید.

کلاس Instance

این کلاس دارای فیلدهای زیر است:

- ویژگی a و b و c از جنس String
- ویژگی x از جنس int
- ویژگی y از جنس long
- ویژگی z از جنس double
- ویژگی p از جنس boolean

برای این ویژگی ها setter و getter تعریف شده است. همچنین این کلاس دارای یک تابع toString و equals و hashCode می باشد که شیوه ی عملکرد آن ها برای این سوال حائز اهمیت نمی باشد.

کلاس StreamImp

این کلاس دارای فیلدهای زیر است:

- ویژگی instance از جنس Instance[]
- ویژگی size از جنس int

برای این ویژگی ها setter و getter تعریف شده است و همچنین این کلاس دارای یک Constructor است که با دریافت آرایه ای از Instanceها ویژگی های کلاس را مقداردهی می کند.

متدهای موجود در کلاس را طبق دستورالعمل زیر پیاده سازی کنید:

۱. متد `print` :

```
public String print(){
 // TODO
}
```

این متد یک String شامل تمام اشیا موجود در آرایه ی instance را برمیگرداند. پس از ویژگی های هر شی، در خط بعد ویژگی های شی بعد می آید.

مثال

```
Instance a=new Instance("a", "b", "c", 1, 2, 3, true);
Instance b=new Instance("a", "b", "c", 1, 2, 3, true);
Instance c=new Instance("d", "b", "e", 1, 4, 3, true);
c.setP(false);
Instance[] instances={a, b, c};
StreamImp streamImp=new StreamImp(instances);
String s = streamImp.print();
/*
s = a b c
1 2 3.0
true
a b c
1 2 3.0
true
d b e
1 4 3.0
false
*/
```

۲. متد `simpleLimit` :

```
public Instance[] simpleLimit(int num){
 // TODO
}
```

این متد آرایه ای از Instance ها برمیگرداند که شامل **num عضو آخر** فیلد instance شی فعلی نمی باشد.

۳. متد `limit` :

```
public StreamImp limit(int num){
 // TODO
}
```

این متد یک StreamImp **جدید** برمیگرداند که آرایه ی instance موجود در آن شامل **num عضو آخر** آرایه ی instance شی فعلی نمی باشد.

۴. متد `simpleSkip` :

```
public Instance[] simpleSkip(int num){
 // TODO
}
```

این متد آرایه ای از Instance ها برمیگرداند که شامل **num عضو اول** فیلد instance شی فعلی نمی باشد.

۵. متد `skip` :

```
public Instance[] skip(int num){
 // TODO
}
```

این متد یک StreamImp **جدید** برمیگرداند که آرایه ی instance موجود در آن شامل **num عضو اول** آرایه ی instance شی فعلی نمی باشد.

۶. متد `simpleMap` :

```
public String[] simpleMap(String var){
 // TODO
}
```

این متد آرایه ای از String ها برمیگرداند که عضو `i` ام آن ویژگی `var` عضو `i` ام آرایه ی instance شی فعلی می باشد.

مثال

```
Instance a=new Instance("m", "n", "o", 43, 123456, 2.5, true);
Instance b=new Instance("p", "q", "t", 17, 37625, 15.54, false);
Instance[] instance={a, b};
StreamImp s=new StreamImp(instance);
String[] map=s.simpleMap("x");
// map = ["43", "17"]
map=s.simpleMap("b");
// map = ["n", "q"]
```

توجه داشته باشید که اگر شی را به یک ویژگی غیر String (مثلا x که از جنس int است.) مپ کنیم، آن ویژگی را به فرم String در آرایه ی بازگشتی متد simpleMap ذخیره می کنیم.

۷. متد `map` :

```
public StreamImp map(String var){
 // TODO
}
```

این متد تقریبا مانند simpleMap عمل میکند با این تفاوت که به جای آرایه ای از String ها آرایه از Instance ها داریم که تمام ویژگی های هر عضو آن، به جز ویژگی `var` ، مقداردهی نشده اند. سپس یک StreamImp **جدید** برمیگردانیم که آرایه ی instance آن، آرایه ی Instanceهای ساخته شده در این متد باشد.

۸. متد `simpleFilter` :

```
public String[] simpleFilter(String var, String type){
 // TODO
}
```

در این متد، ابتدا هر عضو از آرایه ی instance را به ویژگی `var` آن مپ کرده، سپس آن ویژگی هایی که مقدارشان برابر `type` نمی باشد را از آرایه حذف می کنیم و آرایه ی باقی مانده را برمیگردانیم.

توجه داشته باشید که اگر بخواهیم یک ویژگی غیر String (مثلا x که از جنس int است.) از یک شی را فیلتر کنیم، باید مقدار String آن ویژگی برابر `type` باشد.

مثال

```
Instance a=new Instance("m", "n", "o", 43, 123456, 2.5, true);
Instance b=new Instance("p", "n", "t", 17, 37625, 15.54, false);
Instance c=new Instance("s", "n", "d", 986, 25632, 2.5, true);
Instance[] instance={a, b, c};
StreamImp s=new StreamImp(instance);
String[] filter=s.simpleFilter("z", "2.5");
// filter = ["2.5", "2.5"]
filter=s.simpleFilter("b", "n");
// filter = ["n", "n", "n"]
```

۹. متد `filter` :

```
public StreamImp filter(String var, String type){
 // TODO
}
```

این متد تقریبا مانند simpleFilter عمل میکند با این تفاوت که به جای آرایه ای از String ها آرایه از Instance ها داریم که اعضای آن، فقط عضوهایی هستند که ویژگی `var` آن ها برابر `type` است. سپس یک StreamImp **جدید** برمیگردانیم که آرایه ی instance آن، آرایه ی Instanceهای ساخته شده در این متد باشد.

۱۰. متد `count` :

```
public int count(){
// TODO
}
```

این متد تعداد اعضای آرایه ی instance های شی StreamImp فعلی را برمیگرداند.

۱۱. متد `sum` :

```
public String sum(String type){
// TODO
}
```

در این متد، اگر `type` اشاره به نام یک String یا boolean از ویژگی های موجود در کلاس Instance داشته باشد، تمام آن ویژگی ها را با یک فاصله به هم متصل کرده و به شکل یک String جدید برمیگردانیم؛ و اگر `type` اشاره به یک مقدار عددی داشته باشد، تمام آن ویژگی های تمام اعضای آرایه ی instance را باهم جمع کرده، و حاصل جمع را به صورت یک String برمیگردانیم.

مثال

```
Instance a=new Instance("m", "n", "o", 43, 123456, 2.5, true);
Instance b=new Instance("p", "n", "t", 17, 37625, 15.54, false);
Instance c=new Instance("s", "n", "d", 986, 25632, 2.5, true);
Instance[] instance={a, b, c};
StreamImp s=new StreamImp(instance);
String ans=s.sum("a");
// ans = "m p s ";
ans=s.sum("x");
// ans = "1046"
```

۱۲. متد `anyMatch` :

```
public boolean anyMatch(String var, String value){
// TODO
}
```

اگر ویژگی `var` حداقل یکی از اعضای آرایه instance برابر `value` باشد، این متد `true` و در غیر این صورت `false` برمیگرداند.

۱۳. متد `allMatch` :

```
public boolean allMatch(String var, String value){
// TODO
}
```

اگر ویژگی `var` همه ی اعضای آرایه instance برابر `value` باشد، این متد `true` و در غیر این صورت `false` برمیگرداند.

۱۴. متد `noneMatch` :

```
public boolean noneMatch(String var, String value){
// TODO
}
```

اگر ویژگی `var` هیچ کدام از اعضای آرایه instance برابر `value` نباشد، این متد `true` و در غیر این صورت `false` برمیگرداند.

بخش امتیازی

۱۵. متد `distinct` :

```
public StreamImp distinct(){
// TODO
}
```

در این متد، تک به تک فیلدهای هرکدام از اعضای آرایه ی instance را به ترتیب بررسی می کنیم و اگر ویژگی `a` چند شی باهم یکسان باشند، فقط اولین شی را نگه داشته و بقیه را حذف میکنیم. سپس برای ویژگی `b` ، `c` و بقیه ی ویژگی ها این کار را انجام می دهیم تا در نهایت هیچ دو شی ای هیچ ویژگی یکسانی نداشته باشند. سپس یک StreamImp **جدید** برمیگردانیم که آرایه ی instance آن، آرایه ی Instanceهای ساخته شده در این متد باشد.

نکات تکمیلی

- پیشنهاد میشود حتما متدهای `print` و `simpleMap` را پیاده سازی کنید زیرا متد اول در تعداد زیادی از تست ها مورد استفاده قرار گرفته شده است و متد دوم در پیاده سازی تعداد زیادی از دیگر متدها می تواند به شما کمک کند.
- برخی از متدها یک StreamImp برمیگردانند که این ویژگی باعث میشود بتوانیم از سینتکس هایی مشابه سینتکس زیر استفاده کنیم:

```
streamImp.filter("x", "1").map("c").count();
```

که به این عمل `chaining` گفته میشود. تستی با این عنوان وجود دارد که چنین قابلیتی در آن بررسی شده است.

آنچه باید آپلود کنید

فایل ابتدایی که دانلود کردید را پس از کامل کردن زیپ کرده و آپلود کنید.

اگر از پیاده سازی متدی منصرف شدید یک مقدار برگشتی متناسب با مقدار برگشتی آن متد برگردانید. (توجه داشته باشید که فایلتان خطای کامپایل نداشته باشد و امضای هیچ متدی را نیز از آن حذف نکنید.)