TypeMapper

عنوان

- محدودیت زمان: ۱ ثانیه
- سطح: متوسط
- طراح: سید محمد حسینی

در اینجا می‌خواهیم یک Map را با ویژگی‌های منحصر به فرد و متفاوت از Map اصلی جاوا پیاده سازی کنیم.

ابتدا پروژه اولیه را این لینک دانلود کنید.

کلاس Priority: 🔗

#فیلدها:

```
1 | private A[] keys;
2 | private B[] values;
3 | private int capacity;
4 | private int size ;
```

keys : مقادیری منحصربه فرد هستند. values : مقدار هر key است که می‌تواند منحصربه فرد نباشد.

capacity :بیشترین ظرفیت ممکن برای دو آرایه keys و values . size : اندازه دو آرایه keys و values در حال حاضر.

#کانستراکتور: سازنده این کلاس به شکل زیر تعریف می‌شود:

```
1 | public Priority(int capacity);
```

در سازنده این کلاس حداکثر ظرفیت آرایه‌ها داده می‌شود.

#متدها:

```
1 | public void addKeyValue(A key,B value);
```

زوج مرتب key و value را به آرایه‌های موردنظر اضافه می‌کند.

- اگر تعداد زوج مرتب‌ها از حداکثر ظرفیت بیشتر شود، کاربر با خطای FullException پرتاب می‌شود.
- اگر حداکثر ظرفیت برابر صفر باشد خطای EmptyException پرتاب می‌شود.
- اگر مقدار key برابر null باشد خطای NullElementException پرتاب می‌شود.

```
1 | public B peek(A key);
```

این متد مقدار key مورد نظر را برمی‌گرداند.

- اگر حداکثر ظرفیت برابر صفر باشد خطای EmptyException پرتاب می‌شود.
- اگر key موردنظر یافت نشد خطای ElementNotFoundException پرتاب می‌شود.

```
1 | public A chooseFirst(B value);
```

این متد اولین key را که مقدار آن برابر value موردنظر است برمی‌گرداند

- اگر حداکثر ظرفیت برابر صفر باشد خطای EmptyException پرتاب می‌شود.
- اگر زوج موردنظر یافت نشد خطای KeyNotFoundException پرتاب می‌شود.

```
1 | public A chooseLast(B value);
```

این متد آخرین key را که مقدار آن برابر value

- است برمی‌گرداند.
  - اگر حداکثر ظرفیت برابر صفر باشد خطای EmptyException پرتاب می‌شود
- اگر زوج موردنظر یافت نشد خطای KeyNotFoundException پرتاب می‌شود.

```
1 | public B remove();
```

این متد اولین زوج را حذف می‌کند.

- اگر حداکثر ظرفیت برابر صفر باشد خطای `EmptyException` پرتاب می‌شود.

```
1 | public B remove(A key);
```

این متد زوج مورد نظر که `key` آن برابر مقدار داده شده به متد است را حذف می‌کند.

- اگر حداکثر ظرفیت برابر صفر باشد خطای `EmptyException` پرتاب می‌شود.
- اگر زوج موردنظر یافت نشد خطای `KeyNotFoundException` پرتاب می‌شود.

```
1 | public void updateElement(A key,B newValue);
```

این متد با گرفتن `key` موردنظر و مقدار جدید `value` ، مقدار `value` را برای `key` مورد نظر تغییر می‌دهد.

#آنچه باید آپلود کنید شما می‌بایست فایل Priority.java را همراه با خطاهای موجود در پروژه اولیه زیپ کرده و آپلود کنید.

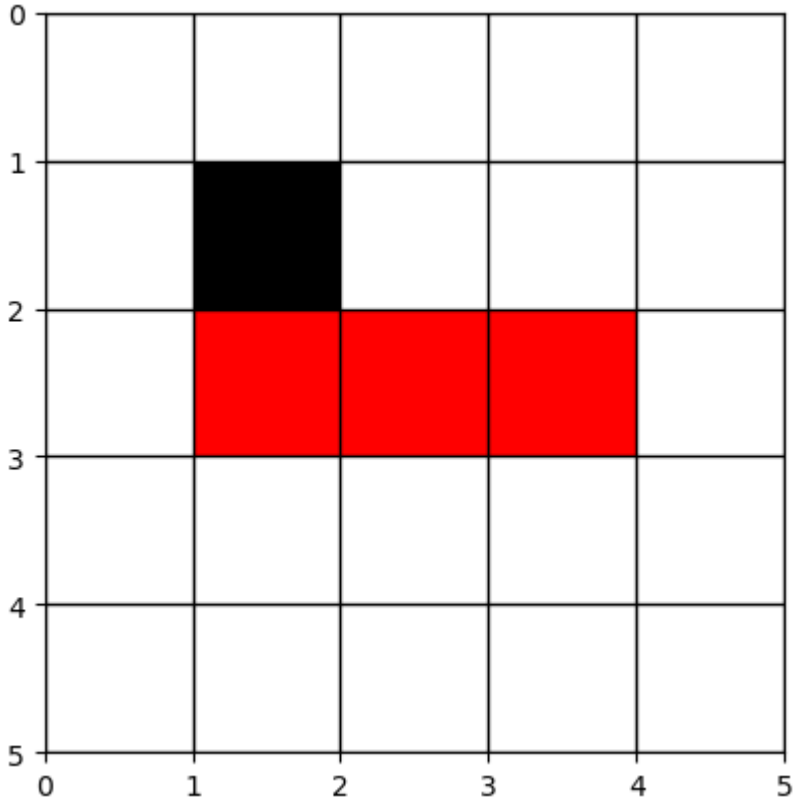
بازی مار

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: آسان
- طراح: آرمان اسدی

شایان به دلیل علاقه زیادی که به بازی های رایانه ای داشت، تصمیم گرفت که عضو انجمن Game Lab دانشگاه شهید بهشتی شود. اعضای انجمن از شایان خواستند تا با طراحی یک بازی خود را به آنان ثابت کند. شایان قصد دارد بازی مار را طراحی کند و خود را به اعضای انجمن ثابت کند. اما متاسفانه شایان درگیر پروژه های دانشگاه است و زمان کافی برای طراحی کامل بازی را ندارد. او از شما می‌خواهد در توسعه قسمتی از این بازی به او کمک کنید. ابتدا فایل توسعه داده شده توسط شایان را از این لینک دانلود کنید.

شکل بازی

زمین بازی از یک مربع n در n تشکیل شده که آنرا به سلول های ۱ در ۱ تقسیم بندی کرده ایم. به شکل زیر دقت کنید:

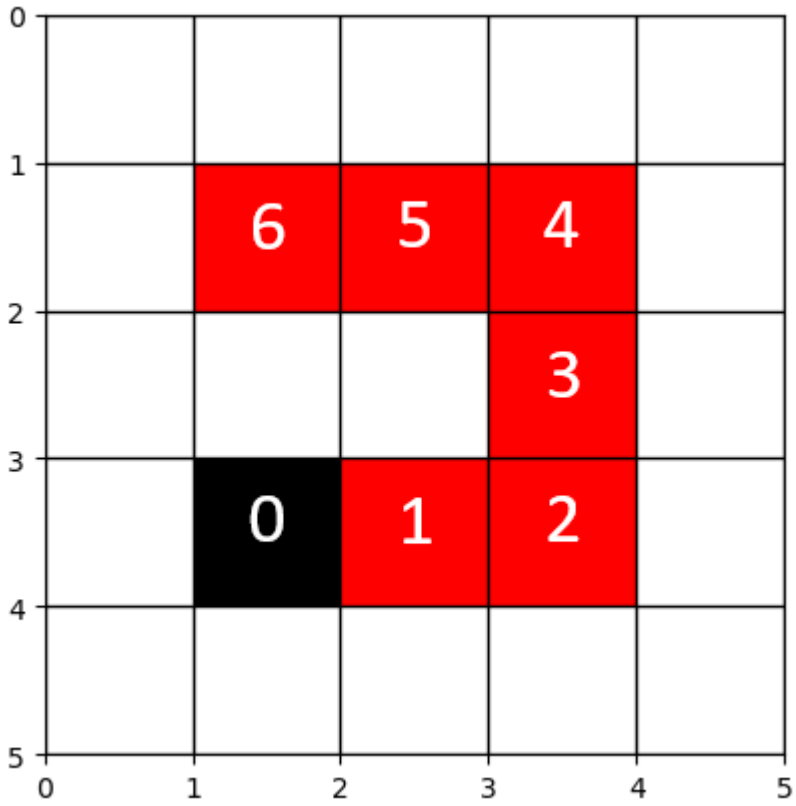


همانطور که در شکل بالا مشاهده می‌کنید بدن مار از سلول هایی به رنگ قرمز که به صورت پیوسته به یکدیگر متصل شده اند، تشکیل شده است. سر مار نیز با رنگ مشکی مشخص شده است. **دقت کنید که در قسمتی از پروژه که وظیفه پیاده‌سازی آن بر عهده شماست، فرض بر این است که سیبی در زمین وجود ندارد و اندازه مار همواره ثابت است.**

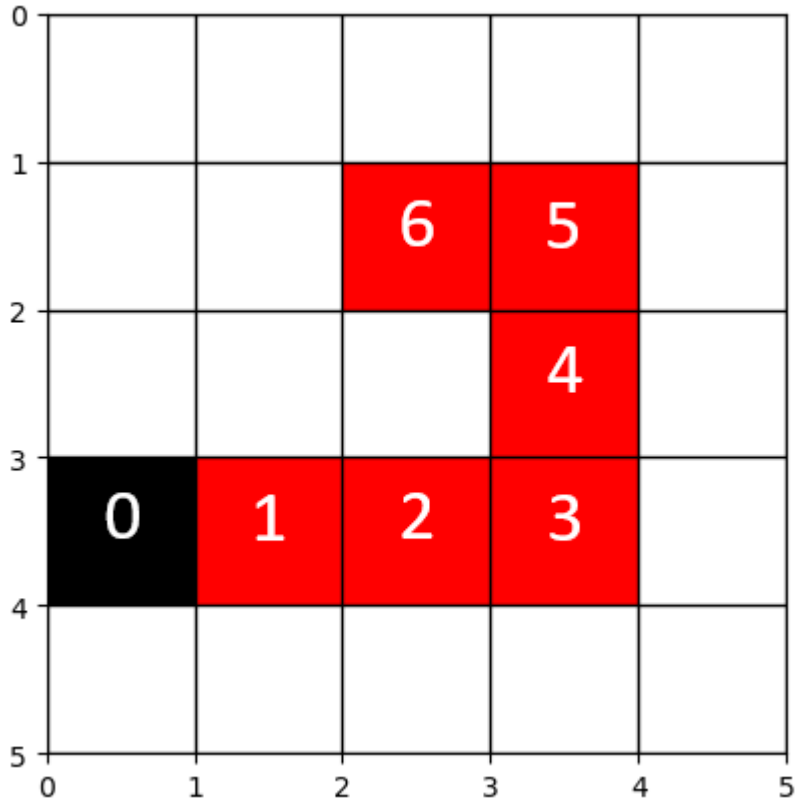
نحوه حرکت مار

در هر حرکت، سر مار به یکی از چهار جهت بالا، پایین، راست یا چپ حرکت می‌کند و بعد از حرکت سرش از سلولی که در آن قرار دارد به سلول مجاوری که در جهت حرکت است، منتقل می‌شود. سپس سلول iام از بدن مار به سلولی از زمین بازی منتقل می شود، که سلول $i-1$ام بدن مار قبل از انجام حرکت در آنجا بوده است. نحوه شماری گذاری سلول های بدن مار به این صورت است که اگر طول مار L باشد، سلول سر مار شماره 0 و سلول دم مار شماره $L-1$ را می‌گیرد. بقیه سلول ها هم به ترتیب از 2 تا $L-2$ شماره گذاری می‌شوند. برای فهم بهتر از حرکت مار به مثال زیر توجه کنید.

قبل از حرکت:



بعد از حرکت به سمت چپ:



حال شما باید پروژه شایان را توسعه بدهید و کلاس هایی را که در ادامه به توصیح آنها می‌پردازیم را پیاده‌سازی کنید.

کلاس Cell

این کلاس یک سلول از زمین بازی را مدل می کند.

فیلد ها

این کلاس دارای فیلد های زیر است:

```
1 public boolean markedAsSnakeHead;
2 public boolean markedAsSnakeBody;
```

- `markedAsSnakeHead` : این فیلد مشخص می‌کند که آیا سر مار در این سلول قرار دارد یا خیر.
- `markedAsSnakeBody` : این فیلد مشخص می‌کند که آیا سلولی از بدن مار در این سلول قرار دارد یا خیر.

متد کانستراکتور

متد کانستراکتور این کلاس به شکل زیر است:

```
1 Cell(boolean markedAsSnakeHead, boolean markedAsSnakeBody);
```

کلاس Coordinate

در ادامه خواهیم دید که زمین بازی از یک آرایه دوبعدی از سلول ها تشکیل شده است. این کلاس به مختصات یک سلول خاص از زمین بازی اشاره می‌کند.

فیلد ها

این کلاس دارای فیلد های زیر است:

```
1 public int rowNumber;
2
```

```
public int columnNumber;
```

- `rowNumber` : این فیلد به شماره سطر یک سلول از زمین بازی اشاره می‌کند.
- `columnNumber` : این فیلد به شماره ستون یک سلول از زمین بازی اشاره می‌کند.

متد کانستراکتور

متد کانستراکتور این کلاس به شکل زیر است:

```
Coordinate(int rowNumber, int columnNumber);
```

کلاس Game

این کلاس برای شبیه‌سازی بازی است.

فیلد ها

این کلاس دارای فیلد های زیر است:

```
private int size;
private Cell[][] playground;
private Coordinate[] snake;
```

- `size` : این فیلد اندازه زمین بازی را مشخص می‌کند.
- `playground` : این فیلد برای مدل کردن زمین بازی استفاده می‌شود. به این صورت که $playground[i][j]$ به سلول واقع در سطر iام و ستون jام از زمین بازی اشاره می‌کند.
- `snake` : این فیلد برای مدل کردن مار استفاده می‌شود. به این صورت که $snake[i]$ یک شی Coordinate است که به مختصات سلولی از زمین بازی اشاره می‌کند، که سلول iام بدن مار در آن قرار دارد. **دقت کنید که $snake[0]$ به مختصات سر مار اشاره می‌کند.**

متد کانستراکتور

متد کانستراکتور این کلاس به شکل زیر است:

```
Game(int size, Coordinate[] snake){
    //TODO
    loadGame();
}
```

شما در این کانستراکتور فقط باید قسمت TODO را تکمیل کنید.

متد ها

```
public void loadGame();
```

این متد برای لود کردن بازی استفاده می‌شود. به این صورت که شما باید سلول هایی از زمین بازی را که سر و بدن مار در آن قرار دارد را مارک کنید.

```
public void move(Direction direction) throws HittingTheWallExcep
```

این متد یک enum از جنس Direction می‌گیرد که جهت حرکت مار را مشخص می‌کند. سپس طبق توضیحاتی که قبل تر داده شد، مار یک حرکت به جهت مشخص شده انجام می‌دهد. پس از انجام حرکت سه حالت ممکن است رخ دهد:

۱. **مار با دیوار برخورد می‌کند**: در این حالت یک اکسپشن از نوع `HittingTheWallException` پرتاب می‌شود. متن پیام این اکسپشن به صورت زیر است:

```
The snake hit the wall.
```

۲. **مار با بدن خود برخورد می‌کند**: در این حالت یک اکسپشن از نوع `SelfInflictException` پرتاب می‌شود. متن پیام این اکسپشن به صورت زیر است:

```
The snake bit itself.
```

۳. **مار بدون هیچ مشکلی حرکت خود را انجام می‌دهد**: در این حالت وضعیت زمین بازی آپدیت می‌شود.

```java
public void displayPlayground(){
    for(int i = 0; i < size ;i++){
        for(int j = 0; j < size ;j++){
            if(playground[i][j].markedAsSnakeHead)
                System.out.print("H");
            else if(playground[i][j].markedAsSnakeBody)
                System.out.print("B");
            else
                System.out.print(".");
        }
        System.out.println();
    }
}
```

این متد زمین بازی را چاپ می‌کند. سلول هایی که خالی هستند با `.` نمایش داده می‌شوند. سلولی که سر مار در آن قرار دارد با `H` و سلول هایی که بدن مار در آن قرار دارد با `B` نمایش داده می‌شوند. **پیاده‌سازی این متد باید به همین شکل باشد و شما نباید آن را تغییر دهید.**

جهت حرکت مار

جهت حرکت مار یک enum از جنس Direction است که به صورت زیر تعریف می‌شود:

```java
public enum Direction {
    UP,
    DOWN,
    RIGHT,
    LEFT
}
```

آنچه باید آپلود کنید

شما باید یک فایل zip با ساختار زیر آپلود کنید:

```
<zip_file_name.zip>
    ├── Cell.java
    ├── Coordinate.java
    ├── Game.java
```

```
├── HittingTheWallException.java
├── SelfInflictException.java
├── Direction.java
```

Hunter, the programmer

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: سخت
- طراح: مهرسا سمیع‌زاده

شکار و شکارچی از دیرباز زندگانی آدمیان را درگیر خود کرده، ولیکن حال که در سال 2024 به سر می‌بریم، وقت آن رسیده که نحوه شکار را upgrade کنیم. به شکارچی قصه‌ی ما کمک کنید، برنامه ای بنویسد تا بتواند روند شکار خود را handle کند.

جزئیات برنامه

ابتدا پروژه اولیه را این لینک دانلود کنید. شکارچی در جنگل به دنبال شکار سه حیوان، Bird ،Deer و Wolf است. و شما برای نگهداری لیست حیوانات نیاز دارید که data type کاستومایز شده‌ای به نام WildList را پیاده سازی کنید.

توجه کنید ترتیب اد کردن حیوانات به جنگل ، در واقع ترتیب فاصله انها از هانتر است(نشان دهنده‌ی distance).

در مواجهه با حمله، Deer احتمال flee و Bird احتما fly و فرار از حمله را دارند.

با کاهش سلامت یک حیوان، ارزش آن نیز کاهش پیدا می‌کند.

در صورت شلیک هانتر به یک حیوان، حیوانات نزدیک محل تیراندازی نیز به دلیل ترس تحت تاثیر قرار می‌گیرند ،Bird به انتهای جنگل میرود ، Deer حداکثر دو حیوان دورتر می‌شود، و Wolf حمله میکند.

برای درک بهتر، جنگل را مانند آرایه ای در نظر بگیرید که هانتر در خانه صفر، حیوان اول در خانه 1 ، حیوان دوم در خانه 2 .... و فاصله هر حیوان تا هانتر با شماره خانه‌اشان برابر است و همچنین انتهای جنگل به معنای جایگاه nام (n تعداد حیوانات) می‌باشد.

توجه داشته باشید ارزش هر حیوان به میزان سلامتش بستگی دارد، مثلا یک حیوان با سلامت 85، ارزشش از مثلا 70 به 59.5 (هشتاد و پنج درصدِ ارزش اولیه) کاهش پیدا می‌کند.

متد های getter و setter برای فیلد های کلاس ها پیاده سازی کنید.

کلاس Hunter

```
public class Hunter <T extends Animal>
```

که دارای فیلدهای زیر است. توجه کنید در ابتدا baseHit برابر با 30، و health برابر با 100 می‌باشد.

```
private int health;
private JungleLaw jungleLaw;
private Double baseHit;
private WildList<T> hunts;
private int experience;
```

سازنده‌ی این کلاس به صورت زیر است.

```
public Hunter(JungleLaw jungleLaw){
    //TODO
}
```

این کلاس شامل متد های زیر است.

```
Public double calculateHit(T animal) throws Exception{
    //TODO
}
```

قدرت ضربه هانتر را در هر تیراندازی محاسبه می‌کند.

قدرت ضربه‌ی وی به فاصله‌ی حیوان، سلامت هانتر و تجربه او بستگی دارد.

۱. پس از هر shoot موفقیت امیز، یکی به exprience هانتر اضافه می‌شود، و سپس به اندازه ده‌درصدتجربه به baseHit اضافه می‌شود (دقت کنید پس از هر تیراندازی موفق، baseHit ما تغییر می‌کند و باید اپدیت شود).

۲. برای وارد کردن ضربه به حیوان مورد نظر باید به میزان فاصله‌ی حیوان ، از baseHit ما کاسته می‌شود.

۳. اما هانتر به اندازه سلامتش می‌تواند از قدرت ضربه‌ی به دست آمده استفاده کنند، درصد سلامتش، میزان درصدی از قدرت ضربه است که می‌تواند در واقعیت ضربه وارد کند.

▼ مثال

زمانی که سلامت هانتر 90 باشد و تجربه 2 تیراندازی موفقیت امیز را داشته باشد، متد shoot برای حیوانی که در فاصله‌ی 2 قرار دارد فراخوانی می‌شود. در حالت اولیه baseHit ما 30 است. پس از تجربه اول:

- به exprience یکی اضافه می‌شود.
- سپس baseHit ما اپدیت می‌شود.

```
baseHit = baseHit + exprience*0.1
baseHit = 30 + 1*0.1
baseHit = 30.1
```

تجربه دوم:

```
baseHit = baseHit + exprience*0.1
baseHit = 30.1 + 2*0.1
baseHit = 30.3
```

اکنون که متد shoot فراخوانی می‌شود، baseHit ما 30.3 است، که :

فاصله را کم میکنیم:

```
30.3 - 2 = 28.3
```

از انجایی که سلامت هانتر 90 است، از 90 درصد 28.3 می‌تواند استفاده کند.

```
0.9 * 28.3 = 25.47
```

پس در نهایت ضربه ای که وارد می‌کند 25.47 قدرت دارد.

و در اخر باید به تجربه اضاف کنیم و baseHit را اپدیت کنیم.

```
Public String Shoot(T animal) throws Exception{
    //TODO
```

```
}
```

این متد روند تیراندازی را کنترل می‌کند، برای این کار نیاز دارید قدرت ضربه را محاسبه کنید. در صورت موفقیت‌آمیز بودن تیراندازی `Shot successful!` و در غیر این صورت `Shot missed!` را پرینت می‌کند.

توجه کنید، در صورتی که نتیجه این متد true باشد، به تجربه هانتر اضافه میشود.

```
public boolean SaveHunt(T animal){
    //TODO
}
```

این متد عملیات شکار را هندل می‌کند و لیست شکار را اپدیت می‌کند.

در صورت موفقیت‌آمیز بودن `<animal name> added to hunts.` و اگر حیوان مورد نظر قبلا شکار شده‌بود `Cannot save <animal name>, already in hunts.` پرینت شود.

```
public Double getTotalHuntValue()throws Exception {
    //TODO
}
```

جمع ارزش تمام شکار هارا برمی‌گرداند.

کلاس انتزاعی Animal

هر سه حیوان از این کلاس ازث‌بری می‌کنند. که دارای فیلد زیر است. توجه کنید سلامت اولیه هر حیوان 100 است.

۱. ویژگی name از نوع String
۲. ویژگی value از جنس Double
۳. ویژگی hunter از جنس Hunter
۴. ویژگی health از جنس int
۵. ویژگی jungleLaw از جنس JungleLaw

سازنده‌ی این کلاس به صورت زیر است.

```
public Animal(String name,Double value, JungleLaw jungleLaw, Hun
    //TODO
}
```

این کلاس شامل متد های زیر است.

```
abstract <E> void attack(E e);
abstract <E> void gettingAttacked(E e);
abstract void trigger();
```

```
public boolean isAlive(){
    //TODO
}
```

کلاس های Deer ، Bird ، Wolf

این کلاس‌ها از کلاس انتزاعی Animal ارث‌بری می‌کنند و فقط همان سازنده را صدا میزنند.

در این کلاس ها شما نیاز دارید که متد های attack و trigger کنید.

```
@Override
public void attack(E e){
    //TODO
};
@Override
public void gettingAttacked(E e){
    //TODO
};
@Override
public void trigger(){
    //TODO
};
```

نکات مورد توجه برای پیاده سازی این متدها:

هانتر به هرسه حیوان می‌تواند حمله کند، Wolf به Hunter و Deer می‌تواند حمله کند ، Bird با شدت خیلی کم به هر سه حمله می‌کند و Deer توانایی حمله را ندارد.

قدرت حمله Wolf ، بیست (20) است و Bird در صورت حمله، قدرتش 5 است

درصورتی که حمله‌ای خارج این حالات صورت بگیرد، و یا پارامتر پاس شده null باشد، اکسپشن پرتاب می‌کند. (در انتهای سوال اکسپشن های لازم را بررسی کنید)

کلاس **Deer** متد flee را پیاده سازی می‌کند که در صورت حمله شدن به این حیوان اگر فاصله حمله کننده حداقل 3، و ضربه وارد شده کمتر از 20 باشد، به فاصله‌ای یکی افزایش میابد و دورتر می‌شود و از حمله فرار می‌کند. (سلامتش ثابت، فاصله اش یکی زیاد)

کلاس **Bird** در صورت حمله شدن به این حیوان اگر فاصله حمله کننده حداقل 2، و ضربه وارد شده کمتر از 15 باشد، از حمله فرار می‌کند (تغییری در سلامت و فاصله اش ایجاد نمیشود، و حمله دفع میشود).

متد زیر رفتار هر حیوان، در زمانی که هانتر به حیوانی در نزدیکی او، حمله می‌کند را مشخص می‌کند.حیواناتی که در فاصله 1 از حیوان مورد هدف قرار دارند trigger می‌شوند.

```
1 | abstract void trigger();
```

کلاس **Wolf** :

صدای تیر، خوی وحشی وی را تحت تاثیر قرار می‌دهد و به هانتر حمله میکند.

کلاس **Deer**:

از ترس، فرار میکند. به اندازه یک یا دو از حیوان مورد حمله دور می‌شود. اگر تعداد حیوانات 7 باشد و حیوان 5 مورد حمله باشد و deer در جایگاه 6 باشد، به اندازه یکی فرار می‌کند و به 7 می‌رود. اما اگر حیوانات 8 تا باشند، deer به 8 می‌رود (حداکثر تغییر فاصله اش 2 است). بدیهی است که اگر امکان تغییر مکان با هیچ میزانی(1 یا 2) وجود نداشت، deer در جایگاه خود میماند.

کلاس **Bird**:

به اخر جنگل میرود. (جایگاه اخرین حیوان)

کلاس JungleLaw

این کلاس اتفاقات جنگل را کنترل می‌کند.

دارای فیلد زیر است.

۱. ویژگی hunter از نوع Hunter
۲. لیستی از jungleAnimals از نوع (WildAnimal)
۳. ویژگی numberOfAnimals از نوع int

سازنده‌ی این کلاس به این صورت است.

```
public JungleLaw(){
    //TODO
}
```

این کلاس دارای متدهای زیر است:

```
public <A extendss Animal> boolean addAnimal(A Animal){
    //TODO
}
```

حیوانات به جنگل(لیست) اضافه میشوند. توجه کنید در صورت مرگ انها باید از جنگل خارج شوند. توجه کنید ترتیب اد کردن حیوانات به جنگل ، در واقع ترتیب فاصله انها از هانتر است(نشان دهنده‌ی distance). مثلا deer ابتدا اد شود و سپش bird ، اگر distance انهارا برگردانیم ، (با استفاده از getDistance کلاس WildList)، deer یک را برمی‌گرداند و bird دو.

```
public void handleTriggers(int dis) throws Exception{
    //TODO
}
```

این متد مشخص میکند کدام حیوانات trigger شوند. (حیوانات با فاصله یکی از حیوانی که مورد حمله قرار گرفته) مثلا حیوان 5 ام مورد حمله است، پس حیوان 4 و 6 trigger میشوند.(در صورت وجود)

کلاس WildList

این کلاس درواقع برای نگهداری Animal ها نیاز داریم که همانند ArrayList عمل می‌کند. و شما نیاز دارید ویژگی های زیر را پیاده سازی کنید.

توجه کنید این کلاس توانایی کار با تمام سه نوع حیوان را دارد. و دیتای تایپ ورودی باید به این سه نوع محدود شود.

فیلد این کلاس:

```
1 private A[] animals;
2 private int size;
3 private static final int DEFAULT_CAPACITY=10;
```

سازنده مورد نیاز:

ابتدا وایلدلیستی با اندازه صفر ساخته میشود، سپس با اضافه کردن حیوانات مورد نظر، سایز نیز تغییر میکند و کم و زیاد می‌شود.

اما ارایه مورد استفاده (animals) را با اندازه اولیه 10 ایجاد میکنیم.

```
1 public WildList(){
2     //TODO
3 }
```

متد های مورد نیاز:

```
1 public boolean add(A animal){
2     //TODO
3 }
```

این متد حیوان جدیدی را به لیست ما اضافه میکند.

```
1 public boolean contains(A animal){
2     //TODO
3 }
```

چک میکند که ایا حیوان مورد نظر در لیست وجود دارد یا خیر.

```
public getDistance(A animal){
    //TODO
}
```

فاصله حیوان مورد نظر را از هانتر برمی‌گرداند.

```
public boolean remove(A animal){
    //TODO
}
```

```
public changeOrder(A animal){
    //TODO
}
```

حیوان مورد نظر را به انتهای لیست می‌برد.

```
public A get(int index){
    //TODO
}
```

حیوان، در ایندکس مورد نظر را برمی‌گرداند.

```
public boolean isEmpty(){
    //TODO
}
```

خالی بودن لیست را چک می‌کند.

```
public void set(A animal,int index){
    //TODO
}
```

حیوان پاس داده شده را در ایندکس مورد نظر قرار می‌دهد.توجه کنید که حیوان در لیست بوده است و صرفا جایگاهش تغییر می‌کند.

```
public int countByType(A animal){
    //TODO
}
```

این متد، تعداد اعضای لیست که تایپ یکسانی با animal ورودی دارند را برمی‌گرداند.

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

تمام حیوانات‌لیست و ارزششان را با فرمت `<animal name> - Value: <animal value>` برمی‌گرداند(هر حیوان در یک لاین).

▼ اکسپشن ها

در کلاس WildList:

کلاس های changeOrder ، remove، getDistance و set در صورت خالی بودن پارامتر پاس شده، اکسپشن با مسیج `null` پرتاب می‌کنند.

متد های remove و getDistance درصورت پیدا نکردن حیوان موردنظر پیام `Animal not found in the WildList` ، کلاس های remove ، changeOrder ، get ، set درصورت خالی بودن لیست، پیام `empty wildList` ، و کلاس های get و set اکسپشن با پیام `Invalid index` پرتاب می‌کنند.

هر سه حیوان در صورتی که متد attackشان را برای کلاسی فراخوانی شود که توانایی حمله به آن را ندارند، اکسپشن با پیام `Cannot attack <class name>` پرتاب می‌کند.

در هر دو متد attack و gettingAttacked در صورتی که کلاس پاس داده شده به متد null باشد، پیام اکسپشن `null` .

با اجرای **Main** زیر:

```
public class Main {    public static void main(String[] args) {
Hunter<Animal> hunter1 = new Hunter<>(jungleLaw);
Bird bird = new Bird("Eagle", 50.0, jungleLaw, hunter1);
Deer deer = new Deer("Bambi", 100.0, jungleLaw, hunter1);
Wolf wolf = new Wolf("Alpha", 150.0, jungleLaw, hunter1);
```

```
jungleLaw.addAnimal(deer);
jungleLaw.addAnimal(wolf);

System.out.println("Initial Jungle State:");        System.out.p
try {
System.out.println(hunter1.shoot(deer));
System.out.println("Hunt saved successfully.");            }
                hunter1.shoot(null);
} catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
                 System.out.println(hunter1.shoot(wolf));
if (!deer.isAlive()) {
                System.out.println("Deer has been hunted.");
 System.out.println("Final Jungle State:");            System.ou
    }
  }
```

خروجی به شکل:

```
Initial Jungle State:
Jungle Animals: Eagle - Value: 50.0
Bambi - Value: 100.0
Alpha - Value: 150.0
23.200000000000003
Shot successful!
true
Bambi added to hunts.
Hunt saved successfully.
Error: null
23.28
Shot successful!
true
Final Jungle State:
Jungle Animals: Alpha - Value: 114.0
Eagle - Value: 50.0
Bambi - Value: 76.0
```

آنچه که باید آپلود کنید:

باید یک فایل زیپ از کلاس های کامل شده‌ی WildList، Hunter، JungleLaw ، Deer ، Bird ، Wolf، Animal آپلود کنید.

RemoteCaller (امتیازی)

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- طراح: برنا ماهرانی

به دلیل شیوع کرونا، برنامه نویسان به این فکر افتادند که برنامه نویسی تماسی را با برنامه نویسی از راه دور جایگزین کنند.

در این شیوه برنامه نویسی، بقیه برنامه نویسان با دادن دستور به سه متد واسط، از برنامه می‌خواهند شیء مورد نظر ساخته شود، متد مورد نظر call شود یا ... .

پس از شما خواسته‌اند برنامه‌ای با استفاده از رفلکشن پیاده سازی کنید که این عمل را انجام دهد.

دستوراتی که باید برنامه شما بتواند هندل کند به شرح زیر است:

برای فهم بیشتر مثال هایی که در ادامه خواهد آمد از کلاس فرضی زیر استفاده خواهد شد:

```
class Person{
    int age;
    String name;
    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }
    public Person(String name){
        this.name = name;
        age = 18;
    }
    public Person(int age){
        this.age = age;
        name = "anonymous";
    }
    public void sayHello(int count, String secondName){
        for(int i = 0; i < count; i++){
            System.out.println("%s: Hello %s.\n");
        }

```

```
21      }
    }
```

پیش از شروع **حتما** فایل اولیه سوال را از این لینک دانلود کنید. تغییر امضای توابع باعث مشکل در داوری خواهد شد.

۱. ساخت آبجکت

متد `creator` در فایل نمونه را طوری پیاده‌سازی کنید که با دریافت یک آرگومان رشته‌ای که حاوی پیامی با فرمت زیر است، **عملیات ساخت شیء** را انجام دهد.

فرمت پارامتر `String command` :

```
create class_name arg1_class_name constructor_arg1, ..., argN_class_r
```

با وارد شدن دستور `create` و دریافت `class_name` باید آبجکتی از کلاس مورد نظر ساخته شود و **در انتها شی ساخته شده برگشت داده شود.** همچنین برای پاس دادن آرگومان های مختلف به سازنده این کلاس، در ادامه، به صورت جفت جفت ابتدا نوع آرگومان و سپس مقدار آرگومان به ترتیب پارامتر های سازنده کلاس مورد نظر وارد می‌شود.

برای سادگی نوع آرگومان وارد شده **فقط** یکی از مقادیر زیر خواهدبود:

۱. `int`
۲. `double`
۳. `String`

مثالی از کاربرد این متد:

```
1  Person p = (Person) RemoteCaller.creator("create ir.sbu.Person S
```

شیئی از کلاس `Person` از پکیج `ir.sbu` با استفاده از کانستراکتور تک پارامتری این کلاس و پاس دادن آرگومان مورد نیاز سازنده ساخته و ریترن می‌شود.

۲. صداکردن متد

متد `caller` در فایل نمونه را طوری پیاده‌سازی کنید که با دریافت دو آرگومان به ترتیب رشته ای با فرمت زیر و یک آبجکت، **عملیات صداکردن متد** را **روی آبجکت پاس داده شده** انجام دهد.

```
call method_name arg1_class_name method_arg1, ..., arg2_class_name me
```

با وارد شدن دستور `call` از شیئی که به عنوان پارامتر داده شده (`baseObject`) متدی با نام `method_name` و در صورتی که متد پارامتر خواست، در ادامه پارامتر هایی مشابه حالت پارامترهای سازنده می‌آید.

برای سادگی نوع آرگومان وارد شده **فقط** یکی از مقادیر زیر خواهدبود:

۱. `int`
۲. `double`
۳. `String`

مثالی از کاربرد این متد:

```
1 | RemoteCaller.caller("call sayHello int 2, String ahmad", p);
```

متد `sayHello` از شیء `p` (که در مثال قبل از متد `createor` دریافت کردیم)با آرگومان های اینتجری ۲ و استرینگی `ahmad` کال می‌شود. که با توجه به بدنه این متد، با خروجی زیر مواجه می‌شویم.

```
mahmood: Hello ahmad
mahmood: Hello ahmad
```

۳. مقداردهی فیلدها

متد `setter` در فایل نمونه را طوری پیاده‌سازی کنید که با دریافت دو آرگومان به ترتیب رشته ای با فرمت زیر و یک آبجکت، **عملیات ست کردن مقدار روی یک فیلد** را **روی آبجکت پاس داده شده** انجام دهد.

```
set field_name value_class_name new_value
```

با وارد شدن دستور `set` شیئی از نوع `value_class_name` با مقدار `new_value` ساخته و روی فیلد `field_name` ذخیره سازی می‌شود.

مثال:

```
RemoteCaller.setter("set age int 19", p);
```

فیلد `age` در شیئی که با دستور اول ساخته شد با مقدار اینتیجری ۱۹ مقدار دهی می‌شود.

تضمین می‌شود:

* اگر آرگومان‌های یک کانستراکتور یا متد رشته باشد، بین رشته کاما (`,`) و اسپیس وجود نخواهد داشت. * همان طور که در بدنه سوال گفته شد، تایپ آرگومان‌های سازنده و متدها در دستورات `create` ، `call` و یکی از سه مقدار `int` ، `double` یا `String` خواهد بود و متد یا سازنده ای با آرگومانی غیر از این تایپ‌ها صدا زده نخواهد شد. * در دستور `call` متدهایی با مقدار برگشتی (غیر وید) کال خواهندشد. * نام کلاسی که قرار است نمونه ای از آن ساخته شود به صورت کامل و با پکیج وارد می‌شود. مثلا اگر کلاس `Person` در پکیج `ir.sbu` قرار داشته باشد:

```
create ir.sbu.Person ...
```

راهنمایی

* برای فراخوانی کانستراکتور یا متد با تایپ آرگومان های وارد شده، از `primitive.class` استفاده کنید.

مثال:

```
//get constructor: Person(int age);

Constructor c = Class.forName("ir.sbu.Person").getConstructor(in
```

* برای ساخت یک شی و پاس دادن به متد یا کانستراکتور نیز از `Wrapper Class` مربوط به `int` و `double` و متد `valueOf` بهره ببرید.

کلاس های Wrapper و کلاس String در پکیج: java.lang قرار دارند.

```
//instantiate an int or a double

Object x = Class.forName("java.lang.Integer").getMethod("valueOf

System.out.println(x); //25
```

* چون تایپ آرگومان های متدها و سازنده‌ها محدود است (int,double,String)، برای گرفتن کلاس مربوطه می‌توانید از switch-case یا if-else استفاده کنید.

کاربرد نمونه

مثلا با اجرای کد زیر خروجی های کامنت شده مدنظر خواهد بود:

```

public static void main(String[] args) throws Exception {

        String x = (String) RemoteCaller.creator("create java.la
        System.out.println(RemoteCaller.caller("call replaceAll

        StringBuilder sb = (StringBuilder) RemoteCaller.creator(
        RemoteCaller.caller("call reverse",sb);

        System.out.println(RemoteCaller.caller("call toString",
        System.out.println(RemoteCaller.caller("call charAt int

}
```

برای راهنمایی بیشتر لطفا تست نمونه را از این لینک دانلود، بررسی و اجرا کنید. </pre/>