# باغ وحش کارتونی

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.**

# تشریحیات

*تصحیح این سؤال به‌صورت دستی است.*

۱. تفاوت بین abstract class و interface در چیست؟

۲. در کد زیر، آیا خوب است متد `hasSameArea` استاتیک شود یا خیر؟ چرا؟

Rectangle.java

```java
public class Rectangle
{
    public double width;
    public double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height:
    }

    public double area() {
        return this.width * this.height;
    }

    public boolean hasSameArea(Rectangle r1, Rectangle r2) {
        return Math.abs(r1.area() - r2.area()) < 0.005;
    }
}
```

## آن‌چه باید آپلود کنید

پاسخ خود را به‌صورت یک فایل *PDF* آپلود کنید.

# بازنویسی

*تصحیح این سؤال به‌صورت دستی است.*

در کلاس زیر:

- متد `getJalaliMonthName` را بدون استفاده از `if` ، `switch` یا عملگر سه‌عملوندی بازنویسی کنید (حداکثر یک شرط مجاز است).
- متد `isValidDate` را طبق کامنت، تنها با به‌کارگیری عبارات منظم پیاده‌سازی کنید.

DateUtil.java

```java
public class DateUtil
{
    public static String getJalaliMonthName(int number)
    {
        if (number == 1) {
            return "Farvardin";
        } else if (number == 2) {
            return "Ordibehesht";
        } else if (number == 3) {
            return "Khordad";
        } else if (number == 4) {
            return "Tir";
        } else if (number == 5) {
            return "Mordad";
        } else if (number == 6) {
            return "Shahrivar";
        } else if (number == 7) {
            return "Mehr";
        } else if (number == 8) {
            return "Aban";
        } else if (number == 9) {
            return "Azar";
        } else if (number == 10) {
            return "Dey";
        } else if (number == 11) {
            return "Bahman";
        } else if (number == 12) {
```

```
            return "Esfand";
        }
        return "Invalid month number";
    }

    /**
     * Check if string is a valid date in YYYY/mm/dd format
     * It is assumed that all months have 30 days
     * @param date date in string format
     * @return true if the string is a valid date in YYYY/mm/dd format
     */
    public static boolean isValidDate(String date) {
        // TODO: Implement
        return false;
    }
}
```

## آن‌چه باید آپلود کنید

پس از اعمال تغییرات، فایل `DateUtil.java` را آپلود کنید.

# تاکسی

*تصحیح این سؤال به‌صورت خودکار است، اما کدهای ارسالی به‌صورت دستی نیز بررسی می‌شوند.*

علی اخیراً یک برنامه‌ی ساده برای یک تاکسی‌رانی نوشته است. متأسفانه او در نوشتن این برنامه دقت کافی نداشته و کدی که نوشته دارای بوهای بد است. حال، مدیریت این تاکسی‌رانی بابت کثیفی کد او اعتراض کرده، زیرا علی معمولاً در افزودن قابلیت‌های جدید به این کد دچار مشکل می‌شود. علی هم‌اکنون از شما درخواست کرده تا کد او را بازآرایی کنید.

## جزئیات پروژه

کدهای اولیه‌ی علی را از این لینک دانلود کنید.

### اینترفیس `Taxi`

این اینترفیس بیانگر نوع تاکسی است و در پروژه‌ی اولیه شامل هیچ متدی نیست. متد `BigDecimal getPrice(Point src, Point dest)` را به این اینترفیس اضافه کنید.

### کلاس `EuclideanTaxi`

این کلاس بیانگر تاکسی‌ای است که فواصل را به‌صورت خطی مستقیم طی می‌کند. در واقع، این نوع تاکسی‌ها از فاصله‌ی اقلیدسی بین مبدأ و مقصد استفاده می‌کنند.

### کلاس `ManhattanTaxi`

این کلاس بیانگر تاکسی‌ای است که فواصل را تنها به‌صورت افقی و عمودی. در واقع، این نوع تاکسی‌ها از فاصله‌ی منهتن بین مبدأ و مقصد استفاده می‌کنند.

### کلاس `TripHandler`

این کلاس شامل متد `public static int getPrice(int srcX, int srcY, int destX, int destY, Taxi taxi)` است که هزینه‌ی سفر را برحسب نقطه‌ی مبدأ و مقصد و نوع تاکسی محاسبه می‌کند. بدنه‌ی این متد به‌صورت زیر است:

```java
public static double getPrice(int srcX, int srcY, int destX, int destY, Taxi
    if (taxi instanceof ManhattanTaxi) {
        return (Math.abs(destX - srcX) + Math.abs(destY - srcY)) * 3;
    }
    if (taxi instanceof EuclideanTaxi) {
        return Math.sqrt(Math.pow(destX - srcX, 2) + Math.pow(destY - srcY, 2
    }
    return 0;
}
```

ایرادات مختلفی در کد بالا وجود دارند که با رعایت نکات زیر، باید آن‌ها را برطرف کنید:

۱. به جای استفاده از پارامترهای متعدد از نوع *primitive* ، باید از شیء پارامتر (*parameter object*) استفاده کرد تا معنای بهتری پیدا کنند و تعدادشان نیز کم شود. کلاسی با نام `Point` تعریف کنید که شامل دو پراپرتی `public final` از نوع `int` با نام‌های `x` و `y` داشته باشند. یک کانستراکتور برای این کلاس در نظر بگیرید که با استفاده از آن بتوان پراپرتی‌ها را مقداردهی کرد. در ادامه، امضای متد `getPrice` را به‌شکل `public static double getPrice(Point src, Point dest, Taxi taxi)` درآورید.

۲. هیچ‌گاه نباید از `double` برای محاسبات پولی استفاده کرد. علت این موضوع را می‌توانید در این مقاله در جاواکاپ (البته بعد از کوییز!) مطالعه کنید. می‌توان از `BigDecimal` برای محاسبات پولی استفاده کرد. بنابراین، امضای متد `getPrice` را به `public static BigDecimal getPrice(Point src, Point dest, Taxi taxi)` تغییر دهید (راهنمایی را مطالعه کنید).

۳. استفاده از `if` های متعدد برای داشتن رفتارهای مختلف برحسب نوع شیء ورودی کار خوبی نیست. بهتر است آن را با *polymorphism* جایگزین کنید. متد `BigDecimal getPrice(Point src, Point dest)` را در اینترفیس `Taxi` تعریف کرده و آن را در کلاس‌های `EuclideanTaxi` و `ManhattanTaxi` پیاده‌سازی کنید.

۴. اعداد `3` و `4` عدد جادویی (*magic number*) محسوب می‌شوند و هدف برنامه‌نویس از آن‌ها به‌طور واضح مشخص نیست. این اعداد ضرایبی هستند که در قیمت محاسبه‌شده توسط تاکسی ضرب می‌شوند و قیمت نهایی حاصل می‌شود. برای رفع این مشکل، یک پراپرتی `private static final int` با نام `PRICE_COEFFICIENT` در کلاس‌های `EuclideanTaxi` و `ManhattanTaxi` تعریف کرده و آن‌ها را براساس کد اولیه مقداردهی کنید.

**توجه:** این سؤال علاوه‌بر تصحیح خودکار، تصحیح دستی نیز دارد.

▼ راهنمایی در خصوص `BigDecimal` برای این سؤال

کلاس `BigDecimal` جاوا، یکی از کلاس‌های دوست‌داشتنی ولی کم‌تر شناخته‌شده است. این کلاس یک عدد اعشاری را به شکل دقیق (با کمک ممیز ثابت) نگه می‌دارد. در این‌صورت، مشکلات ممیز شناور مثل برابر نبودن پیش نمی‌آید. در پیاده‌سازی این کلاس، از کلاس `BigInteger` استفاده شده و به آن قابلیت ممیز اضافه شده. برای مطالعه‌ی بیش‌تر، می‌توانید از این لینک استفاده کنید. خودتان هم سرچ کنید!

برای تبدیل یک عدد به `BigDecimal` و انجام عملیات‌های ریاضی روی آن روش‌های زیر وجود دارند:

```
BigDecimal bdFromString = new BigDecimal("0.1");
BigDecimal bdFromCharArray = new BigDecimal(new char[] {'3','.','1','6','1'
BigDecimal bdlFromInt = new BigDecimal(42);
BigDecimal bdFromLong = new BigDecimal(123412345678901L);

//Operations in BigDecimal:
System.out.println(bdFromString.compareTo(bdlFromInt) < 0); //true

BigDecimal bd1 = new BigDecimal("4.0");
BigDecimal bd2 = new BigDecimal("2.0");

BigDecimal sum = bd1.add(bd2);
BigDecimal difference = bd1.subtract(bd2);
BigDecimal quotient = bd1.divide(bd2);
BigDecimal product = bd1.multiply(bd2);
BigDecimal pow = bd1.pow(5);

BigDecimal bd = new BigDecimal("1000");
// for now leave mc to be 10
MathContext mc = new MathContext(10);
BigDecimal sqrt = bd.sqrt(mc);
System.out.println(sqrt);  //31.6227766
```

نکات دیگری که باید در نظر داشته باشید:

- عملیات‌ها باید با متدهای `BigDecimal` انجام شوند، این‌که محاسبات مثلاً `pow` و `multiply` را با `double` انجام دهید و سپس از روی نتیجه `BigDecimal` بسازید، دقت کافی را ندارد و عملاً دقتی برابر `double` خواهد داشت که مناسب نیست. مثلا کد زیر دقتی بیش‌تر از `double` ندارد و مورد قبول نیست (البته شاید نمره‌ای کسب کند، ولی این نکته کلی بود).

```
return new BigDecimal(
    (Math.abs(dest.x - src.x) +
        Math.abs(dest.y - src.y)
    ) * PRICE_COEFFICIENT
);
```

- در عوض از عملوندها `BigDecimal` می‌سازیم و سپس روی آن متدهای مختلف را صدا می‌کنیم.
- شیء `BigDecimal` غیرقابل تغییر (*immutable*) است، بنابراین متدها (مثلاً `add` ) خود آن را تغییر نمی‌دهند.
- متد `pow` فقط عدد صحیح قبول می‌کند و `sqrt` نیز در جاوای ۹ اضافه شده، با توجه به نسخه جاوای ۸ کوئرا، می‌توانید از این لینک استفاده کنید.
- دقت نتیجه‌ی محاسبات `BigDecimal` شما تا ۵۰ رقم بعد از اعشار مدنظر است، بنابراین اگر بعد از رقم ۵۰اُم، محاسبات دقت کافی را نداشتند، نگران نباشید.
- نگران تنظیم `scale` برای `BigDecimal` های خود نباشید، در تست کردن این مورد در نظر گرفته شده. مثلاً `200.0000000` همان `200` در نظر گرفته می‌شود.

## آن‌چه باید آپلود کنید

پس از بازآرایی کد علی، فایل‌های زیر را بدون هیچ پوشه‌بندی‌ای زیپ کرده و آپلود کنید:

```
.
├── EuclideanTaxi.java
├── ManhattanTaxi.java
├── Point.java
├── Taxi.java
└── TripHandler.java
```