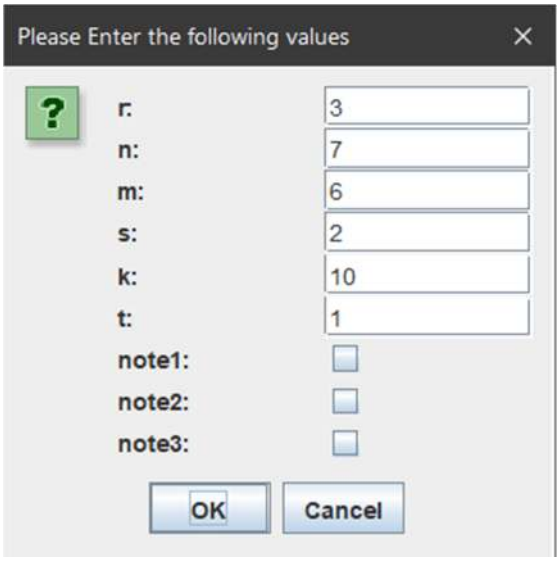
بخش ۱ – حالت شبیه سازی با ریسمان

هدف این بخش پیاده سازی پروژه با در نظر گرفتن هر موجود باغ وحش به عنوان یک ریسمان است. کلاس Zoo وظیفه نگه داری اطلاعات کلی مربوط به باغ وحش را برعهده دارد. کلاس Cell در واقع همان خانه‌های جدول یا به قفس برای Animal ها است. آرایه دو بعدی [][]Cell جدول باغ وحش را نمایش میدهد که در هر خانه آن تعدادی حیوان وجود دارند. همچنین کلاس ZooManager به عنوان کنترل کننده چرخه زندگی عمل میکند. کلاس های دیگر نیز وظایف جانبی دارند، مثلا کلاس GUI رابط گرافیکی برنامه است.

در طراحی برنامه اصل کمترین دسترسی برای اشیا تا حد امکان لحاظ شده است. همچنین متغییرهای مهم و توابع غیر بدیهی کامنت گذاری شده اند. برای اجرای برنامه کافیست Main.java را اجرا کنید.

در ابتدا برای اجرا از شما متغییر ها درخواست می شود:



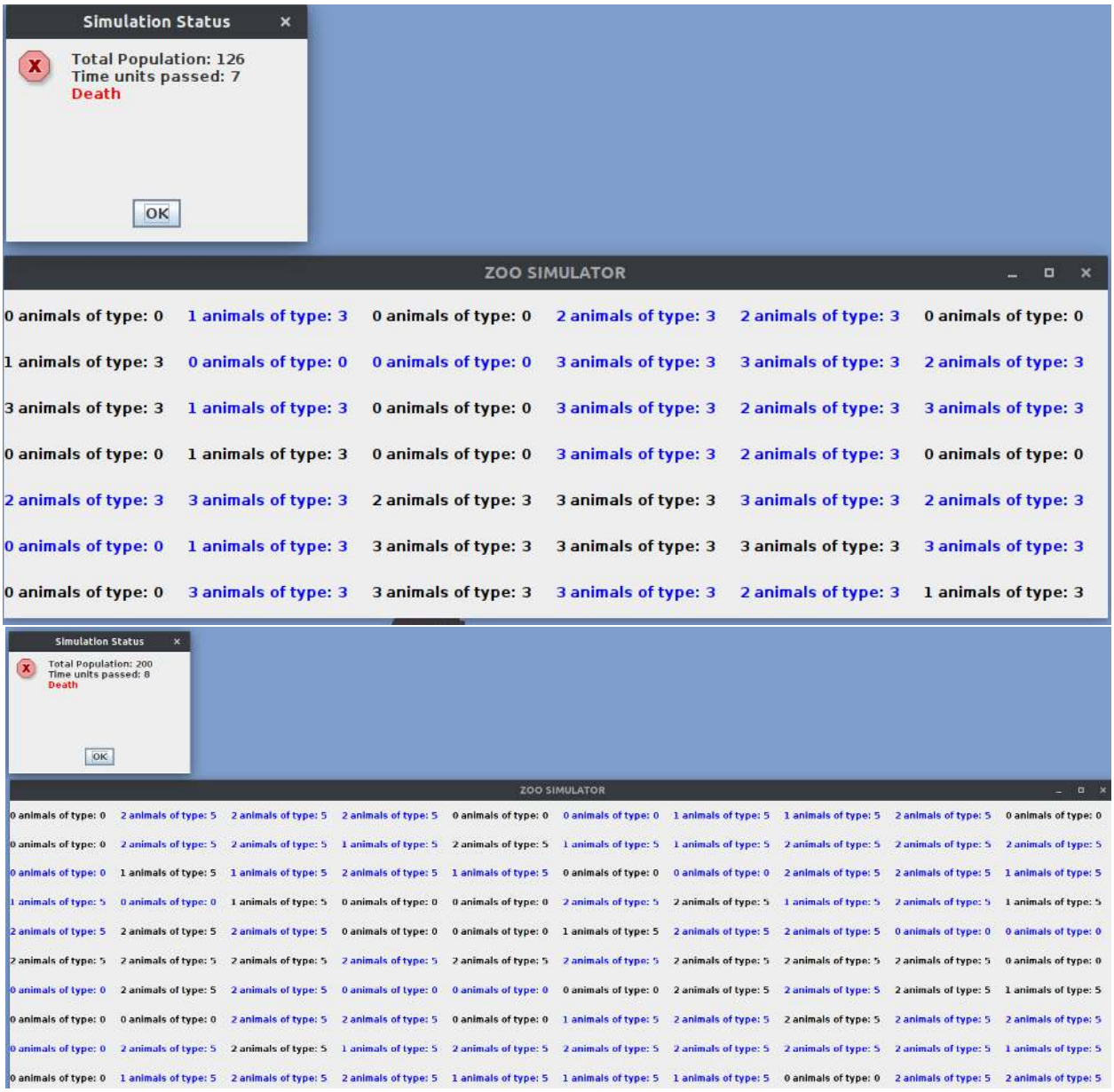
Note1 و note 2 به ترتیب تبصره های اول و دوم هستند.Note 3 شرط جدیدی است که با توجه به ابهام موجود در صورت به صورت زیر در نظر گرفته شده:

اگر note 3 = false در هر واحد زمانی (t)، همه حیوانات زاد و ولد می کنند. در غیر اینصورت حیوان نوع i در خانه ای که به اندازه i واحد زمانی از آخرین زاد و ولد (یا شروع برنامه) گذشته، زاد و ولد میکند.

دقت کنید اگر متغییر ها داری مقادیر نادرستی باشند، چیزی در صفحه نشان داده نمی شود یا با ارور مواجه می شوید.

دقت کنید که این برنامه با جاوا 1.8 (ورژن ۸) در سیستم عامل linux Ubuntu 18.04 نوشته شده و تست شده است.

اجرای برنامه به صورت زیر است:



اطلاعات هر خانه در صفحه نمایش داده می شود. در هر واحد زمانی، خانه ای که دچار تغییر شود به رنگ آبی تغییر پیدا میکند.

دقت کنید که با توجه به تصادفی بودن بعضی رویداد ها، در اجراهای متفاوت نتایج متفاوتی به دست می آید.

دقت کنید که بخش هایی از پروژه که دارای ابهام بودند، به صورت برداشت آزاد با پایبندی به شرایط و قوانین پروژه پیاده شده اند.

```
private void busyWait() {
    try {
        // adding the animal to waiting list!
        zooManager.getTotalWaitingAnimals().addAndGet( delta: 1);
        synchronized (zooManager.getObject()) {
            // waiting to be notified
            zooManager.getObject().wait();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        zooManager.getTotalWaitingAnimals().addAndGet( delta: -1);
    }

}
```

در طراحی ریسمان ها، هر Animal یک ریسمان دارد که توسط zoo manager مدیریت میشوند. Zoo سمافور باینری (mutex) ایی دارد که zoo manager با استفاده از آن تضمین می کند که چرخه حیات به ترتیب و درست انجام شود.

هر کدام از حالت تولد، زندگی و مرگ و نمایش قبل از ورود قفل را میگیرند. همچنین یک شی به صورت Objectبرای خبر دار کردن ریسمان ها طراحی شده و تابع busyWait نیز تضمین میکند که هیچ حیوانی زمان نامناسب در حرکت نیستند. همچنین خود animal ها نیز اگر چرخه در حالت life نباشد در حالت busy wait قرار میگیرند تا اینکه وضعیت عوض شود و notify شوند :

```
@Override
public void run() {
    //cycle of life!
    while (running && zoo.isOpen()) {
        System.out.println("new cycle");
        updateScreen();
        birth();
        life();
        death();
        timeUnitsPassed.addAndGet( delta: 1);
        System.out.println("cycle ended");


    }

}
```

دقت کنید که قسمتی از منطق حرکت کردن حیوانات و شکار و تولد آن ها در خود خانه ها cell پیاده شده که zoo manager صرفا دستور میدهد. مثلا zoo manager با صدا زدن breed به خانه دستور میدهد که در زمان تولد است و موجودات درون آن باید زاد و ولد کنن. برسی شرایط دیگر به عهده خود cellاست. حرکت حیوانات نیز توسط خود Animalکنترل می شود و cell اجازه ورود یا عدم ورود را می دهد. تبصره ها نیز در zoo manager و cell پیداده شده اند:

تبصره ۱:

```
private void startHunt() {
    // get cells with animals inside
    // for each cell find xy
    // for each cell get all neighbors
    // calculate xy of neighbors
    // compare cell value with neighbors value
    // if passed kill the cell!
    List<Cell> fullCells;
    if (zoo.isNote1()) {
        for (int r = 1; r < zoo.getKinds() + 1; r++) {
            fullCells = getTotalNonEmptyCells(r);
            huntCells(fullCells);
        }
    } else {
        fullCells = getTotalNonEmptyCells(kinds-1);
        huntCells(fullCells);
    }

}
```

تبصره ۲:

```
private boolean isCellValidNeighbor(int i, int j, Cell cell) {
    int y = cell.getY();
    int x = cell.getX();
    if (x == 0 && j < x) {
        return false;
    }
    if (x == zoo.getNumColumns() - 1 && j > x) {
        return false;
    }
    if (y == 0 && i < y) {
        return false;
    }
    if (y == zoo.getNumRows() - 1 && i > y) {
        return false;
    }
    if (zoo.isNote2()) {
        // statement 1 pass
        if (cell.getCellAnimalKind() > zoo.getKinds() / 2) {
            Cell neighbor = zoo.getTable()[i][j];
            // statement 2 pass
            if (neighbor.getCellAnimalKind() < cell.getCellAnimalKind()) {
                if ((j == x - 1 && i == y - 1) ||
                        (j == x + 1 && i == y + 1) ||
                        (j == x + 1 && i == y - 1) ||
                        (j == x - 1 && i == y + 1)) {
                    return false;
                }
            }

        }
    }
    return cell.getX() != j || cell.getY() != i;
```

بخش ۲– حالت شبیه سازی با پردازه

این قسمت به طور کامل پیاده سازی نشده اما طراحی به این صورت است که هر پردازه با استفاده از input and output strems با پردازه اصلی ارتباط برقرار میکند. کلاس zoo manager در اینجا نیز مثل قبل وظیفه کنترل را دارد. از طرفی برای هر پردازه animal یک ریسمان anima thread وجود دارد که وظیفه آن صرفا دریافت و ارسال اطلاعات است. اطلاعات توسط هر پردازه حیوان ارسال میشود و ریسمان متناظر این اطلاعات را دریافت و به zoo manager ارسال میکند. برای همگام سازی پردازه ها وجود همچین روشی الزامی است. پردازه اصلی هرگاه از تمام پردازه های دیگر اطلاعات تاییدی را گرفت حق دارد به مرحله بعدی چرخه زندگی برود.