شمارش جزیره ها

- محدودیت زمان: ۴ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: آسان
- طراح: مهدی افشاری

شرح مسئله

نقشه‌ای به شما داده می‌شود که به صورت یک جدول (grid) دو بعدی نمایش داده شده است. در این نقشه، برخی مناطق خشکی و برخی آب هستند. وظیفه شما این است که برنامه‌ای بنویسید که تعداد جزیره‌های موجود در این نقشه را شمارش کند.

قوانین

- این جدول شامل دو نوع کاراکتر است: 'X' نمایانگر خشکی و '.' نمایانگر آب است.
- یک جزیره از یک یا چند خانه 'X' تشکیل شده است که به صورت **افقی** یا **عمودی** به یکدیگر متصل هستند.
- خانه‌های 'X' که فقط به صورت **مورب** با یکدیگر در تماس هستند، بخشی از یک جزیره واحد محسوب **نمی‌شوند**.

پس خانه های 'X' برای تشکیل جزیره تنها میتوانند از بالا، پایین، چپ و راست به همدیگر متصل شوند!

ورودی

خط اول شامل دو عدد صحیح M (تعداد سطرها) و N (تعداد ستون‌ها) است که با یک فاصله از هم جدا شده‌اند.

در M خط بعدی که هر کدام شامل N کاراکتر خواهند بود، یک سطر از نقشه آورده شده است. کاراکترها در هر خط **بدون فاصله** از هم آمده اند.

$$1 \leq M, N \leq 200$$

کاراکترهای جدول فقط می‌توانند '.' یا 'X' باشند. (فقط 'X' بزرگ داریم و جدول شامل 'x' کوچک نمیباشد)

راهنمایی

برای جدا کردن کاراکتر ها در هر خط میتوانید از دستور زیر استفاده کنید:

```
String nextLine = scanner.nextLine();
String[] parts = nextLine.split("");
```

خروجی

برنامه باید یک عدد صحیح را به عنوان خروجی چاپ کند: این عدد، تعداد کل جزیره‌ها در نقشه است.

مثال

ورودی نمونه

```
5 5
X.XX.
X....
...X.
.X..X
XX...
```

خروجی نمونه

```
5
```

توضیح مثال

۱. جزیره عمودی در خانه‌های [0][0], [0][1]

۲. جزیره افقی در خانه‌های [2][0], [3][0]

۳. جزیره L مانند در خانه‌های [0][4], [1][4], [1][3]

۴. جزیره تنها در خانه [3][2]

۵. جزیره تنها در خانه [4][3]

NeoBot Security System

- سطح: متوسط
- طراح: محسن نوروزی

فایل اولیه پروژه را از این لینک دریافت کنید.

در سال‌های نه‌چندان دور، انسان‌ها سامانه‌ای ساختند تا محافظ هوش‌های مصنوعی باشد . یک نگهبان سایبری میان دنیای دیجیتال و واقعیت. نامش **NeoBot AI Security System** بود.

سیستمی که قضاوت می‌کرد، می‌پذیرفت، یا رد می‌کرد...

و هیچ موجود دیجیتالی بدون تأیید آن نمی‌توانست وارد شبکه‌های جهانی شود.

کلاس NeoRobot

هر **NeoRobot** موجودی مستقل است.

آن‌ها فقط ماشین نیستند؛ بلکه ترکیبی از منطق، حافظه و چیزی شبیه به "خودآگاهی مصنوعی" دارند.

برای هر ربات سه ویژگی اصلی تعیین شد:

۱. **نام (Name) :** هویت مستقل دیجیتالی(نام یونیکی که هر ربات دارد).
۲. **سطح هوش (Intelligence Level) :** عددی میان 0 تا 100، نمایانگر میزان درک ربات از جهان.
۳. **کد عصبی (Intelligence Code) :** زنجیره‌ای از نشانه‌ها که الگوی عصبی و مرکز تصمیم‌گیری مغز مصنوعی را تعریف می‌کند(مهم ترین بخش در ساخت ربات).

نکته :حواستان به کامل کردن constructor و getter ها باشد!

هر ربات به شکل رشته‌ای از داده در حافظه متولد می‌شود، اما فقط زمانی زنده می‌ماند که توسط سامانهٔ امنیتی تأیید شود.

نمونه‌ای از تولد یک ربات:

```
NeoRobot r1 = new NeoRobot("Alpha", 72, "Neo@coreMatrix8");
```

کلاس NeoSecuritySystem

در قلب آزمایشگاه، سیستمی وجود دارد با نام **NeoSecuritySystem**:

ویژگی‌های اساسی:

```
private NeoRobot[] robots = new NeoRobot[50];
```

آرایه‌ای برای نگه داری دیتای ربات ها.

```
private int robotCount = 0;
```

برای نگه داشتن تعداد فعلی ربات ها (در متد ها باید در صورت نیاز آپدیت شود).

متد **registerRobot :**

نگهبانی که با دقت تمام بررسی می‌کند هر موجود جدید واجد شرایط ورود هست یا نه. وظایف آن:

- بررسی تعداد ربات ها (نباید بیشتر از ظرفیت حافظه که 50 تا است بشوند). در صورتی که تعداد بیشتر از 50 تا شد `MaxRobotException` پرتاب کند.
- بررسی تکراری نبودن ربات‌ها(فقط بر اساس نام ربات (Name) بررسی شود). اگر رباتی تکراری بود باید `RobotAlreadyExistsException` پرتاب کند.
- ارزیابی سطح هوش(Intelligence Level). اگر سطح هوش ربات از حد مورد نظر کمتر بود(سطح مورد نظر 50 است). باید `InvalidIntelligenceLevelException` پرتاب کند.
- تجزیه‌وتحلیل کد عصبی (Intelligence Code) با الگوریتم **NeoValidator**. خروجی این بخش (اگر اکسپشن باشد متن آن) باید در یک string ذخیره شود و در خروجی نمایش داده شود.

اگر تمام موارد بالا بدون مشکل بودند ، ربات ما به لیست تیم امنیت دیجیتال اضافه می‌شود و متن زیر برای آن نمایش داده می‌شود:

```
Robot Alpha registered successfully! AI code successfully validated!
```

نکته : در مثال بالا نام ربات `Alpha` در نظر گرفته شده است .

متد **`findRobot` :**

که وظیفه آن گشتن در حافظه بر اساس نام ربات و برگرداندن اطلاعات آن است. اگر ربات مورد نظر پیدا نشد یک `RobotNotFoundException` پرتاب شود.

متد **`showAllRobots` :**

وظیفه آن برگرداندن تمام ربات های داخل آرایه است. اگر هیچ رباتی در حافظه نبود متن زیر نمایش داده شود :

```
there is no Robot !
```

متد **`deleteRobot` :**

وظیفه این متد پاک کردن ربات از حافظه و شیفت دادن ربات ها (یک واحد) به سمت چپ است(در صورت نیاز). اگر ربات مورد نظر پیدا نشد یک `RobotNotFoundException` پرتاب شود.

کلاس `NeoValidator`

کلاس `NeoValidator` شامل یک متد ثابت به نام `validateAI(String code)` است؛ متدی که کارش **تجزیه، تحلیل و غربالگری کد هوش** هر ربات است. هر بار که یک ربات تلاش می‌کند در `NeoSecuritySystem` ثبت شود، این متد فراخوانی می‌شود و رشته‌ی هوش او را با دقت بررسی می‌کند.

مراحل اعتبارسنجی هوش (AI Code Validation)

۱. **بررسی وجود ویروس یا نفوذگر**

اگر کد شامل واژه‌های "hack" یا "virus" باشد، فوراً **استثنای `AIHackAttemptException`** پرتاب می‌شود؛ یعنی مغز ربات آلوده است.

۲. **ارزیابی قدرت کد**

اگر طول کد کمتر از ۱۲ کاراکتر باشد، **WeakIntelligenceCodeException** فعال می‌شود — مغز خیلی کوچک و کم‌دانش است!

۳. **تشخیص الگوی عصبی (Neural Pattern)**

باید یکی از نمادهای @ , # , یا ! در کد وجود داشته باشد؛ در غیر این صورت، استثنای **MissingNeuralPatternException** نشان می‌دهد الگوی ارتباط عصبی ناقص است.

۴. **بررسی هسته‌ی عددی (Numeric Core)**

مغز هر ربات باید حاوی عدد باشد تا بخش منطقی فعال شود. در غیر این صورت خطای **MissingNumericCoreException** رخ می‌دهد.

۵. **اتصال حسی (Sensory Link)**

وجود **حروف بزرگ (A–Z)** نشانه‌ی حس بینایی و شنیداری هوش است. نبود آن سبب پرتاب **MissingSensoryLinkException** می‌شود.

۶. **منطق درونی (Logic Matrix)**

وجود **حروف کوچک (a–z)** لازمه‌ی عقل و استدلال در ربات است. اگر نباشد، **MissingLogicMatrixException** شناسایی می‌شود.

در صورت موفقیت تمام بررسی‌ها، خروجی این متد جمله‌ای است که ثبت موفقیت مغز ربات را اعلام می‌کند:

```
AI code successfully validated!
```

ورودی نمونه

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

        System.out.println("=== NeoBot AI Security System ===\n");
        NeoSecuritySystem system = new NeoSecuritySystem();

        NeoRobot[] robots = {
                new NeoRobot("Alpha", 80, "Neo@coreMatrix8"),         // Vali
                new NeoRobot("Beta", 60, "Beta#LogicCore77"),         // Vali
                new NeoRobot("HackerBot", 90, "hack@engine9"),        // AIHa
                new NeoRobot("Shorty", 70, "Neo@9"),                  // Weak
                new NeoRobot("NoPattern", 70, "NeoBrainMatrix9"),     // Miss

```

```
                new NeoRobot("NoNumber", 70, "Neo@coreMatrix"),       // Miss
                new NeoRobot("NoUppercase", 70, "@corematrix8"),      // Miss
                new NeoRobot("NoLowercase", 70, "NEO@COREMATRIX8"),   // Miss
                new NeoRobot("SimpleBot", 35, "Neo@coreMatrix9"),     // Inva
                new NeoRobot("Omega", 95, "Omega@neuronCore777")      // Vali
        };

        // Try to register them
        for (NeoRobot r : robots) {
            try {
                String result = system.registerRobot(r);
                System.out.println(result);
            } catch (InvalidIntelligenceLevelException |
                     RobotAlreadyExistsException |
                     AIHackAttemptException |
                     WeakIntelligenceCodeException |
                     MissingNeuralPatternException |
                     MissingNumericCoreException |
                     MissingSensoryLinkException |
                     MissingLogicMatrixException e) {
                System.out.println("Error registering " + r.getName() + ": ["
                        e.getClass().getSimpleName() + "] " + e.getMessage())
            }
        }

        System.out.println("\n--- Registered Robots ---");
        system.showAllRobots();

        System.out.println("\n--- Searching for Omega ---");
        NeoRobot found = system.findRobot("Omega");
        if (found != null)
            System.out.println("Found: " + found);
        else
            System.out.println("Robot not found.");
    }
}
```

خروجی نمونه

```
=== NeoBot AI Security System ===


```

```
Robot Alpha registered successfully! AI code successfully validated!
Robot Beta registered successfully! AI code successfully validated!
Error registering HackerBot: [AIHackAttemptException] AI Hack attempt detecte
Error registering Shorty: [WeakIntelligenceCodeException] Weak intelligence c
Error registering NoPattern: [MissingNeuralPatternException] Missing neural p
Error registering NoNumber: [MissingNumericCoreException] Missing numeric cor
Error registering NoUppercase: [MissingSensoryLinkException] Missing sensory
Error registering NoLowercase: [MissingLogicMatrixException] Missing logic ma
Error registering SimpleBot: [InvalidIntelligenceLevelException] Invalid inte
Robot Omega registered successfully! AI code successfully validated!

--- Registered Robots ---
1. NeoRobot{name='Alpha', level=80, code='Neo@coreMatrix8'}
2. NeoRobot{name='Beta', level=60, code='Beta#LogicCore77'}
3. NeoRobot{name='Omega', level=95, code='Omega@neuronCore777'}

--- Searching for Omega ---
Found: NeoRobot{name='Omega', level=95, code='Omega@neuronCore777'}
```

Cafe Management System

- سطح: متوسط
- طراح: آیسودا فضلی خانی

می‌خواهیم با استفاده از MVC و Singleton، سیستمی برای مدیریت سفارشات و منوی یک کافی شاپ طراحی کنیم.

پروژه اولیه را می‌توانید از این لینک دانلود کنید.

جزئیات پروژه

▼ ساختار فایل پروژه

```
cafe-manager
    ├── CafeManager.java
    ├── MenuItem.java
    ├── MenuView.java
    ├── Order.java
```

کلاس `MenuItem`

اشیاء ساخته شده از این کلاس بیانگر هرکدام از آیتم‌های موجود در منوی کافه هستند.

```java
public class MenuItem {
    private String name;
    private double price;
    private String category;

    public MenuItem(String name, double price, String category) {
        // TODO: Implement
    }

    public String getName() {
        // TODO: Implement
```

```
    }

    public double getPrice() {
        // TODO: Implement
    }

    public String getCategory() {
        // TODO: Implement
    }
}
```

پراپرتی‌ها:

۱. `name` : نام محصول از نوع `String` .

۲. `price` : قیمت محصول از نوع `double` .

۳. `category` : دسته بندی محصول از نوع `String` که می‌تواند شامل `tea` ، `food` و `coffee` و... باشد.

کلاس `Order`

اشیاء ساخته شده از این کلاس بیانگر هرکدام از سفارشات هستند که شامل اطلاعات سفارش نظیر شماره سفارش، نام مشتری، مجموع قیمت و... هستند.

```
1  public class Order {
2      private String orderId;
3      private String customerName;
4      private List<MenuItem> items;
5      private double totalAmount;
6      private String status;
7
8      public Order(String customerName) {
9          // TODO: Implement
10     }
11
12     public void addItem(MenuItem item) {
13         // TODO: Implement
14     }
15
16     public void completeOrder() {
```

```
        // TODO: Implement
    }

    public String getOrderId() {
        // TODO: Implement
    }

    public String getCustomerName() {
        // TODO: Implement
    }

    public List<MenuItem> getItems() {
       // TODO: Implement
    }

    public double getTotalAmount() {
       // TODO: Implement
    }
    public String getStatus() {
       // TODO: Implement
    }
}
```

پراپرتی‌ها:

۱. `orderId` : شماره سفارش از نوع `String` .

۲. `customerName` : نام مشتری از نوع `String` .

۳. `items` : لیست محصولات سفارش داده شده از نوع `List<MenuItem>` .

۴. `totalAmount` : جمع قیمت تمام آیتم‌های انتخاب شده از نوع `double` که در ابتدا **صفر** می‌باشد.

۵. `status` : نشان‌دهنده وضعیت فعلی سفارش از نوع `String` که می‌تواند **pending** و یا **completed** باشد.

متدها:

۱. `addItem` : این متد یک `MenuItem` را بعنوان ورودی می‌گیرد و آن آیتم را به سفارش مشتری اضافه می‌کند.

۲. `completeOrder` : با فراخوانی این متد، وضعیت سفارش از pending به completed تغییر می‌کند.

در کانستراکتور این کلاس، وضعیت فعلی pending و totalAmount صفر مقداردهی می‌شود و فرمت orderId به شکل زیر می‌باشد:

`ORD + number`

کلاس CafeManager

مدیریت اصلی سیستم به عهده این کلاس است که باید به شکل Singleton پیاده‌سازی شود. برای دریافت تنها شی این کلاس باید متد getObject به‌صورت static پیاده‌سازی شود.

پراپرتی‌ها:

۱. menuItems : لیست تمامی محصولات موجود در منوی کافه از نوع List<MenuItem> .
۲. orders : لیست تمامی سفارشات ثبت شده در سیستم از نوع List<Order> .
۳. totalRevenue : مجموع درآمد کافه از همه سفارشات تکمیل شده از نوع double که در ابتدا **صفر** می‌باشد.
۴. view : از نوع MenuView و برای نمایش اطلاعات به کاربر.

متدها:

۱. addMenuItem : این متد یک ورودی از نوع MenuItem می‌گیرد و آن را به منوی کافه اضافه می‌کند. نکته: اگر نام آیتم تکراری باشد و قبلا در منو وجود داشته باشد فقط باید پیام Error: This item already exists نمایش داده شود.
۲. displayMenu : با فراخوانی این متد، منوی کافه نمایش داده می‌شود (از کلاس MenuView استفاده کنید).
۳. createOrder : ورودی این متد یک String بعنوان نام سفارش‌دهنده است و یک شی جدید از نوع Order می‌سازد و آن را به لیست سفارشات کافه اضافه می‌کند و شی ساخته شده را برمی‌گرداند.
۴. addItemToOrder : این متد یک ورودی از نوع Order و یک ورودی از نوع String می‌گیرد که نام یک آیتم است. اگر آیتم داخل منو موجود باشد باید به سفارش اضافه شود و در غیر اینصورت پیام خطای Item not found! نمایش داده می‌شود.

۵. `completeOrder` : این متد با دریافت یک `Order` بعنوان ورودی، وضعیت آن را به completed تغییر می‌دهد و سپس مبلغ سفارش را به `totalRevenue` اضافه می‌کند و در نهایت پیام زیر را نمایش می‌دهد:

`order id` + completed!

۶. `showOrder` : با دریافت یک `Order` ، جزئیات آن را نمایش می‌دهد.

۷. `findItem` : تمام آیتم‌های موجود در منو را بررسی می‌کند و آیتمی که نامش با ورودی متد مطابقت دارد را برمی‌گرداند.

کلاس `MenuView`

این کلاس نقش view در دیزاین را دارد.

متدها:

۱. `showMenu` : این متد با دریافت لیست آیتم‌های موجود در منو، منو را نمایش می‌دهد. نمایش منو به این شکل است که دسته‌بندی محصولات به ترتیب حروف الفبا نمایش داده می‌شود و داخل هردسته نیز آیتم‌های آن دسته هم به ترتیب حروف الفبا نمایش داده می‌شوند. برای مثال:

```
cofee:
1.americano $3.75
2.cappuccino $4.50
3.espresso $3.50
4.latte $4.75
food:
1.croissant $2.50
2.muffin $3.25
3.sandwich $6.00
tea:
1.black tea $2.25
2.green tea $2.50
```

۲. `showOrder` : این متد با دریافت یک سفارش بعنوان ورودی، جزئیات آن را نمایش می‌دهد. در خط اول شماره سفارش، در خط دوم نام مشتری، در خط بعدی وضعیت سفارش و در انتها مبلغ سفارش

نمایش داده می‌شود. مثال:

```
order:ORD1700000000123
customer:Aysuda
status:pending
total:$7.25
```

۳. showMessage : این متد نیاز به پیاده‌سازی ندارد و برای نمایش پیام‌ها می‌باشد.

ورودی نمونه

```
System.out.println("=== Cafe Manager ===");
CafeManager manager = CafeManager.getObject();

System.out.println("\nAdding items to menu:");
manager.addMenuItem(new MenuItem("latte", 4.75, "coffee"));
manager.addMenuItem(new MenuItem("espresso", 3.50, "coffee"));
manager.addMenuItem(new MenuItem("cappuccino", 4.50, "coffee"));
manager.addMenuItem(new MenuItem("green tea", 2.50, "tea"));
manager.addMenuItem(new MenuItem("black tea", 2.25, "tea"));
manager.addMenuItem(new MenuItem("sandwich", 6.00, "food"));
manager.addMenuItem(new MenuItem("croissant", 2.50, "food"));
manager.addMenuItem(new MenuItem("muffin", 3.25, "food"));

System.out.println("\nTesting duplicate item:");
manager.addMenuItem(new MenuItem("latte", 5.00, "coffee"));

System.out.println("\nDisplaying menu:");
manager.displayMenu();

System.out.println("\nCreating orders:");
Order order1 = manager.createOrder("Aysuda Fazlikhani");
Order order2 = manager.createOrder("Diana Amiri");
System.out.println("Order 1 created: " + order1.getOrderId());
System.out.println("Order 2 created: " + order2.getOrderId());

System.out.println("\nAdding items to first order:");
manager.addItemToOrder(order1, "latte");
manager.addItemToOrder(order1, "croissant");
manager.addItemToOrder(order1, "green tea");
```

```
System.out.println("\nTesting non-existent item:");
manager.addItemToOrder(order1, "pizza");

System.out.println("\nShowing order details:");
manager.showOrder(order1);

System.out.println("\nAdding items to second order:");
manager.addItemToOrder(order2, "espresso");
manager.addItemToOrder(order2, "muffin");

System.out.println("\nShowing second order details:");
manager.showOrder(order2);

System.out.println("\nCompleting orders:");
manager.completeOrder(order1);
manager.completeOrder(order2);

System.out.println("\nTotal revenue: $" + manager.getTotalRevenue());
```

خروجی نمونه

```
=== Cafe Manager ===

Adding items to menu:

Testing duplicate item:
Error: This item already exists

Displaying menu:
coffee:
1.cappuccino $4.5
2.espresso $3.5
3.latte $4.75
food:
4.croissant $2.5
5.muffin $3.25
6.sandwich $6.0
tea:
7.black tea $2.25
8.green tea $2.5
```

```
Creating orders:
Order 1 created: ORD1762196136260
Order 2 created: ORD1762196136263

Adding items to first order:

Testing non-existent item:
Item not found!

Showing order details:
order:ORD1762196136260
customer:Aysuda Fazlikhani
status:pending
total:$9.75

Adding items to second order:

Showing second order details:
order:ORD1762196136263
customer:Diana Amiri
status:pending
total:$6.75

Completing orders:
ORD1762196136260 completed!
ORD1762196136263 completed!

Total revenue: $16.5
```

دعوای قطرها

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- سطح: سخت
- طراح: سید محمد حسینی

ورودی

ورودی ابتدا شامل یک عدد n است که نمایانگر ابعاد یک ماتریس دو بعدی به اندازه $n * n$ می‌باشد. سپس عدد k ورودی گرفته می‌شود. سپس تک تک اعضای ماتریس را با ورودی از کاربر دریافت می‌کنید(اگر اسم ماتریس ما mat باشد شما در ابتدا عنصر $mat[0][0]$ را ورودی گرفته و سپس $mat[0][1]$ را ورودی گرفته و به همین شکل ادامه می‌دهید تا عنصر $mat[0][numberOfColumn - 1]$ را ورودی بگیرید و سپس به سراغ سطر دوم بروید).

خروجی

در خروجی شما باید بیشترین اختلاف بین جمع عناصر روی قطر اصلی و فرعی ماتریس $k * k$ که در داخل ماتریس $n * n$ قرار دارند چاپ کنید.

نکته: شما باید تمام ماتریس‌های ممکن با ابعاد $k * k$ را در نظر گرفته و بیشترین مقدار اختلاف بین مجموع مقادیر روی قطر اصلی و قطر فرعی را برگردانید.

مثال

ورودی نمونه ۱

```
4 2
1 2 3 4
5 6 7 8
9 1 2 3
4 5 6 7
```

خروجی نمونه ۱

```
9
```

توضیح: در این نمونه ورودی با توجه به مقدار ورودی n، ما باید با یک ماتریس $4 * 4$ کار کنیم. مقدار k برابر 2 است و به این معناست که باید تمام ماتریس‌های $2 * 2$ زیر مجموعه ماتریس اصلی را در نظر بگیریم. زیر مجموعه‌های این ماتریس به ابعاد $2 * 2$ به صورت زیر می‌باشند:

```
first submatrix:
1 2
5 6
output for this matrix: (1 + 6) - (2 + 5) = 0

second submatrix:
3 4
7 8
output for this matrix: (3 + 8) - (4 + 7) = 0

third submatrix:
5 6
9 1
output for this matrix = (5 + 1) - (9 + 6) = -9
.
.
.
main submatrix that has max difference:
9 1
4 5
output for this submatrix: (9 + 5) - (1 + 4) = 9
```

نکته: بین تمام خروجی‌های ممکن از ماتریس های با ابعاد $2 * 2$، عدد 9 بیشترین مقدار است و به عنوان

خروجی چاپ می‌شود.

ورودی نمونه ۲

```
5 3
2 1 -3 4 5
-1 3 2 1 0
4 2 -5 6 1
3 1 2 7 4
-2 4 3 2 1
```

خروجی نمونه ۲

```
14
```

نکته: در این نمونه ورودی ماتریسی که باعث بیشترین اختلاف شده‌است:

```
4 2 -5
3 1 2
-2 4 3
output for this matrix: (4 + 1 + 3) - (-5 + 1 + (-2)) = 14
```

می‌باشد.