مدیریت رستوران

فایل اولیه‌ی تمرین را می‌توانید از این لینک دانلود کنید.

در این تمرین، هدف طراحی مجموعه ای از کلاس های شی گرا برای مدیریت سفارشات رستوران است.

ساختار پروژه

```
src.zip/
└─ main/
    └─ java/
        └─ c4/
            ├─ Address.java
            ├─ Customer.java
            ├─ Food.java
            ├─ Invoice.java
            └─ Item.java
```

کلاس‌ها و مشخصات دقیق

▼ **کلاس `Address`**

کلاس `Address` نمایانگر موقعیت مکانی یک آدرس است و شامل سه ویژگی `latitude` (عرض جغرافیایی)، `longitude` (طول جغرافیایی) و `written_address` (آدرس نوشتاری) می‌باشد. این کلاس باید به گونه‌ای پیاده‌سازی شود که تمام فیلدها از خارج کلاس قابل دسترسی باشند. سازنده‌ی کلاس باید از ساخته‌شدن شیء بدون مقادیر معتبر جلوگیری کند؛ به این معنا که اگر مقدار `written_address` برابر `null` یا رشته‌ی خالی باشد، باید خطای `IllegalArgumentException` پرتاب شود. همچنین مقدار `latitude` باید در بازه‌ی ۹۰- تا ۹۰ و مقدار `longitude` در بازه‌ی ۱۸۰- تا ۱۸۰ قرار داشته باشند، در غیر این صورت نیز باید

`IllegalArgumentException` پرتاب شود.

کلاس باید دارای متدی به نام `distance_from(Address ad)` باشد که یک شیء `Address` دیگر را به عنوان ورودی دریافت کرده و فاصله‌ی این دو نقطه را بر اساس طول و عرض جغرافیایی و با استفاده از فرمول زیر بر حسب کیلومتر محاسبه کند:

$$\theta = lo_1 - lo_2$$

$$d = \sin(\mathrm{rad}(la_1)) \cdot \sin(\mathrm{rad}(la_2)) + \cos(\mathrm{rad}(la_1)) \cdot \cos(\mathrm{rad}(la_2)) \cdot \cos(\mathrm{rad}(\theta))$$

$$\mathrm{distance} = \mathrm{degree}(\cos^{-1}(d)) \times 60 \times 1.1515 \times 1.609344$$

فرض کنید `A` و `B` دو شیء از کلاس `Address` باشند. در این متد، فاصله‌ی بین یک آدرس با خودش باید صفر باشد (`A.distance_from(A) == 0`)، فاصله‌ی بین دو آدرس متفاوت باید عددی مثبت باشد (`A.distance_from(B) > 0`)، و فاصله باید خاصیت تقارن داشته باشد به طوری که `A.distance_from(B) == B.distance_from(A)` برقرار باشد. همچنین آدرس‌هایی با مختصات متفاوت باید فاصله‌های متفاوتی تولید کنند تا از بازگرداندن مقدار ثابت جلوگیری شود. خروجی این متد باید از نوع `double` باشد و در پیاده‌سازی آن باید از تبدیل صحیح زاویه‌ها به رادیان و برعکس اطمینان حاصل شود.

▼ کلاس `Customer`

اشیای این کلاس نماینده‌ی مشتری‌ها در سیستم مدیریت سفارشات رستوران هستند. هر شی از این کلاس دارای یک شماره مشتری `(Customer Number)` است که یک عدد طبیعی مثبت و حداکثر ۵ رقمی بوده و پس از ساخته شدن شیء غیرقابل تغییر `(immutable)` است. این شماره باید به‌صورت خودکار هنگام ساخته شدن هر مشتری تنظیم شود و برای هر مشتری جدید مقدار متفاوتی داشته باشد. می‌توانید فرض کنید که تعداد کل مشتری‌ها کمتر از ۱۰۰,۰۰۰ نفر است. این مقدار فقط از

طریق متد getCustomerNumber قابل دسترسی است و هیچ setter ای برای آن وجود ندارد. هر مشتری علاوه بر شماره، دارای یک نام (name) از نوع String و یک آدرس (address) از نوع کلاس Address است که در بخش قبل تعریف شده است. مقدار نام و آدرس باید از طریق متدهای getName ، setName ، getAddress و setAddress قابل دسترسی باشند. کلاس باید یک سازنده (constructor) داشته باشد که با استفاده از نام و آدرس، یک شیء جدید از نوع Customer بسازد. در هنگام ساخت شیء (constructor) و نیز در متدهای setter باید از دادن ورودی‌های خالی یا null جلوگیری شود. به عبارت دیگر، اگر نام مشتری null یا رشته‌ی خالی باشد باید یک IllegalArgumentException پرتاب شود و اگر آدرس مشتری null باشد نیز باید IllegalArgumentException پرتاب شود.

▼ کلاس Food

کلاس Food نمایانگر یک غذای موجود در منوی رستوران است. در این کلاس باید یک متغیر کلاس به نام menu از نوع ArrayList<Food> تعریف شود تا تمام غذاهای ایجادشده را نگهداری کند، و هر بار که شیء جدیدی از نوع Food ساخته می‌شود، به‌صورت خودکار به این لیست اضافه گردد. هر غذا دارای دو ویژگی name از نوع String و price از نوع int است که باید در سازنده مقداردهی شوند و پس از ساخت شیء دیگر قابل تغییر نباشند (هیچ متد setter وجود نداشته باشد). نام غذا نباید null یا رشته‌ی خالی باشد و قیمت آن نیز نباید منفی باشد؛ در غیر این صورت باید IllegalArgumentException پرتاب شود. برای دسترسی به مقادیر، تنها متدهای getName ، getPrice و متد static به نام getMenu تعریف می‌شوند.

▼ کلاس Item

هر شیء از کلاس Item نشان‌دهنده‌ی یک آیتم از سفارش مشتری است. هر آیتم شامل سه ویژگی (attribute) زیر است: food از نوع Food که نشان‌دهنده‌ی غذای انتخاب‌شده است، count که یک عدد طبیعی بوده و تعداد آن غذای خاص در سفارش را نشان می‌دهد، و description از نوع String که توضیحات اضافی در مورد آیتم سفارش را نگهداری می‌کند. هر

سه ویژگی باید در سازنده (constructor) مقداردهی شوند. در زمان ساخت شیء، باید از ورود مقادیر نامعتبر جلوگیری شود: اگر مقدار food برابر با null باشد، باید یک IllegalArgumentException پرتاب شود؛ اگر مقدار count صفر یا منفی باشد، باید IllegalArgumentException پرتاب شود؛ مقدار description می‌تواند رشته‌ی خالی باشد، اما نباید null باشد. پس از ساخته شدن شیء، مقادیر این ویژگی‌ها تنها از طریق متدهای getFood ، getCount و getDescription قابل خواندن هستند و نباید قابلیت تغییر (setter) داشته باشند.

▼ کلاس Invoice

کلاس Invoice نشان‌دهنده‌ی یک فاکتور سفارش است. هر فاکتور دارای ویژگی‌ای به نام state است که وضعیت سفارش را مشخص می‌کند. اگر مقدار آن ۱- باشد، یعنی سفارش در حال ثبت است؛ اگر ۰ باشد، سفارش در صف آماده‌سازی است؛ اگر ۱ باشد، سفارش در صف ارسال است؛ و اگر ۲ باشد، سفارش به دست مشتری رسیده است. هنگام ساخت یک فاکتور جدید با یک مشتری، مقدار state باید به طور خودکار ۱- تنظیم شود. ان ویژگی باید private تعریف شودو ازطریق تابع getState قابل دسترسی باشد.

هر فاکتور باید شامل یک شیء از نوع Customer باشد که مشتری مربوط به سفارش را نشان می‌دهد (غیرقابل تغییر و فقط از طریق متد getCustomer قابل خواندن است). همچنین فاکتور دارای یک ویژگی زمانی (submitTime) از نوع LocalDateTime است که در زمان ساخته شدن شیء به طور خودکار مقداردهی می‌شود و از طریق متد getSubmitTime قابل دسترسی است.

این کلاس باید شامل یک لیست از آیتم‌های سفارش (<ArrayList<Item) باشد که فقط از طریق متد getItems قابل خواندن است. این متد باید یک کپی از لیست آیتم‌ها را بازگرداند تا از تغییر مستقیم آن توسط سایر بخش‌های برنامه جلوگیری شود.

برای افزودن آیتم به فاکتور از متد addItem(Item item) استفاده می‌شود. این متد تنها زمانی باید کار کند که سفارش در حال ثبت (state == -1) باشد؛ در غیر این صورت باید false

برگرداند. اگر آیتم با موفقیت اضافه شد، متد باید `true` برگرداند. همچنین اگر ورودی `item` برابر `null` باشد باید `IllegalArgumentException` پرتاب شود.

برای حذف آیتم از فاکتور از متد `removeItem(Item item)` استفاده می‌شود. این متد فقط زمانی باید فعال باشد که سفارش در حال ثبت باشد. در این حالت، اگر آیتمی با غذای مشابه در لیست وجود داشته باشد، باید حذف شده و مقدار `true` بازگردانده شود. در غیر این صورت یا در حالت‌های دیگر، باید `false` بازگردانده شود.

متد `nextStage` وظیفه دارد وضعیت سفارش (`state`) را یک مرحله به جلو ببرد (تا حداکثر مقدار ۲).

در نهایت، متد `getTotalPrice` باید قیمت کل سفارش را به صورت یک عدد صحیح بازگرداند. این مقدار از جمع قیمت تمام آیتم‌ها به‌علاوه‌ی مالیات محاسبه می‌شود و نتیجه باید گرد شود. نرخ مالیات باید به صورت یک متغیر کلاسی ثابت `(static final float)` با نام `tax_rate` برابر با ۹.۴ درصد تعریف شود و مقدار آن غیرقابل تغییر باشد.

توجه کنید که تمام پیام‌های خطا `(exception)` در این پروژه باید از نوع `IllegalArgumentException` باشند. نباید از `System.out` برای مدیریت خطا استفاده شود.

یک مثال ساده از `IllegalArgumentException` :

```
import java.lang.IllegalArgumentException;

public class Example {

    public static void main(String[] args) {
        Example ex = new Example();
        ex.setAge(25);
        ex.setAge(-3); // This will throw an exception
    }

```

```
    public void setAge(int age) {
        // Throw IllegalArgumentException if the provided age
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be
        }
    }
}
```

توجه کنید که نباید به ساختار پکیجینگ پروژه دست بزنید.

Mini Edu

فایل اولیه‌ی تمرین را می‌توانید از این لینک دانلود کنید.

در این تمرین، هدف طراحی مجموعه ای از کلاس های شی گرا برای مدیریت دانشجویان، دروس، انتخاب واحد و نمره دهی است.

ساختار پروژه

```
src.zip/
└─ university/
    └─ management/
        ├─ Course.java
        ├─ Enrollment.java
        ├─ Student.java
        └─ UniversitySystem.java
```

کلاس‌ها و مشخصات دقیق

▼ کلاس `Student`

کلاس `Student` نمایانگر یک دانشجو است و شامل ویژگی‌های `firstName` (نام دانشجو)، `lastName` (نام خانوادگی دانشجو)، `entryYear` (سال ورود)، `studentNumber` (شماره دانشجویی) و `enrollments` (لیست `List` دروسی که دانشجو اخذ کرده است، از نوع کلاس `Enrollment`) می‌باشد. توجه داشته باشید که شماره دانشجویی باید ترکیب سه رقم سال ورود و سه رقم شمارنده سالانه باشد. برای مثال، اولین دانشجوی سال `404` شماره‌ی `404001` خواهد داشت، دومین دانشجوی همان سال `404002` و به همین ترتیب برای سایر دانشجویان. شمارنده‌ی سالانه باید برای هر سال ورود به‌صورت جداگانه مدیریت شود.

کلاس Student دارای دو سازنده است: سازنده پیش‌فرض که پارامترهای firstName و lastName را دریافت می‌کند و سال ورود پیش‌فرض آن 403 است. در این سازنده شماره دانشجویی باید به‌صورت خودکار تولید شود و لیست enrollments باید خالی باشد. سازنده دوم پارامترهای firstName ، lastName و entryYear را دریافت می‌کند و شماره دانشجویی مطابق شمارنده سالانه تولید می‌شود و لیست enrollments همچنان خالی است.

متدهای دسترسی getFirstName ، getLastName ، getEntryYear و getStudentNumber به ترتیب نام، نام خانوادگی، سال ورود و شماره دانشجویی را بازمی‌گردانند. متد getEnrollments باید همواره یک لیست معتبر بازگرداند و هیچ‌گاه مقدار null برنگرداند؛ در صورت نداشتن درس، یک لیست خالی بازمی‌گرداند.

برای مدیریت دروس، کلاس Student باید دو متد addEnrollment(Enrollment enrollment) و removeEnrollment(Enrollment enrollment) داشته باشد. در addEnrollment اگر پارامتر enrollment مقدار null باشد، یک IllegalArgumentException پرتاب شود، و در غیر این صورت، شیء Enrollment به لیست اضافه می‌شود. متد removeEnrollment امکان حذف یک درس از لیست را فراهم می‌کند.

متد getAverage(int semester) برای محاسبه معدل دانشجو در ترم مشخص استفاده می‌شود و باید حتماً در پیاده سازی آن از داده‌های موجود در لیست enrollments استفاده شود. این متد فقط درس‌هایی را در نظر می‌گیرد که در همان ترم اخذ شده‌اند و نمره‌ای برای آن‌ها ثبت شده باشد. اگر دانشجو در آن ترم هیچ درسی نداشته باشد یا هیچ نمره‌ای ثبت نشده باشد، مقدار 0.0 بازگردانده می‌شود. مقدار معدل باید تا دو رقم اعشار با استفاده از Math.floor گرد شود تا تنها دو رقم اعشار حفظ گردد.

توجه کنید که کلاس Student باید با کلاس‌های Course و Enrollment هماهنگ باشد (در کلاس‌های بعدی تعریف می‌شوند) تا تست‌های واحد ارائه شده پاس شوند.

▼ **کلاس Course**

کلاس Course نمایانگر یک درس دانشگاهی است و شامل ویژگی‌های name (نام درس)، semester (شماره ترم ارائه)، و studentCount (تعداد دانشجویان ثبت‌نام‌شده) می‌باشد. فهرست دروس مجاز باید از پیش تعریف شود و تنها شامل دروس زیر است:

- ترم ۱: Math_I , Basic_Programming , Farsi
- ترم ۲: Math_II , Advanced_Programming , English

در سازنده‌ی Course(String name, int semester) ، در صورت نامعتبر بودن نام درس یا شماره‌ی ترم باید یک IllegalArgumentException پرتاب شود. مقدار اولیه‌ی studentCount باید 0 باشد. متدهای getName ، getSemester و getStudentCount به ترتیب نام، شماره ترم و تعداد دانشجویان را بازمی‌گردانند. متد increaseStudentCount باید با هر ثبت‌نام جدید، شمارنده‌ی دانشجویان را یک واحد افزایش دهد، و متد decreaseStudentCount نیز در صورت وجود دانشجو، آن را یک واحد کاهش دهد (اما هیچ‌گاه نباید مقدار منفی شود).

همچنین باید یک متد toString پیاده‌سازی شود که رشته‌ی خروجی آن در قالب "CourseName (semester <semesterNumber>) – students: <studentCount>" بازگردانده شود؛ برای مثال: "Math_I (semester 1) – students: 3" .

▼ کلاس Enrollment

کلاس Enrollment نمایانگر رابطه‌ای بین یک دانشجو (Student) و یک درس (Course) در یک ترم مشخص است. این کلاس شامل ویژگی‌های student ، course و grade می‌باشد. پارامترهای student و course نباید null باشند و در صورت null بودن، یک IllegalArgumentException پرتاب می‌شود. مقدار اولیه‌ی grade نیز null است و تا زمان تخصیص نمره با متد assignGrade(double grade) خالی باقی می‌ماند.

در هنگام ایجاد یک شیء Enrollment ، درس مربوطه به لیست enrollments دانشجو اضافه شده و شمارنده‌ی دانشجویان درس (studentCount) نیز با فراخوانی متد

increaseStudentCount افزایش می‌یابد. متد assignGrade(double grade) نمره‌ای بین 0 تا 20 دریافت می‌کند و در صورت خارج بودن از این بازه، IllegalArgumentException پرتاب می‌شود. متدهای getGrade , getCourse و getStudent به ترتیب نمره، درس و دانشجوی مربوطه را بازمی‌گردانند.

همچنین توجه داشته باشید که ثبت یک درس برای دانشجو نباید تکراری باشد و کلاس باید با کلاس‌های Student و Course هماهنگ باشد تا تمامی تست کیس‌ها پاس شوند. به این ترتیب، Enrollment نه تنها داده‌های دانشجو و درس را مدیریت می‌کند بلکه تضمین می‌کند که ثبت‌نام و نمره‌دهی مطابق محدودیت‌های تعریف شده انجام شود و اطلاعات همواره سازگار باقی بمانند.

▼ کلاس UniversitySystem

کلاس UniversitySystem نمایانگر سیستم مرکزی دانشگاه است که تمام داده‌ها و عملیات مدیریتی مرتبط با دانشجویان و دروس در آن نگهداری می‌شوند. این کلاس شامل دو فیلد students و courses است که به ترتیب فهرست تمام دانشجویان ثبت‌شده و فهرست تمام دروس ثابت (ترم 1 و 2) را نگهداری می‌کنند. لیست دروس هنگام ایجاد سیستم به‌صورت خودکار با دروس زیر مقداردهی می‌شود:

- ترم 1: Math_I , Basic_Programming , Farsi
- ترم 2: Math_II , Advanced_Programming , English

سازنده‌ی UniversitySystem این لیست‌ها را مقداردهی اولیه کرده و لیست دانشجویان را خالی ایجاد می‌کند. برای افزودن دانشجو، دو متد وجود دارد: addStudent(String firstName, String lastName, int entryYear) که یک شیء Student ایجاد کرده و شماره دانشجویی آن را به‌صورت خودکار می‌سازد و به لیست اضافه می‌کند و شیء ایجاد شده را برمیگرداند و addStudent(String firstName, String lastName) که سال ورود پیش‌فرض 403 را استفاده می‌کند. همچنین متدهای listStudents و listStudentsByPrefix(String prefix) امکان دریافت فهرست تمامی دانشجویان یا دانشجویان با شماره‌ای که با پیشوند مشخص

شروع می‌شود را فراهم می‌کنند و در صورت نبود دانشجو، لیست خالی بازمی‌گردانند.

برای مشاهده دروس ترم مشخص، متد `getCoursesBySemester(int semester)` فهرست دروس مربوطه را بازمی‌گرداند. عملیات ثبت‌نام دانشجو در یک درس با متد `assignCourse(String courseName, int semester, String studentNumber)` انجام می‌شود؛ در این متد ابتدا دانشجو و درس پیدا می‌شوند و سپس بررسی می‌شود که دانشجو پیش‌تر همان درس را ثبت‌نام نکرده باشد. اگر قبلاً ثبت‌نام کرده باشد، `IllegalArgumentException` پرتاب می‌شود. در غیر این صورت، یک شیء `Enrollment` ایجاد و به دانشجو اضافه می‌گردد و شمارنده‌ی دانشجویان درس (`studentCount`) یک واحد افزایش می‌یابد. برای حذف درس، متد `removeCourse(String courseName, int semester, String studentNumber)` وجود دارد که درس موردنظر را از لیست انتخابی دانشجو حذف کرده و شمارنده‌ی درس کاهش می‌یابد؛ اگر دانشجو چنین درسی نداشته باشد، `IllegalArgumentException` پرتاب می‌شود.

ثبت نمره با متد `assignGrade(String courseName, int semester, String studentNumber, double grade)` انجام می‌شود؛ در این متد ابتدا بررسی می‌شود که دانشجو درس موردنظر را اخذ کرده باشد و سپس نمره بین `0` تا `20` اختصاص می‌یابد. در صورت عدم وجود درس برای دانشجو، `IllegalArgumentException` پرتاب می‌شود. محاسبه معدل با متد `getAverage(String studentNumber, int semester)` انجام می‌شود و فقط دروس دارای نمره لحاظ می‌شوند. مقدار معدل باید تا دو رقم اعشار و با گرد کردن به پایین (`Math.floor`) بازگردانده شود و اگر دانشجو در ترم هیچ نمره‌ای نداشته باشد، مقدار `0.0` بازگردانده می‌شود.

کلاس همچنین شامل متدهای کمکی `findStudent(String studentNumber)` و `findCourse(String name, int semester)` است که برای پیدا کردن دانشجو یا درس استفاده می‌شوند و در صورت نبود، `IllegalArgumentException` پرتاب می‌کنند.

توجه کنید که تمام پیام‌های خطا (exception) در این پروژه باید از نوع `IllegalArgumentException` باشند. نباید از `System.out` برای مدیریت خطا استفاده شود.

یک مثال ساده از `IllegalArgumentException` :

```
import java.lang.IllegalArgumentException;

public class Example {

    public static void main(String[] args) {
        Example ex = new Example();
        ex.setAge(25);
        ex.setAge(-3); // This will throw an exception
    }

    public void setAge(int age) {
        // Throw IllegalArgumentException if the provided age
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be
        }
    }
}
```

توجه کنید که نباید به ساختار پکیجینگ پروژه دست بزنید.

JSON Parser

ساختار مورد انتظار:

```
someName.zip/
├─ Main.java
├─ SecondClass.java
.
.
.
└─ LastClass.java
```

در هیچ کدام از کلاس‌های این سوال نباید از Packaging استفاده کرد!!

طراحی UML

شما باید طبق توضیحات و نیازمندی‌های زیر یک UML طراحی کرده و فایل PDF آن را درکنار فایل های تمرین ارسال کنید. این بخش دارای 25 نمره علاوه بر نمره داخل کوئرا است. (نمره امتیازی نیست)

در این تمرین تلاش خواهیم کرد، با استفاده از مفاهیم شی‌گرایی و به طور بازگشتی یک فایل json را تجزیه و سپس تحلیل کنیم. در این راستا نیاز می‌باشد که در ورودی یک فایل json در یک خط داده شده‌است و سپس تا به هنگامی که ورودی داده می‌شود دستور زیر اجرا شود.

در هر خط آدرسی از json وارد شده به شما داده می‌شود و شما باید مقدار آن را برگردانید.

```
<keyAddress>
```

که در آن keyAddress یک رشته‌است که برای مثال در برابر json زیر

```
{
    "Level1Key": {
        "Level2Key": [
            {
                "Level3Key": "The Key points here",
                "ABBD": 2
            },
            {},
            {}
        ],

        "Level2Kez": "23"
    },
    "Level1Key2": "1",
    "Level1Key3": 2
}
```

ورودی	خروجی
Level1Key/Level2Key[0]/Level3Key	The Key points here
Level1Key/LeveL2Key	[{},{},{"ABBD":2,"Level3Key":"The Key points here"}]
Level1Key	Level2Key":[{"ABBD":2,"Level3Key":"The Key"} {"points here"},{},{}],"Level2Kez":"23
Level1Key2	1
Level1Key3	2

آدرس زیر به مقدار "The Key points here" اشاره می‌کند.

```
Level1Key/Level2Key[0]/Level3Key
```

اگر کلید به رشته یا عدد یا مقدار اولیه دیگری اشاره کرد همان چاپ شود. و همچنین اگر به یک شی ء (Object) یا یک آرایه اشاره بکند، مقدار آن به صورت json برگردانده شود .اگر دو یا چند شی در یک سطح قرار داشتند، اولویت چاپ با شیء ای می باشدکه از لحاظ الفبایی،*Lexicographically* کوچکتر است . ،

توجه: اعضای Json میتوانند ارایه های 2بعدی هم باشند.

توجه: اگر آدرس یافت نشد یا اینکه مقدار داخلی $null$ بود عبارت زیر چاپ شود.

```
null
```

To Do List

ساختار مورد انتظار:

```
someName.zip/
├─ Main.java
├─ SecondClass.java
.
.
.
└─ LastClass.java
```

در هیچ کدام از کلاس‌های این سوال نباید از Packaging استفاده کرد!!

طراحی UML

شما باید طبق توضیحات و نیازمندی‌های زیر یک UML طراحی کرده و فایل PDF آن را درکنار فایل های تمرین ارسال کنید. این بخش دارای 25 نمره علاوه بر نمره داخل کوئرا است. (نمره امتیازی نیست)

در این تمرین تلاش خواهیم کرد، با استفاده از مفاهیم شی‌گرایی و استفاده از ساختارهای داده، یک *لیست برنامه‌ریزی* یا یک *To do list* طراحی کنیم. در راستای انجام این هدف یک *برنامه* را به شکل زیر تعریف می‌کنیم.

برنامه: به یک مجموعهٔ شامل *عنوان، شمارهٔ یکتا، مهلت انجام، میزان اهمیت* (عدد ۱ معادل کمترین اهمیت و ۱۰ بیشترین اهمیت است) و مجموعه‌ای از *دسته‌بندی‌ها* (category) برنامه گفته می‌شود.

همچنین هرکدام از دسته‌بندی‌ها خود یک اهمیتی دارند که این اهمیت نیز از یک تا ده به شکل اهمیت برنامه‌ها شماره‌گذاری می‌شوند.

در راستای مدیریت برنامه‌ها، یک سری دستور تعریف می‌شود که مراتب انجام آن‌ها به شرح ذیل است.

▼ اضافه کردن دسته‌بندی

```
add category -name <name> -importance <importance coefficient>
```

هر دسته‌بندی یک نام و یک ضریب اهمیت دارد که نام فقط از حروف [a-zA-Z0-9] تشکیل شده‌است، و ضریب اهمیت نیز یک عدد صحیح بین ۱ تا ۱۰ خواهد بود. نام دسته‌بندی باید یکتا باشد.

خطاها

اگر نام به فرمت قابل قبول نبود:

```
The format of the name is not valid.
```

اگر نام تکراری بود:

```
The name is already used.
```

اگر ضریب اهمیت عدد نبود:

```
The importance coefficient must be a number.
```

اگر ضریب بین ۱ – ۱۰ نبود:

```
The importance coefficient must be 1-10.
```

و در صورت عدم خطا:

```
Category <name> added.
```

▼ اضافه کردن برنامه

```
add todo -name <todo
name> -deadline <Remaining days> -categories <category 1>
<category 2> …. <category n> -importance <importance
coefficient>
```

نام فقط از حروف [a-zA-Z0-9] تشکیل شده‌است،

و ضریب اهمیت نیز یک عدد صحیح بین ۱ تا ۱۰ خواهد بود. همچنین مهلت باید یک عدد

اعشاری مثبت و دسته‌بندی‌ها نیز باید وجود داشته باشند. توجه کنید که نام برنامه

باید یکتا باشد.

خطاها

اگر نام به فرمت قابل قبول نبود:

```
The format of the name is not valid.
```

اگر نام تکراری بود:

```
The name is already used.
```

اگر ضریب اهمیت عدد نبود:

```
The importance coefficient must be a number.
```

اگر ضریب بین ۱ – ۱۰ نبود:

```
The importance coefficient must be 1-10.
```

اگر مهلت عدد نبود یا مثبت نبود:

```
The deadline must be a positive number.
```

اگر یکی از دسته‌بندی‌ها یافت نشد (درصورت این‌که چند دسته‌بندی یافت نشدند

آن که در این دستور زودتر وارد شده‌است در نظر گرفته می‌شود).

```
Category <category> not found.
```

و در صورت عدم خطا:

```
Todo <name> added.
```

▼ تغییر اهمیت دسته‌بندی

```
update category -name <name> -importance <importance coefficient>
```

خطاها

اگر نام یافت نشد:

```
The category not found.
```

اگر ضریب اهمیت عدد نبود:

```
The importance coefficient must be a number.
```

اگر ضریب بین ۱ – ۱۰ نبود:

```
The importance coefficient must be 1-10.
```

و در صورت عدم خطا:

```
Category <name> updated.
```

▼ حذف دسته‌بندی

```
remove category -name <name>
```

در طی این دستور باید تمامی برنامه‌هایی که دسته‌بندی مدنظر را شامل می‌شوند،

این دسته‌بندی را حذف کنند.

خطاها

اگر نام یافت نشد:

```
The category not found.
```

درصورت عدم خطا:

```
The category deleted.
```

▼ حذف برنامه

```
remove todo -name <name>
```

خطاها

اگر نام یافت نشد:

```
The todo not found.
```

درصورت عدم خطا:

```
The todo deleted.
```

▼ نشان کردن برنامه به عنوان انجام‌شده

```
do todo -name <name>
```

خطاها

اگر نام یافت نشد:

```
The todo not found.
```

درصورت عدم خطا:

```
The todo is done.
```

▼ نمایش برنامه‌ها

توجه شود، که در تمامی مراحل چاپ کردن، ترتیب چاپ کردن برنامه‌ها با «اولویت»

بالاتر است. و اگر اولویت دو برنامه یکسان بود آنگاه آن که از نظر الفبایی بزرگتر

است چاپ می‌شود (Lexicographically).

اولویت یک برنامه به شکل زیر تعریف می‌شود.

اولویت برنامه: (اهمیت برنامه * ۱۰ + جمع اهمیت‌های تمام دسته‌بندی‌ها) تقسیم

بر مهلت ارائه

```
print todos (-isNotDone) (-deadline <x>) (-category <y>)
```

هر کدام از سه عبارت داخل پرانتز ممکن است بیاید یا نیاید.

اگر `-isNotDone` بیاید آنگاه

فقط برنامه‌هایی نمایش داده می‌شود که هنوز انجام نشده‌اند.

اگر `-deadline` وارد شود، فقط

برنامه‌هایی نمایش داده می‌شود که مهلت باقی‌مانده‌اشان از x کوچکترمساوی باشد.

اگر `category` بیاید آنگاه

فقط برنامه‌هایی که در دسته‌بندی y قرار دارند چاپ می‌شود.

اگر با فاکتور های مدنظر هیچ برنامه‌ای موجود نبود متن زیر چاپ می‌شود.

```
No todo found.
```

تضمین می‌شود در این دستور ورودی‌های عجیبی که باعث ایجاد خطا بشوند داده

نشود.

نحوهٔ چاپ کردن هر برنامه نیز به قالب زیر خواهد بود.

```
Task: <name>\t| importance: <importance>\t| deadline: <deadline|9
```

تکته: اگر از زمان ددلاین گذشته بود، همان `0.00` را چاپ کنید.

▼ جلوبردن زمان

```
advance time <x> days
```

که در آن x یک عدد (نه

لزوما صحیح) نامنفی است.

اگر عدد وارد شده معتبر نبود خطای زیر داده می‌شود:

```
The days you entered is not valid.
```

▼ چاپ کردن دسته‌بندی‌ها

```
print categories
```

در این دستور دسته‌بندی‌ها به طور نزولی به ترتیب اهمیت و در صورت برابری

اهمیت به صورت الفبایی، در هر خط چاپ می‌شوند. فورمت چاپ هر دسته بندی به شکل زیر

است.

```
<category> (<importance>)
```

و اگر هیچ دسته‌بندی‌ای یافت نشد متن زیر ارسال گردد.

```
No categories yet.
```

▼ دستور اتمام

```
exit
```

با این دستور از برنامه خارج می‌شویم.

اگر دستوری وارد شد که در دستورهای گفته شده معتبر نبود متن زیر چاپ می‌شود.

```
Invalid command!
```

چند نمونه از ورودی و خروجی برنامه

▼ نمونه 1

ورودی نمونه ۱

```
print categories

add category -name work -importance 8
```

```
add category -name hork -importance 8

print categories

add category -name study -importance 6

add category -name personal -importance 5

add category -name invalid@ -importance 4

add category -name work -importance 7

add category -name hobby -importance ten

add category -name fun -importance 11

add category -name travel -importance 9

add todo -name task1 -deadline 5 -categories work study -importar

add todo -name task2 -deadline 3 -categories personal -importance

add todo -name task3 -deadline 4 -categories hobby -importance 8

add todo -name task4 -deadline -2 -categories work -importance 7

add todo -name task5 -deadline 6 -categories work sport -importar

add todo -name task6 -deadline 2 -categories work -importance 11

add todo -name task7 -deadline 7 -categories travel -importance 1

add todo -name invalid@ -deadline 4 -categories work -importance

add todo -name task8 -deadline 5 -categories work study travel -i

add todo -name task1 -deadline 5 -categories work -importance 6
```

```
print todos

print todos -isNotDone

print todos -deadline 4

print todos -category work

do todo -name task3

do todo -name task100

update category -name work -importance 10

update category -name notexist -importance 5

remove category -name hobby

remove category -name notexist

print todos -isNotDone -category work

advance time 2 days

advance time -1 days

advance time two days

print todos -deadline 3

remove todo -name task2

remove todo -name task100

print todos -isNotDone

do todo -name task7
```

```
print todos -isNotDone

exit
```

خروجی نمونه ۱

```
No categories yet.

Category work added.

Category hork added.

hork (8)

work (8)

Category study added.

Category personal added.

The format of the name is not valid.

The name is already used.

The importance coefficient must be a number.

The importance coefficient must be 1-10.

Category travel added.

Todo task1 added.

Todo task2 added.

Category hobby not found.
```

```
The deadline must be a positive number.

Category sport not found.

The importance coefficient must be 1-10.

Todo task7 added.

The format of the name is not valid.

Todo task8 added.

The name is already used.

Task: task1    | importance: 9    | deadline: 5.00 days

Task: task8    | importance: 7    | deadline: 5.00 days

Task: task2    | importance: 5    | deadline: 3.00 days

Task: task7    | importance: 10    | deadline: 7.00 days

Task: task1    | importance: 9    | deadline: 5.00 days

Task: task8    | importance: 7    | deadline: 5.00 days

Task: task2    | importance: 5    | deadline: 3.00 days

Task: task7    | importance: 10    | deadline: 7.00 days

Task: task2    | importance: 5    | deadline: 3.00 days

Task: task1    | importance: 9    | deadline: 5.00 days

Task: task8    | importance: 7    | deadline: 5.00 days

The todo not found.
```

```
The todo not found.

Category work updated.

The category not found.

The category not found.

The category not found.

Task: task1    | importance: 9    | deadline: 5.00 days

Task: task8    | importance: 7    | deadline: 5.00 days

The days you entered is not valid.

The days you entered is not valid.

Task: task2    | importance: 5    | deadline: 1.00 days

Task: task1    | importance: 9    | deadline: 3.00 days

Task: task8    | importance: 7    | deadline: 3.00 days

The todo deleted.

The todo not found.

Task: task1    | importance: 9    | deadline: 3.00 days

Task: task8    | importance: 7    | deadline: 3.00 days

Task: task7    | importance: 10    | deadline: 5.00 days

The todo is done.

Task: task1    | importance: 9    | deadline: 3.00 days
```

```
Task: task8    | importance: 7    | deadline: 3.00 days
```

▼ نمونه 2

ورودی نمونه ۲

```
add category -name fitness -importance 8

add category -name university -importance 9

add category -name errands -importance 5

add category -name leisure -importance 7

add category -name invalid# -importance 4

add todo -name workout -deadline 2 -categories fitness -importan

add todo -name groceryShopping -deadline 1 -categories errands -i

add todo -name projectA -deadline 5 -categories university -impo

add todo -name weekendTrip -deadline 10 -categories leisure -impo

add todo -name readBook -deadline 4 -categories leisure fitness -

add todo -name gym -deadline -3 -categories fitness -importance 8

print todos

do todo -name groceryShopping

print todos -isNotDone
```

```
update category -name fitness -importance 10

print todos -category fitness

advance time 1 days

print todos -deadline 3

remove todo -name weekendTrip

print todos -isNotDone

exit
```

خروجی نمونه ۲

```
Category fitness added.

Category university added.

Category errands added.

Category leisure added.

The format of the name is not valid.

Todo workout added.

Todo groceryShopping added.

Todo projectA added.

Todo weekendTrip added.
```

```
Todo readBook added.

The deadline must be a positive number.

Task: groceryShopping    | importance: 5    | deadline: 1.00 days

Task: workout    | importance: 7    | deadline: 2.00 days

Task: projectA    | importance: 10    | deadline: 5.00 days

Task: readBook    | importance: 4    | deadline: 4.00 days

Task: weekendTrip    | importance: 6    | deadline: 10.00 days

The todo is done.

Task: workout    | importance: 7    | deadline: 2.00 days

Task: projectA    | importance: 10    | deadline: 5.00 days

Task: readBook    | importance: 4    | deadline: 4.00 days

Task: weekendTrip    | importance: 6    | deadline: 10.00 days

Category fitness updated.

Task: workout    | importance: 7    | deadline: 2.00 days

Task: readBook    | importance: 4    | deadline: 4.00 days

Task: groceryShopping    | importance: 5    | deadline: 0.00 days

Task: workout    | importance: 7    | deadline: 1.00 days

Task: readBook    | importance: 4    | deadline: 3.00 days

The todo deleted.
```

```
Task: workout    | importance: 7    | deadline: 1.00 days

Task: projectA    | importance: 10    | deadline: 4.00 days

Task: readBook    | importance: 4    | deadline: 3.00 days
```

پناهگاه حیوانات

در این تمرین می‌خواهیم یک سامانه شیءگرا برای مدیریت پناهگاه حیوانات طراحی و پیاده‌سازی کنیم که بتواند حیوانات، قفس‌ها، درخواست‌های سرپرستی، صف انتظار و گزارش‌های ساده را مدیریت کند.

ساختار پروژه

```
src/
└─ model/
|    ├─ Animal.java
|    ├─ Enclosure.java
|    ├─ Applicant.java
|    └─ AdoptionRequest.java
├─ core/
|    └─ Shelter.java
└─ Main.java
```

کلاس‌ها و مشخصات دقیق

▼ کلاس `Animal`

نمایندهٔ هر حیوان در پناهگاه است.

ویژگی‌ها

نوع	نام	توضیح
`String`	`id`	شناسه یکتا
`String`	`name`	نام حیوان
`String`	`species`	گونه (مثلاً `DOG` , `CAT` , `RABBIT`)

int	age	سن
Set<String>	traits	ویژگی‌های رفتاری (مثل friendly ، calm ، trained)
String	status	وضعیت سلامت (HEALTHY , UNDER_TREATMENT , SPECIAL_NEED)

سازنده

```
public Animal(String id, String name, String species, int ag
```

متدها

```
public String getId()
public String getName()
public String getSpecies()
public int getAge()
public Set<String> getTraits()
public String getStatus()
public void setStatus(String status)
public boolean isAdoptable()
public String getSummary()
```

توضیحات:

- `isAdoptable()` : فقط اگر حیوان سالم بود، یعنی وضعیت او `"HEALTHY"` باشد، مقدار `true` برمی‌گرداند در غیر این صورت باید `false` برگرداند.
- `getSummary()` : خروجی آن دقیقا به صورت زیر است:

```
<id> - <name> (<species>, age <age>, status <status>)
```

▼ کلاس `Enclosure`

نمایندهٔ قفس (یا محل نگهداری حیوانات).

ویژگی‌ها

نوع	نام	توضیح
`String`	`id`	شناسه قفس
`int`	`capacity`	ظرفیت نگهداری
`Set<String>`	`allowedSpecies`	گونه‌های مجاز
`ArrayList<Animal>`	`animals`	حیوانات درون قفس

سازنده

```
public Enclosure(String id, int capacity, Set<String> allowe
```

سازنده این کلاس، سه مقدار ذکر شده را دریافت و آنها را در کلاس مقداردهی کرده و لیست `animals` را نیز `new` می‌کند.

متدها

```
public String getId()
public int getCapacity()
public boolean hasSpace()
public boolean addAnimal(Animal a)
public boolean removeAnimal(String animalId)
public String listAnimals()
```

توضیحات:

- `hasSpace()` : اگر پناهگاه ظرفیت خالی داشته باشد، `true` و در غیر این صورت `false` برمی‌گرداند.
- `addAnimal(a)` : فقط در صورتی که گونه حیوان در `allowedSpecies` و جا وجود داشته باشد، حیوان جدید را اضافه می‌کند و مقدار `true` برمی‌گرداند. در غیر این صورت `false` برگردانده شود.
- `removeAnimal(id)` : اگر شناسه حیوان یافت شد حذف و `true` ، در غیر این صورت `false` برمی‌گرداند.
- `listAnimals()` :با استفاده از `Iterator` تمام حیوانات را با استفاده از متد `getSummary()` آنها لیست می‌کند، با یک `\n` بهم چسبانده و رشته نهایی را ریترن می‌کند. (برای درک بهتر، نمونه خروجی را مشاهده کنید.)

▼ **کلاس `Applicant`**

نمایندهٔ فردی که قصد سرپرستی دارد.

ویژگی‌ها

نوع	نام	توضیح
`String`	`id`	شناسه متقاضی
`String`	`name`	نام متقاضی
`Set<String>`	`preferredSpecies`	گونه‌های مورد علاقه
`boolean`	`hasChildren`	آیا متقاضی کودک دارد یا خیر

سازنده

```
public Applicant(String id, String name, Set<String> preferr
```

متدها

```
public String getId()
public String getName()
public Set<String> getPreferredSpecies()
public boolean hasChildren()
public String getProfile()
```

توضیحات:

- `getProfile()` : خروجی آن به صورت زیر است:

```
<id> - <name> (prefers <species1, species2>, hasChildren=<tr
```

▼ کلاس **`AdoptionRequest`**

نمایندهٔ درخواست سرپرستی است.

ویژگی‌ها

نوع	نام	توضیح
String	code	کد درخواست
String	applicantId	شناسه متقاضی
String	species	گونه درخواستی
String	status	وضعیت درخواست (ACTIVE , MATCHED , CANCELED)

سازنده

```
public AdoptionRequest(String code, String applicantId, Stri
```

در هنگام ایجاد، مقدار `status` برابر `"ACTIVE"` تنظیم شود.

متدها

```
public String getCode()
public String getApplicantId()
public String getSpecies()
public String getStatus()
public void setStatus(String status)
public String getInfo()
```

توضیحات:

- متد `getInfo()` باید خروجی‌ای به صورت زیر داشته باشد:

```
<code> - <applicantId> requested <species> (status <status>)
```

▼ کلاس `Shelter`

کلاس اصلی سیستم که عملیات موردنیاز پناهگاه را مدیریت می‌کند.

ویژگی‌ها

نوع	نام	توضیح
`Map<String, Animal>`	`animals`	حیوانات ثبت‌شده
`Map<String, Enclosure>`	`enclosures`	قفس‌ها
`Map<String, Applicant>`	`applicants`	متقاضیان

Map<String, AdoptionRequest>	requests	درخواست‌ها
LinkedList<String>	waitingList	صف انتظار برای گونه‌های بدون قفس

سازنده

```
public Shelter()
```

متدها

```
public boolean addAnimal(Animal a)
public boolean addEnclosure(Enclosure e)
public boolean registerApplicant(Applicant a)
public AdoptionRequest createRequest(String applicantId, Str
public boolean assignAnimalToEnclosure(String animalId, Stri
public boolean adoptAnimal(String requestCode, String animal
public String listAllAnimals()
public String listAvailableAnimals(String species)
public String listRequests()
```

توضیحات:

- `addAnimal(a)` : اگر حیوان با همان `id` وجود نداشته باشد، اضافه کند و `true` برگرداند. در غیر این صورت `false` برمی‌گرداند.
- `addEnclosure(e)` : در صورت یکتا بودن `id` ، قفس را به لیست قفس‌ها اضافه کند و `true` برگرداند. در غیر این صورت `false` برگرداند.
- `registerApplicant(a)` : متقاضی جدید را در صورت یکتا بودن `id` در لیست متقاضیان ثبت کند و `true` برگرداند. در غیر این صورت `false` برگرداند.

- `createRequest(applicantId, species)` : کد درخواست باید از شمارنده‌ی `static int counter` تولید شود و در *Map* درخواست‌ها ذخیره گردد. (کد درخواست از متغیر ایستای `counter` شروع‌شده با مقدار 1 ساخته می‌شود، مثلاً `R1` ، `R2` ، ...)
- `assignAnimalToEnclosure()` : اگر گونه‌ی حیوان در `allowedSpecies` و ظرفیت موجود باشد، حیوان اضافه شود و `true` برگردانده شود. در غیر این صورت شناسه حیوان به `waitingList` اضافه گردد و `false` برگردانده شود.
- `adoptAnimal(requestCode, animalId)` : اگر حیوان قابل واگذاری (بررسی با `isAdoptable()`) و درخواست فعال باشد، حیوان از پناهگاه حذف و وضعیت درخواست به `MATCHED` تغییر کند، حیوان از قفس مربوطه و از لیست حیوانات پناهگاه حذف شود و در نهایت مقدار `true` برگردانده شود. در غیر این صورت `false` برگردانده شود و درخواست `ACTIVE` بماند.
- `listAllAnimals()` : همه حیوانات پناهگاه را با `getSummary()` هر حیوان در خطوط جدا بازگرداند.
- `listAvailableAnimals(species)` : فقط حیوانات آن گونه که قابل واگذاری‌اند (`isAdoptable()`) را لیست کرده، خروجی `getSummary()` آنها را با یک `\n` به هم چسبانده و رشته نهایی را ریترن کند.
- `listRequests()` : تمام درخواست‌ها با وضعیت فعلی‌شان را با استفاده از متد `getInfo()` با یک `\n` به هم چسبانده و رشته نهایی را ریترن کند.

نکات مهم

۱. پکیج‌های پروژه لزوما باید مشابه ساختار ذکر شده در ابتدای این سوال باشند.
۲. در تمامی کلاس‌ها، اصول شی‌گرایی رعایت شود.
۳. همه ویژگی‌های کلاس‌ها باید کپسوله‌سازی شده و متدهای `getter` و `setter` داشته باشند.
۴. دقت کنید سامانه داوری به بزرگی یا کوچکی حروف حساس است. خروجی‌های رشته‌ای، باید دقیقا

مطابق با نمونه ذکر شده در صورت سوال باشند، در غیر این صورت امتیاز آن بخش را دریافت نخواهید کرد.

۵. به جز موارد خطایی که در سوال ذکر شده، سایر خطاها نیازی به بررسی و شرط ندارند و فرض کنید ورودی معتبر به کد داده خواهد شد.

کلاس Main

```
public class Main {
    public static void main(String[] args) {
        Shelter shelter = new Shelter();
        Animal rex = new Animal("A001", "Rex", "DOG", 3, new H
        Animal mimi = new Animal("A002", "Mimi", "CAT", 2, new

        shelter.addAnimal(rex);
        shelter.addAnimal(mimi);
        shelter.addEnclosure(new Enclosure("E1", 2, new HashSe
        shelter.assignAnimalToEnclosure("A001", "E1");
        shelter.registerApplicant(new Applicant("P01", "Sara",
        AdoptionRequest req = shelter.createRequest("P01", "CA
        shelter.adoptAnimal(req.getCode(), "A002");
        System.out.println(shelter.listAllAnimals());
        System.out.println(shelter.listRequests());
    }
}
```

خروجی نمونه

```
A001 - Rex (DOG, age 3, status HEALTHY)
A002 - Mimi (CAT, age 2, status UNDER_TREATMENT)
R1 - P01 requested CAT (status ACTIVE)
```