برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

پروژه اول: جیسونیک!

سونیک که به تازگی یادگیری برنامه‌نویسی را شروع کرده تصمیم گرفته است تا یک سیستم مدیریت داده‌ها بسازد. او می‌خواهد سیستمی طراحی کند که مانند خودش سریع، کارآمد و قابل اعتماد باشد. اما سونیک به دلیل درگیری با ماموریت‌های دیگرش، وقت کافی برای تکمیل این پروژه را ندارد! به همین دلیل، از شما که وقت و مهارت کافی دارید، درخواست کمک کرده است تا **جیسونیک** را برای او پیاده‌سازی کنید.

هدف این پروژه، پیاده‌سازی برنامه‌ای است که با دریافت دستورات خاص، تغییرات لازم را روی داده‌ها اعمال کند و خروجی‌های مورد نظر را به درستی نمایش دهد. این برنامه باید بتواند انواع داده‌های سفارشی را بر اساس ساختارهایی که کاربر تعریف می‌کند، مدیریت کند و عملیات‌هایی مانند افزودن، حذف، به‌روزرسانی و جستجو را با سرعت و دقت بالا انجام دهد.

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

دریافت ورودی و خروجی

برنامه باید به صورت تعاملی عمل کند. زمانی که برنامه اجرا می‌شود، منتظر دریافت اولین دستور از کاربر می‌ماند. پس از ورود اولین دستور، برنامه آن را پردازش کرده و خروجی اجرای دستور را چاپ می‌کند. مجدد برنامه منتظر دریافت دستور بعدی می‌شود و این چرخه تا دریافت کلید واژه خاصی (مثلا quit) ادامه پیدا می‌کند.

برای چاپ سطر های خروجی، فرمت چاپ آزاد و به عهده خودتان است، ولی دقت داشته باشید که نام ستون ها و همچنین مقدار متناظر با هر ستون باید به خوبی قابل تشخیص باشد.

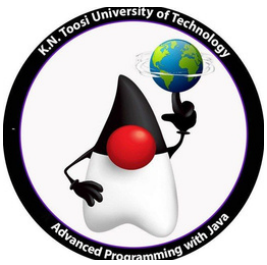
برای دریافت دستورات ورودی به نکات زیر دقت کنید:

- تمام دستور در یک سطر دریافت و پردازش می‌شود و خروجی متناظر با آن چاپ می‌شود.
- بین توکن های دستور (نام، پرانتز، کاما و ...) می‌تواند تعداد دلخواهی فاصله (space) قرار بگیرد و در اجرای دستور تاثیری ندارد.
- ورودی حساس به حروف بزرگ و کوچک (case-sensitive) نیست.

مدیریت خطا ها

یکی از مهم ترین ویژگی های یک نرم افزار خوب برخورد مناسب با خطا هاست. بنابراین برنامه باید قادر باشد به صورت مناسب با حالت های خطا برخورد کند. به عنوان مثال در صورتی که کاربر اقدامی تعریف نشده انتخاب کند، برنامه نباید متوقف شود و باید با پیغام خطای مناسب کاربر را از مشکل مطلع کند.

در ادامه به ازای هر اقدام، حالت های خطایی که برنامه شما ممکن است با آنها مواجه شود ذکر شده است. حین پیاده سازی توجه لازم را به این حالت های خطا داشته باشید و در صورت برخورد با آن ها، با متن خطای مناسب، کاربر را از مشکل مطلع کنید.



برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

آشنایی با JSON

JSON یک فرمت متنی سبک[1] برای تبادل داده ها است که به صورت ساختارمند و خوانا طراحی شده است. این فرمت به طور گسترده در برنامه های وب برای ارسال و دریافت داده‌ها بین سرور و کلاینت استفاده می‌شود. از جمله ویژگی های کلیدی این فرمت میتوان به موارد زیر اشاره کرد

- **ساختار ساده و خوانا:** داده‌ها به صورت جفت کلید-مقدار (key-value) نوشته می‌شوند.
- **مستقل از زبان برنامه‌نویسی:** اگرچه از جاوا اسکریپت الهام گرفته شده، اما توسط تمام زبان‌های برنامه‌نویسی قابل استفاده است.
- **فرمت متنی:** داده‌ها به صورت متن ذخیره می‌شوند که باعث سهولت در انتقال و خواندن می‌شود.

1.Light weight text format

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

JSON از دو نوع ساختار اصلی تشکیل شده است

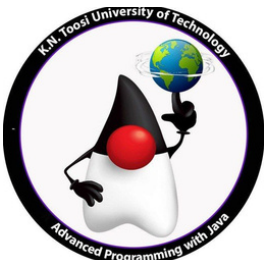
- **شئ (Object) :** مجموعه ای از جفت های کلید-مقدار محصور شده توسط کروشه {}. کلید همیشه از نوع String است اما مقدار می‌تواند از انواع مختلفی باشد که در این پروژه برای سادگی از تعدادی از آنها صرف نظر شده است و در ادامه درمورد آنها توضیحات لازم داده می‌شود.

```
{
    "id": 1,
    "name": "Ali",
    "age": 19,
    "email": "ali@kntu.ac.ir"
}
```

- **آرایه (Array) :** یک لیست مرتب از مقادیر محصور شده توسط براکت []. این مقادیر هم می‌توانند عضو هرکدام از انواع مختلف قابل استفاده برای مقدار ها در شئ ها باشند.

```
"skills": ["C++", "SQL"]
```

در توضیحات دستورات برنامه چندین مثال از ساختار کلی (JSON) مد نظر در این پروژه آورده شده و همین مقدار برای خواسته های این پروژه کفایت می‌کند. برای آشنایی و کسب اطلاعات بیشتر درمورد این فرمت به بخش منابع مفید مراجعه کنید.



برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

دستورات اصلی

قالب کلی دستورات این برنامه به صورت زیر می‌باشد:

command type (parameter) {JSON-like input}

در صورتی که هرکدام از قسمت های ذکر شده خالی باشد نیازی به آوردن نماد آن نیست و آن قسمت حذف می شود.

مثال دستور بدون پارامتر:

command type {JSON-like input}

❌**خطا ها:** قالب دستور مطابق ساختار ذکر شده رعایت نشده باشد، دستور موجود نباشد.

- **دستور create:** برای تعریف یک نوع داده جدید استفاده می‌شود. این دستور امکان مشخص کردن ساختار داده ها و قید هایی مانند نوع داده، اجباری بودن و یکتا بودن را برای هر فیلد را فراهم می‌کند. خروجی این دستور پیامی با محتوای موفقیت آمیز بودن یا نبودن عملیات است.

ساختار کلی دستور:

```
create <TypeName> {
      "field1": {"type": "data_type", "required": true/false, "unique": true/false},
      "field2": {"type": "data_type"},
      ...
}
```

🔔**توجه:** در برنامه اصلی دستورات در یک خط داده می‌شوند اما در مثال ها برای واضح تر شدن ساختار کلی، در چند خط نمایش داده شده‌اند.

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

🔔**توجه:** بخش های required و unique برای هر فیلد می‌توانند نوشته شده باشند یا نشده باشند. در صورت نبودن، مقدار پیش فرض false در نظر گرفته می‌شود. ترتیب نوشته شدن به صورت

type, required, unique

خواهد بود. نوع داده های برنامه محدود به موارد زیر می‌باشد:

نوع داده	مثال	مقدار پیشفرض	معادل جاوا
string	"Hello"	""	String
int	21	0	Integer
dbl	21.22	0.0	Double
bool	true	false	Boolean

نام تایپ ها و فیلد ها باید فقط شامل حروف الفبا، اعداد و کاراکتر زیر خط (_) باشد.

📎**مثال:**

```
create Person {
        "id": {"type": "int", "unique": true},
        "name": {"type": "string", "required": true},
        "age": {"type": "int", "unique": false},
        "email": {"type": "string"  "required": true, "unique": true},
        "isStudent": {"type": "bool"}
 }
```

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

ایجاد تایپ با نام Person شامل یک عدد صحیح منحصر به فرد برای هر شخص به عنوان id، نام به صورت رشته متنی اجباری که باید در هر ورودی وجود داشته باشد، یک عدد اختیاری به عنوان سن، یک رشته متنی اجباری که برای هر شخص منحصر به فرد است به عنوان ایمیل و یک بخش اختیاری برای مشخص شدن دانشجو بودن یا نبودن فرد.

❌**خطا ها:** تایپ با این نام از قبل وجود داشته باشد، بخش field ها خالی باشد، چند field با نام مشابه وجود داشته باشد، نوع داده نامعتبر باشد.

- **دستور insert:** این دستور امکان اضافه کردن نمونه های جدید به یک نوع داده را فراهم می‌کند. خروجی این دستور اطلاعات نمونه اضافه شده است.

  ساختار کلی دستور:

```
insert <TypeName> {"field1": value1,"field2": value2, ...}
```

📎**مثال:**

```
insert Person {
        "id": 1,
        "name": "Ali",
        "age": 19,
        "email": "ali@kntu.ac.ir",
        "isStudent": true
 }
```

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

با وارد کردن این دستور اطلاعات یک فرد جدید با اطلاعات داده شده به برنامه اضافه می‌شود.

🔔**توجه:** هنگام اضافه کردن داده جدید، باید قید های ذکر شده هنگام ساختن تایپ مورد نظر بررسی شوند. در هنگام اجرای مثال قبلی، باید بررسی شود از قبل نمونه ای از همین تایپ با id یا ایمیل مشابه با ورودی داده شده وجود نداشته باشد و درصورت وجود از اضافه شدن این نمونه جلوگیری شود. همچنین اطلاعات ضروری نمیتوانند خالی باشند اما اطلاعات غیرضروری میتوانند ذکر نشوند و در این صورت مقدار پیش فرض با نوع داده مربوطه جایگزین آن فیلد می شود.

📎**مثال:**

```
insert Person {
        "id": 2,
        "name": "Reza",
        "email": "reza@gmail.com",
        "isStudent": true
 }
```

چون فیلد age مقدار دهی نشده و جزو فیلد های ضروری نیست، مقدار پیش فرض اعداد صحیح (0) برای این نمونه قرار می گیرد.

❌**خطا ها:** تایپ با نام ذکر شده وجود نداشته باشد، قید ها (unique, required) رعایت نشده باشند، برای مقدار دهی یک فیلد از نوع داده نامعتبر استفاده شود (مثلا برای فیلد از نوع عدد صحیح، رشته متنی قرار داده شده باشد)، در تایپ مشخص شده فیلد با نام مورد نظر وجود نداشته باشد.

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

- **دستور update:** این دستور اطلاعات یک یا چند داده از برنامه را بر اساس فیلتر داده شده، به روز رسانی می‌کند. خروجی این دستور تعداد نمونه هایی که در نتیجه این دستور به روز رسانی شده اند می‌باشد.

  ساختار کلی دستور:

```
update <TypeName> (parameter) { "field1": new_value1 ,...}
```

📎مثال:

به روز رسانی همه نمونه های تایپ Person:

```
update Person { "age": 30 }
```

📎مثال:

به روز رسانی فیلد age و isStudent نمونه با id = 1 از تایپ Person:

```
update Person (id = 1) { "age": 30, “isStudent”: true }
```

عملگرهای مقایسه برای فیلتر محدود به موارد زیر می باشند:

عملگر	توضیح	مثال
=	بررسی برابری	name = “Mahdi”
> , <	مقایسه	age < 20

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

مثال:

```
update Person (30 > age) { "age": 30, “isStudent”:true }
```

خطا ها: تایپ با نام ذکر شده وجود نداشته باشد، قید ها (unique, required) هنگام به روز رسانی رعایت نشده باشند، برای مقدار دهی یا فیلتر یک فیلد از نوع داده نامعتبر استفاده شود (مثلا برای فیلد از نوع عدد صحیح، رشته متنی قرار داده شده باشد)، در تایپ مشخص شده فیلد با نام مورد نظر وجود نداشته باشد (هم در بخش فیلتر هم به روز رسانی)

- **دستور search:** این دستور امکان جست و جو و بازیابی داده ها از یک تایپ را فراهم می‌سازد. خروجی این دستور، نمونه هایی از تایپ مورد نظر که در شرایط فیلتر داده شده صدق می کنند می باشد. برای نحوه نمایش دادن اطلاعات شرط خاصی وجود ندارد ولی همه نمونه ها با همه اطلاعاتشان باید به صورت واضح و خوانا نمایش داده شوند.

ساختار کلی دستور:

```
search <TypeName> (parameter)
```

مثال:

نمایش تمامی نمونه ها از تایپ Person

```
search Person
```

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

مثال:

```
search Person (age > 30)
```

خطا ها: خطا های این دستور مشابه خطاهای بخش فیلتر دستور update است. تنها در نظر داشته باشید که اینجا مقدار دهی صورت نمیگیرد.

- **دستور delete:** این دستور امکان حذف تعدادی از نمونه های موجود از یک تایپ را فراهم می‌کند. خروجی این دستور تعداد نمونه های حذف شده توسط دستور داده شده است

  ساختار کلی دستور:

```
delete <TypeName> (parameter)
```

مثال:

حذف همه نمونه های تایپ Person:

```
delete Person
```

مثال:

حذف فرد با id = 1

```
delete Person  (id = 1)
```

خطا ها: مشابه خطا های بخش فیلتر دستور update

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

🚨 **نکات پیاده سازی:** پیاده سازی شما باید مبتنی بر متدهای استرینگ در جاوا باشد. پیاده سازی برنامه مشابه زبان C نمره ای نخواهد داشت (در نظر گرفتن flag، یا استفاده از رشته به صورت آرایه ای کاراکترها). استفاده از کتابخانه ها و پارسر های آماده برای عبارات **JSON** مجاز نیست و دانشجویان باید خودشان با استفاده از متد های پردازش رشته در جاوا و **regex** این عبارات را تحلیل کنند. همچنین تمام تلاش خود را برای نوشتن کدی خوانا و دارای قابلیت نگهداری مناسب به کار ببرید. توصیه می شود **مدلسازی اولیه** را انجام دهید و قابلیت هایی که قرار است اضافه کنید را در ذهن داشته باشید. مدل سازی خود را می توانید با دستیار آموزشی بررسی کنید و اشکالات احتمالی را رفع کنید. سپس بعد از انجام مدل سازی اولیه اقدام به پیاده سازی قابلیت ها کنید.

🧩**راهنمایی:** برای ذخیره سازی و نگهداری داده ها می‌توانید از **آرایه دو بعدی**، **ArrayList**، **HashMap**، و یا ترکیب هایی از آنها استفاده کنید. همچنین برای راحتی می توانید فرض کنید ظرفیت نمونه ها برای هر تایپ یک عدد ثابت مانند 100 است اما توجه کنید این عدد باید به راحتی از داخل کد قابل تغییر باشد

🔔**توجه:** دقت کنید موارد اصلی و خواسته شده در صورت پروژه از اولویت بیشتری برخوردارند. ابتدا موارد اصلی پروژه را با دقت پیاده و تست کنید و سپس بر روی موارد امتیازی کار کنید.

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

قابلیت های امتیازی

- **نوع داده آرایه:** برنامه باید علاوه بر انواع داده های گفته شده، از نوع داده آرایه هم پشتیبانی کند.

نوع داده	مثال	مقدار پیشفرض	معادل جاوا
list	[1, 2, 3]	[]	Array

📎**مثال:**

```
create Person {
        "id": {"type": "int", "unique": true},
        "name": {"type": "string", "required": true},
        "age": {"type": "int", "unique": false},
        "email": {"type": "string", "required": true, "unique": true},
        "skills": {"type": "list", "items": "string"}
}


insert Person {
    "id": 1,
    "name": "Ali",
    "age": 25,
    "email": "ali@kntu.ac.ir",
    "skills": ["Java", "Python"]
  }
```

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

با اضافه شدن این نوع داده یک فیلتر دیگر هم به برنامه اضافه می شود که در مثال توضیح داده شده است:

📎**مثال:**

نمونه های تایپ Person که در آرایه skills آنها مقدار "Java" وجود داشته باشد:

search Person (skills include "Java")

- **نوع داده زمانی:** برنامه باید علاوه بر انواع داده های گفته شده، از نوع داده زمان هم پشتیبانی کند.

نوع داده	مثال	مقدار پیشفرض	معادل جاوا
time	2025-03-26T12:30:00	زمان فعلی سیستم	LocalDateTime

عملگر های < = > باید برای این نوع داده هم عملکرد صحیح داشته باشند.

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

- **فرمت خروجی:** هر میزان که خروجی به صورت مرتب و بهتری نمایش داده شود، نمره امتیازی دریافت میکنید، نمونه ای از خروجی مطلوب جدولی را در زیر مشاهده میکنید.

id	name	age	email	skills
1	Ali	25	ali@example.com	["Java", "Python"]

همچنین می توانید خروجی ها را با فرمت آبجکت های JSON نمایش دهید

- **پشتیبانی از nested JSON:** برنامه شما باید از JSON های تو در تو پشتیبانی کند. برای درک بهتر به مثال ها توجه کنید

📎مثال:

```
create Person {
  "id": {"type": "int", "unique": true},
  "address": {
    "city": {"type": "string"},
    "street": {"type": "string"}
  }
}
```

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

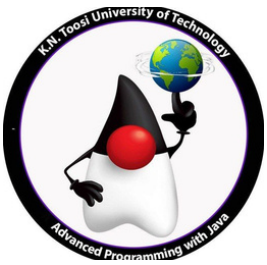
📎مثال:

```
insert Person {
    "id": 1,
    "address": {
      "city": "Tehran",
      "street": "Azadi"
    }
  }
```

در صورت پیاده سازی فیلتر بر اساس JSON های تو در تو، نمره امتیازی جداگانه در نظر گرفته می شود

📎مثال:

```
search Person (address.city = "Tehran")
```



برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

- **فیلتر های چند شرطی:** برنامه شما باید برای دستور های حاوی فیلتر، از شرط های با بیش از یک پارامتر هم پشتیبانی کند، تعداد این پارامتر ها میتواند هر عددی باشد و باید نمونه هایی که در همه این پارامتر ها صدق می کنند قبول شوند. ساختار کلی دستورات به این صورت تغییر می کند:

command type (parameter1, ...) {JSON-like input}

📎مثال:

update Person (name = “ali” , age < 30) { "age": 30, “isStudent”:true }

- **مقایسه فیلد های مختلف:** فیلتر های برنامه علاوه بر مقایسه هر فیلد با مقادیر ثابت، از مقایسه دو فیلد مختلف از یک تایپ با یکدیگر هم پشتیبانی کند. برای درک بهتر به مثال توجه کنید.

📎مثال:

search Student (first_grade < second_grade)

مقادیر first_grade و second_grade فیلد هایی از نوع عدد صحیح از تایپ Student هستند. این فیلتر همه دانشجویانی که نمره دومشان از نمره اول بیشتر باشد را بر میگرداند.

❌خطا ها: مقایسه دو فیلد از نوع های غیر قابل مقایسه (عدد با رشته، رشته با بولین، عدد با بولین)

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

- **عملگرهای جمع و تفاضل:** برنامه باید علاوه بر عملگرهای قبلی از عملگر های جمع و تفاضل هم پشتیبانی کند.

عملگر	توضیح	مثال
+ , -	جمع یا تفاضل	age + 10 - 6 > 20 + 4

این عملگر ها فقط بر روی داده های عددی (اعداد صحیح و اعشاری) کار می کنند و اگر برای نوع های دیگر استفاده شوند باید خطا با متن مناسب نمایش داده شود. همچنین دقت کنید در طرفین عملگرهای مقایسه ای هر تعداد دلخواه عملگر جمع یا تفاضل می تواند قرار گیرد.

برنامه سازی پیشرفته

دکتر اثنی عشری،مهندس زمانیان

بهار 1404

منابع مفید

- https://www.w3schools.com/js/js_json_intro.asp
- https://www.baeldung.com/java-hashmap
- https://www.geeksforgeeks.org/arraylist-in-java/
- https://bito.ai/resources/nested-json-example-json-explained/
- https://regex101.com/
- https://www.baeldung.com/java-8-date-time-intro