Task 1: Advanced Data Structures

Here is your task

Your task is to implement a novel data structure - your project lead is calling it a power of two max heap. The rest of your team is doing their best to come up with a better name. The requirements of the data structure are as follows:

- The heap must satisfy the heap property.
- Every parent node in the heap must have 2^x children.
- The value of x must be a parameter of the heap's constructor.
- The heap must implement an insert method.
- The heap must implement a pop max method.
- The heap must be implemented in Java.
- The heap must be performant.
- You must use a more descriptive variable name than x in your implementation.

Think carefully about how you implement each method, and manage the underlying data. Performance is critical, so keep cycles and memory usage to a minimum. Be sure to test your heap with very small and very large values of x. As always, keep a weather eye out for sneaky edge cases.

Solution

```java
import java.util.Arrays;
import java.util.NoSuchElementException;

public class PowerHeap {
    private double x;
    private int size;
    private int[] heapArray;

    // Constructor
    public PowerHeap(double x, int capacity) {
        this.size = 0;
        heapArray = new int[capacity + 1];
        this.x = x;
        Arrays.fill(heapArray, -1);
    }

    private int parent(int i) {
        return (int) ((i - 1) / Math.pow(2, x));
    }
```

```
    public boolean isFull() {
        return size == heapArray.length;
    }

    public void insert(int value) {
        if (isFull()) {
            throw new NoSuchElementException("Heap is full, no space to insert
new element.");
        } else {
            heapArray[size++] = value;
            heapifyUp(size - 1);
        }
    }

    private void heapifyUp(int i) {
        int tmp = heapArray[i];
        while (i > 0 && tmp > heapArray[parent(i)]) {
            heapArray[i] = heapArray[parent(i)];
            i = parent(i);
        }
        heapArray[i] = tmp;
    }

    public int popMax() {
        int maxItem = heapArray[0];
        heapArray[0] = heapArray[size - 1];
        heapArray[size - 1] = -1;
        size--;

        int i = 0;
        while (i < size - 1) {
            heapifyUp(i);
            i++;
        }

        return maxItem;
    }

    public void print() {
        for (int i = 0; i < size; i++) {
            System.out.print(heapArray[i]);
            System.out.print(',');
        }
        System.out.println();
```

```
    }

    public static void main(String[] args) {
        double x = 2; // Example value for x
        int capacity = 10; // Example capacity

        PowerHeap heap = new PowerHeap(x, capacity);
        heap.insert(5);
        heap.insert(10);
        heap.insert(3);

        heap.print();

        int maxItem = heap.popMax();
        System.out.println("Max item: " + maxItem);

        heap.print();
    }
}
```

Output

```
PS C:\Users\gaura\java\forge\heap> cd "c:\Users\gaura\ja
10,5,3,
Max item: 10
3,5,
PS C:\Users\gaura\java\forge\heap\src>
```

Task 2: Software Architecture

Here is your task

Your task is to draft a UML class diagram describing the data processors for a pipeline. The component responsible for reading in input data is being designed by another engineer, so you only need to worry about what happens to the data when it reaches your processor. You may assume three classes already exist:

Datapoint: this class represents both raw and processed data points. Any time data moves between methods you may use this class as an abstraction.
ModeIdentifier: an enum used to identify a processor mode.
DatabaseIdentifier: an enum used to identify a database connection.

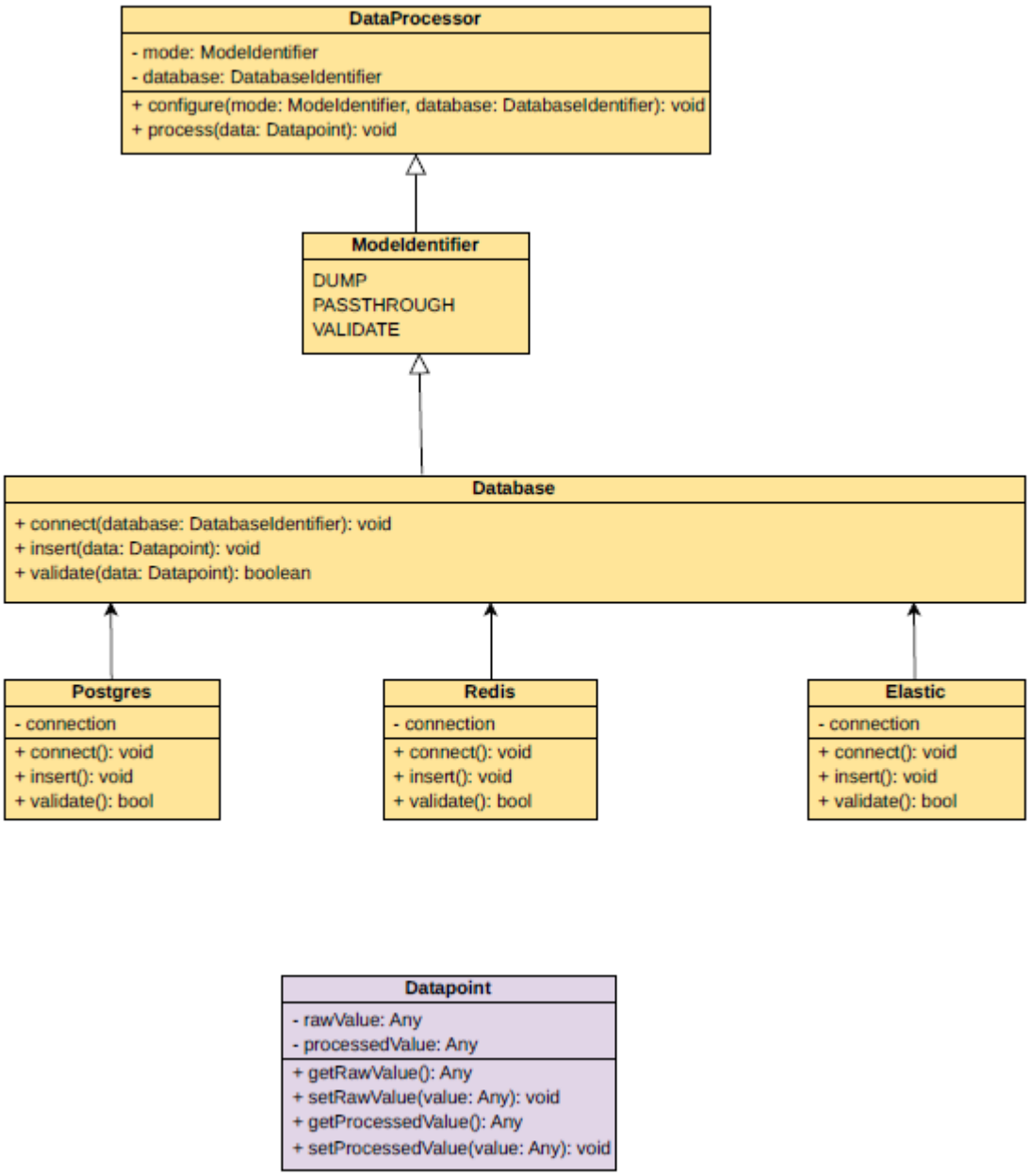
Here are the requirements for your design:

- The processor must implement a configure method that accepts a ModeIdentifier and a DatabaseIdentifier as parameters.
    - This method is called to change the operating mode of the processor, and/or select the current database.
- The processor must be able to change between the following modes:
    - Dump mode: simply drops the data.
    - Passthrough mode: inserts the data into the currently configured database.
    - Validate mode: validates the data, then inserts it (both operations are carried out on the currently configured database).
- The processor must be able to swap between the following databases. Each database will require a different implementation to insert and validate data
    - Postgres.
    - Redis.
    - Elastic.
- The processor must implement a process method that accepts a DataPoint as a parameter.
    - This method will have different behavior depending on the currently configured mode and database.

There is no need to get into implementation specifics, keep things abstract and high level. For example, you need only specify connect, insert, and validate methods for each database, there is no need to specify how those methods go about performing their verbs. The point of this task is to think about how code is structured.

When you’re finished, convert your class diagram to a PDF and submit it below. You'll see an example answer on the next step, but we encourage you to give it a go first!

Solution


DataProcessor
- mode: ModelIdentifier
- database: DatabaseIdentifier
+ configure(mode: ModelIdentifier, database: DatabaseIdentifier): void
+ process(data: Datapoint): void
ModelIdentifier
DUMP
PASSTHROUGH
VALIDATE
Database
+ connect(database: DatabaseIdentifier): void
+ insert(data: Datapoint): void
+ validate(data: Datapoint): boolean
Postgres
- connection
+ connect(): void
+ insert(): void
+ validate(): bool
Redis
- connection
+ connect(): void
+ insert(): void
+ validate(): bool
Elastic
- connection
+ connect(): void
+ insert(): void
+ validate(): bool
Datapoint
- rawValue: Any
- processedValue: Any
+ getRawValue(): Any
+ setRawValue(value: Any): void
+ getProcessedValue(): Any
+ setProcessedValue(value: Any): void

Task 3: Relational Database Design

Here is your task

Your task is to draft a UML class diagram describing the data processors for a pipeline. The component responsible for reading in input data is being designed by another engineer, so you only need to worry about what happens to the data when it reaches your processor.

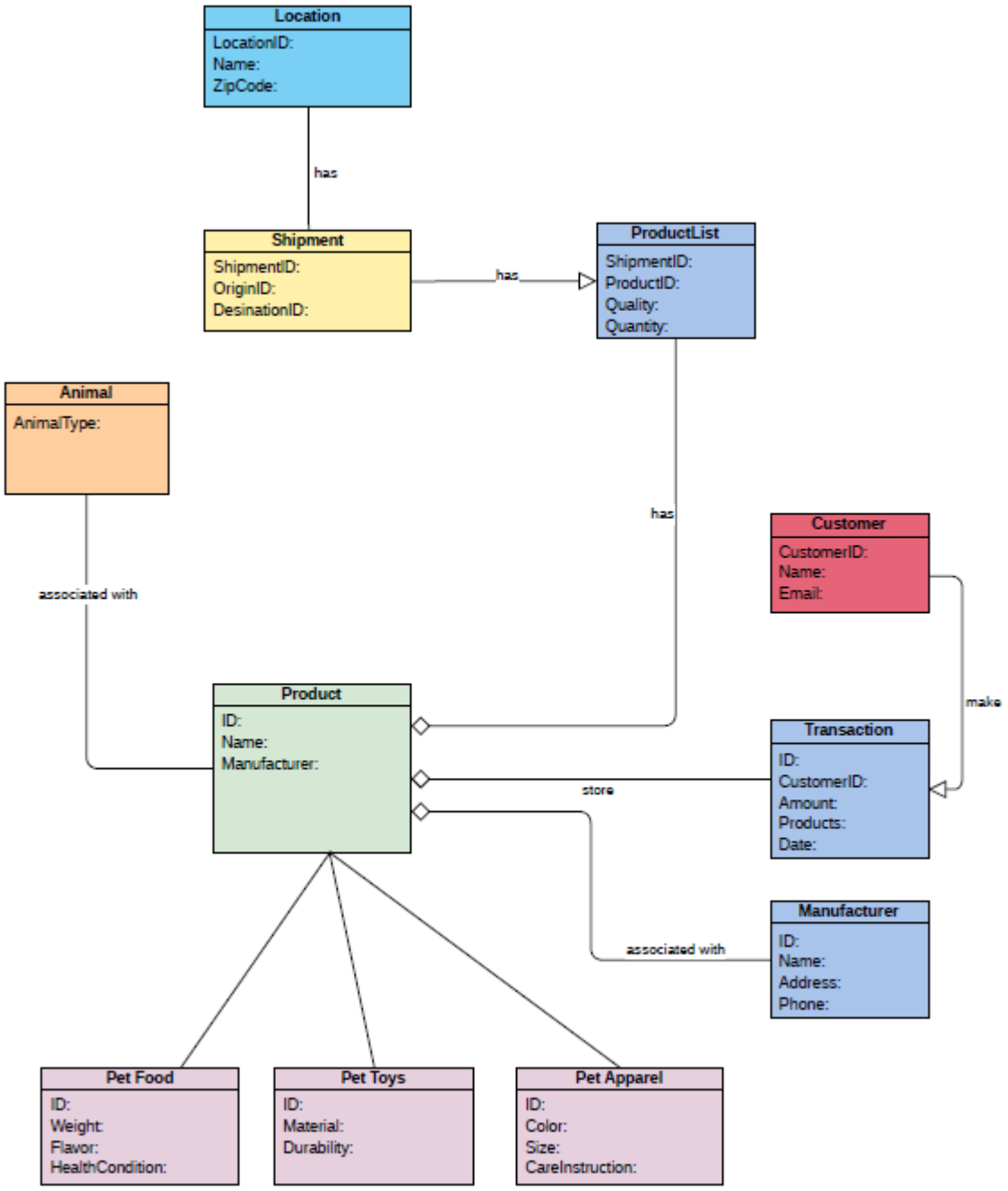
You may assume three classes already exist:

1. Datapoint: this class represents both raw and processed data points. Any time data moves between methods you may use this class as an abstraction.
2. ModeIdentifier: an enum used to identify a processor mode.
3. DatabaseIdentifier: an enum used to identify a database connection.

Your task is to draft an ERD for an appropriately normalized relational database that satisfies these requirements:

- The database should store information related to the following products.
    - Pet food, which has a name, manufacturer, weight, flavor, and target health condition.
    - Pet toys, which have an associated material, name, manufacturer, and durability.
    - Pet apparel, which has a color, manufacturer, size, name, and specific care instructions.
- Each product should be associated with one or more animals.
- Each product should be associated with a manufacturer.
- The database should track customers and their transactions.
    - It should store customer names and email addresses.
    - Customers can make transactions to purchase one or more products.
    - Each transaction should store the date and the products involved.
- The database should track shipments to various Walmart locations.
    - Each location should be represented by a name and a zip code.
    - A shipment is recorded as an origin, a destination, and a collection of products, each with an associated quantity.

Solution


Location
LocationID:
Name:
ZipCode:
has
Shipment
ShipmentID:
OriginID:
DesinationID:
has
ProductList
ShipmentID:
ProductID:
Quality:
Quantity:
Animal
AnimalType:
has
Customer
CustomerID:
Name:
Email:
associated with
make
Product
ID:
Name:
Manufacturer:
Transaction
ID:
CustomerID:
Amount:
Products:
Date:
store
Manufacturer
ID:
Name:
Address:
Phone:
associated with
Pet Food
ID:
Weight:
Flavor:
HealthCondition:
Pet Toys
ID:
Material:
Durability:
Pet Apparel
ID:
Color:
Size:
CareInstruction:

Task 4: Data Munging

Here is your task

Part 1: Get the data

First, you need to get your hands on the relevant data. The shipping department has been kind enough to provide you with a repository containing all of their spreadsheets, as well as a copy of the sqlite database. First, fork and clone the repository at: https://github.com/theforage/forage-walmart-task-4

Part 2: Populate the database

Your task is to insert all of the data contained in the provided spreadsheets into the SQLite database. You will write a Python script which:

- Reads each row from the spreadsheets.
- Extracts the relevant data.
- Munges it into a format that fits the database schema.
- Inserts the data into the database.

Spreadsheet 0 is self contained and can simply be inserted into the database, but spreadsheets 1 and 2 are dependent on one another. Spreadsheet 1 contains a single product per row, you will need to combine each row based on its shipping identifier, determine the quantity of goods in the shipment, and add a new row to the database for each product in the shipment. The origin and destination for each shipment in spreadsheet 1 are contained in spreadsheet 2. You may assume that all the given data is valid - product names are always spelled the same way, quantities are positive, etc.

Solution

```
import csv
import sqlite3

def create_tables(cursor):
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS shipping_data_0 (
            origin_warehouse TEXT,
            destination_store TEXT,
            product TEXT,
            on_time TEXT,
            product_quantity INTEGER,
```

```
            driver_identifier TEXT
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS shipping_data_1 (
            shipment_identifier TEXT,
            product TEXT,
            on_time TEXT,
            origin_warehouse TEXT,
            destination_store TEXT
        )
    """)

def insert_shipping_data_0(cursor):
    with open('data/shipping_data_0.csv', 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)
        for row in csv_reader:
            origin_warehouse, destination_store, product, on_time,
product_quantity, driver_identifier = row
            cursor.execute("INSERT INTO shipping_data_0 (origin_warehouse,
destination_store, product, on_time, product_quantity, driver_identifier) VALUES
(?, ?, ?, ?, ?, ?)",
                           (origin_warehouse, destination_store, product,
on_time, product_quantity, driver_identifier))

def insert_shipping_data_2(cursor):
    with open('data/shipping_data_2.csv', 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)
        shipping_data_2_rows = [row for row in csv_reader]

    with open('data/shipping_data_1.csv', 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)
        for row in csv_reader:
            shipment_identifier, product, on_time = row
            matching_rows = [r for r in shipping_data_2_rows if r[0] ==
shipment_identifier]
            if matching_rows:
                origin_warehouse, destination_store, driver_identifier =
matching_rows[0][1], matching_rows[0][2], matching_rows[0][3]
                cursor.execute("INSERT INTO shipping_data_1 (shipment_identifier,
product, on_time, origin_warehouse, destination_store) VALUES (?, ?, ?, ?, ?)",
```

```
                                (shipment_identifier, product, on_time,
origin_warehouse, destination_store))

if __name__ == "__main__":
    conn = sqlite3.connect('shipment_database.db')
    cursor = conn.cursor()

    create_tables(cursor)  # Create the necessary tables

    insert_shipping_data_0(cursor)
    insert_shipping_data_2(cursor)

    conn.commit()
    conn.close()
```