Assignment 3

Question 1

Write an OOPs program to build a calculator for basic arithmetic operations. The calculator accepts only integer inputs. In certain cases, other characters may be entered as input. Implement the calculator such that if the input entered is not a digit[0-9], then it should continue with the appropriate exception handling. Also, handle the incorrect operation “division by zero” in your code.

Input format:
3 lines of input.
1st line is the operator. 2nd line 1st argument. 3rd line 2nd argument.
Output format:
Print result if operation is valid. Else, show the appropriate exception.
Sample Input and Output:
Input:
+ 5 7
Output:
12
Input:
/ 10 0
Output:
ArithmeticException: Division by zero handled. Enter a valid argument.

Question 2

Consider a checkerboard of size 8x8. Each square in the board can be defined as i*j where i and j ranges from 1 to 8. You are given the initial position on the board along with some movements. Write a Java program to find the final position after the movements. If the final position goes beyond the chessboard, handle the exception and print the message “Overflow”. Otherwise, print the final position as [i,j].

Input Format :

First line specifies the initial position.(row_number column_number)
Second line defines ‘N’ (the number of movements).
Third line has the ‘N’ movements separated by space. (Movements are Left, Right, Up, Down)

Output Format :

One line of output which shows the final position if it exists, otherwise print “Overflow”.

Sample Input and Output:

Input:

```
53
4
Up 2 Right 3 Down 5 Left 1
```

Output :

```
[8,5]
```

Input :

```
31
1 Left 2
```

Output :

```
Overflow
```

Question 3

Create a class “Queue” with two operations enqueue() and dequeue(). Use enqueue() to insert elements into a queue and dequeue() to delete elements from the queue. During execution, handle the underflow exception condition by printing “EmptyQueue”.

Input Format :

One line input:

- If the operation is enqueue() then add the given element to the queue list.
- If the operation is dequeue() then delete the element from the queue list.

Output Format :

One line output:

- If the operation is enqueue(), print “Success”.
- If the operation is dequeue(), print the element, if an exception occurs then prin“EmptyQueue”.

Sample Input and Output :

Note: E indicates enqueue and D indicates dequeue

Input:

```
E 10 E 34 E 60 D
D D D
```

Output:

```
Success Success Success 10
34
 60 EmptyQueue
```

Question 4

Create a class ‘Book’ that represents a book in a library. The class should have the following properties:

- title (String)
- author (String)
- price (double)
- yearPublished (int)

Your class should have:

1. A constructor that initializes all the properties of the class.
2. A method displayDetails() that prints the details of the book.

A main method in another class named ‘Library’ where you create three different Book objects using the constructor and then call the displayDetails() method for each object. Create a Java program for this scenario.

<u>Sample Input:</u>

```
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Price: $10.99
Year Published: 1925

Title: To Kill a Mockingbird
Author: Harper Lee
Price: $7.99
Year Published: 1960

Title: 1984
Author: George Orwell
Price: $8.99
Year Published: 1949
```

<u>Sample output:</u>

```
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Price: $10.99
Year Published: 1925

Title: To Kill a Mockingbird
Author: Harper Lee
Price: $7.99
Year Published: 1960

Title: 1984
Author: George Orwell
Price: $8.99
Year Published: 1949
```

Question 5

Design a class named ‘Sensor’ that represents a generic sensor in an IoT system. The class should have the following properties:

- sensorId: a unique identifier for the sensor.
- sensorType: a String representing the type of sensor (e.g., "temperature", "humidity").
- isActive: a boolean indicating whether the sensor is active.

Whenever an instance of the Sensor class is created, it should automatically generate a unique sensorId, set the isActive property to true, and require the sensorType to be provided during initialization.
Implement the constructor for the Sensor class to achieve this functionality. Also include a Display() function to print the details of the sensor.
Create a Java program to add 2 sensors and print their details .

Sample Input:

```
sensorType: temparature
sensorType: humidity
```

Sample Output:

```
sensorId: 4754876398
sensorType: temparature
isActive: true

sensorId: 3423567
sensorType:  humidity
isActive: true
```

Question 6

University Management System

Create a menu driven program for a university management system. The program should have the following classes:

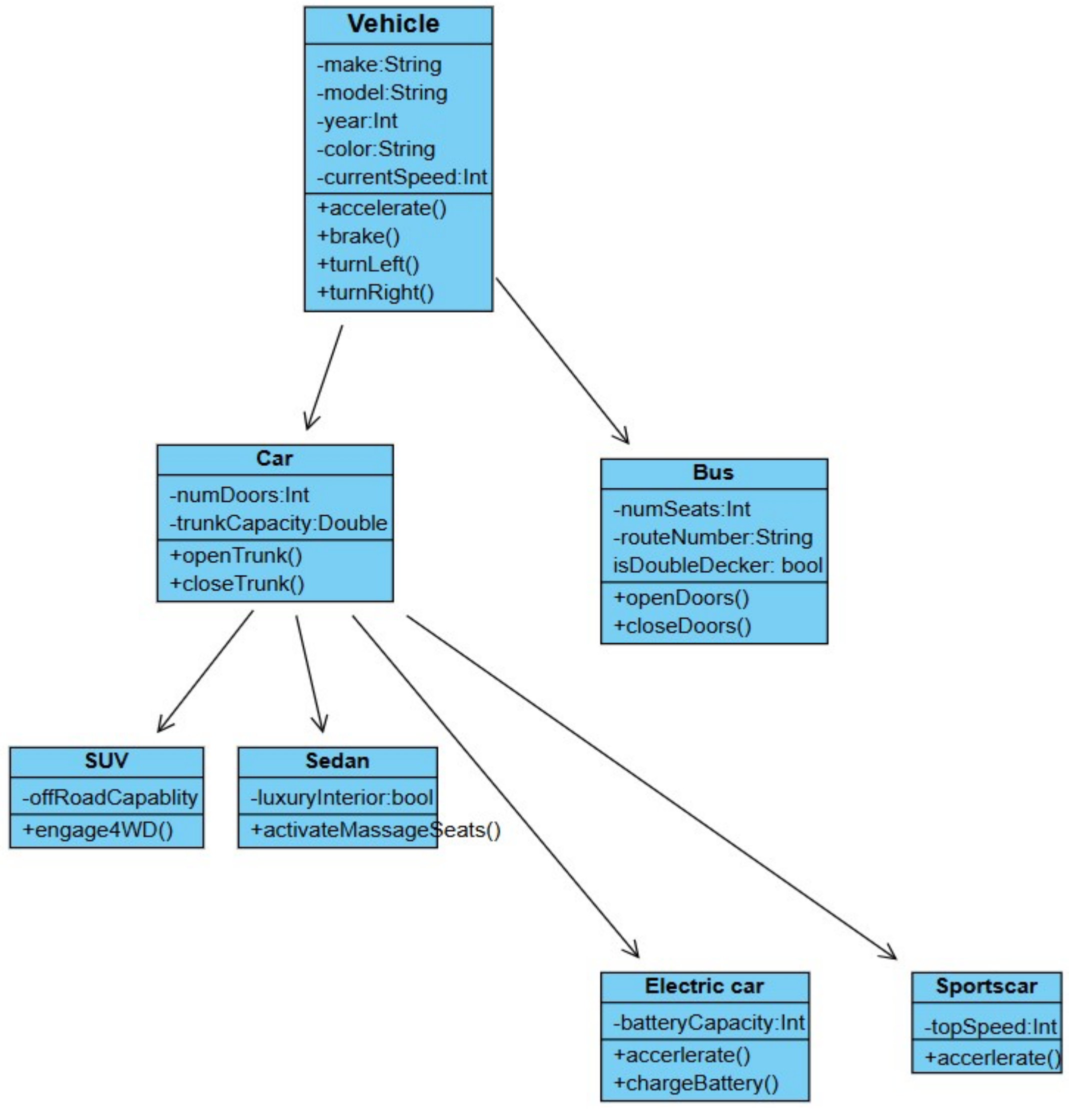
- University
- Department
- Professor
- Student

Requirements:

1. The University class should contain a list of departments.
2. The Department class should contain a list of professors and a list of students.
3. The Professor class should contain information about the professor (e.g., name, id, department).
4. The Student class should contain information about the student (e.g., name, id, major).

Question 7

Design a vehicle management system that involves multiple levels of inheritance, as well as multiple classes inheriting from the same base class. The system includes a **Vehicle** class as the base class, and several derived classes like **Car, Bus**. Additionally, extend the **Car** class further into more specialized classes like **ElectricCar** and **SportsCar**. You also have scenarios where different classes like **SUV** and **Sedan** inherit from the **Car** class.
Class Diagram:



Question 8
In a company, the employee management system is designed to manage different types of employees, such as officers and managers. The system needs to store common details about employees like name, age, address, phone number, and salary, as well as specific details about officers and managers. Officers have a specialization field, and managers have a department field.
Requirements:

1. **Common Employee Details:**
   - The system should allow for retrieving and updating common details of any employee.
   - There should be a method to calculate the annual salary of an employee.

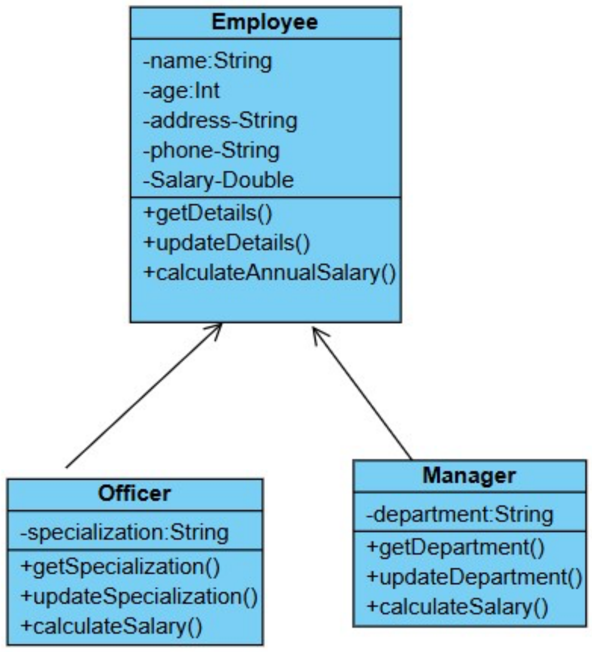
2. Officer Specific Details:

- The system should manage the specialization field specific to officers.
- There should be a method to calculate the salary of an officer, factoring in their specialization.

3. Manager Specific Details:

- The system should manage the department field specific to managers.
- There should be a method to calculate the salary of a manager, factoring in their department.

Class Diagram:



Question 9

Write a Java program for a simple banking system. Create the following classes:

- Account - A base class with the following members:
    - Private field accountNumber (String)
    - Protected field balance (double)
    - Public method deposit(double amount) which adds the amount to balance.
    - Public method getBalance() which returns the current balance.
- SavingsAccount - A subclass of Account with an additional feature:
    - Protected field interestRate (double)
    - Public method addInterest() which adds interest to the balance based on interestRate.

Instantiate a SavingsAccount object, deposit an amount, add interest, and print the balance. Explain how access specifiers affect visibility in this hierarchy.

Question 10

Write a Java program to demonstrate how access specifiers affect access from different packages. Create two packages: **company** and department. In the company package, define the following classes:

- Employee - A class with:
    - Protected field name (String)
    - Public method getName() to get the name.
- Manager - A subclass of Employee with:
    - Public method printName() to print the name.

In the department package, define a class **Office** which tries to access and print the **name** field from an Employee instance and show how the **name** field can be accessed and modified in the Manager class.