**Business Case Scenario: Customer Contact Validation**

**In this scenario, you are developing a system where customer details are entered. Every customer must provide at least one contact number (either mobile or landline). If the customer does not provide any contact number, the system will throw a user-defined exception.**

1. **Define the Customer class**: Create a class to represent the customer's information (name, mobile number, and landline number).
2. **Create a custom exception**: Define a user-defined exception, for example, ContactNumberNotProvidedException, which will be thrown if both contact numbers are missing.
3. **Validate the contact numbers**: Inside the Customer class or another validation class, check if at least one contact number is provided. If neither is provided, throw the custom exception.
4. **Catch and handle the exception**: In the main() method, handle the custom exception using try-catch.
5. **Provide feedback**: Display a message to inform the user of the error.

**Step-by-Step Implementation:**

1. **Create the Customer class**:
   * Define fields for the customer’s name, mobile number, and landline number.
   * Create a constructor to initialize these fields.
2. **Define the custom exception class (ContactNumberNotProvidedException)**:
   * Extend the Exception class to create your custom exception.
3. **Validate customer contact information**:
   * Create a method to validate whether at least one contact number is provided. Throw the custom exception if both fields are empty.
4. **Catch and handle the exception**:
   * Use a try-catch block in the main() method to handle the custom exception when it is thrown.

**Code Implementation:**

**1. Custom Exception Class:**

java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Custom exception class

class ContactNumberNotProvidedException extends Exception {

public ContactNumberNotProvidedException(String message) {

super(message);

}

}

**2. Customer Class:**

java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class Customer {

private String name;

private String mobileNumber;

private String landlineNumber;

public Customer(String name, String mobileNumber, String landlineNumber) {

this.name = name;

this.mobileNumber = mobileNumber;

this.landlineNumber = landlineNumber;

}

// Method to validate the contact numbers

public void validateContactNumber() throws ContactNumberNotProvidedException {

if ((mobileNumber == null || mobileNumber.isEmpty()) &&

(landlineNumber == null || landlineNumber.isEmpty())) {

throw new ContactNumberNotProvidedException("Customer must provide at least one contact number!");

}

}

@Override

public String toString() {

return "Customer{name='" + name + "', mobileNumber='" + mobileNumber + "', landlineNumber='" + landlineNumber + "'}";

}

}

**3. Main Class:**

java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class Main {

public static void main(String[] args) {

try {

// Creating a customer with no contact numbers

Customer customer = new Customer("John Doe", "", "");

customer.validateContactNumber(); // Validate contact numbers

} catch (ContactNumberNotProvidedException e) {

System.out.println("Exception caught: " + e.getMessage());

}

}

}

**Sample Output:**

sql

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exception caught: Customer must provide at least one contact number!

Customer details are valid: Customer{name='Jane Doe', mobileNumber='9876543210', landlineNumber=''}

**Steps to Follow for the Students:**

1. **Understand the Problem**:
   * The system should enforce that every customer has at least one contact number (either mobile or landline).
2. **Create a Custom Exception**:
   * Define a class that extends Exception for handling the "missing contact number" case.
3. **Throw the Custom Exception**:
   * Write validation logic in the Customer class. If both contact numbers are empty or null, throw the custom exception.
4. **Use try-catch Blocks**:
   * In the main() method, attempt to create and validate a Customer. Handle the exception using try-catch to provide user feedback when the validation fails.
5. **Test the Solution**:
   * Test the validation with different input scenarios (e.g., both numbers missing, only mobile number provided).