Self Practice - Day 08 - Exception Handling

1. Question 1

Problem Statement:

You are developing a Java application for a bookstore that manages book inventory. As part of the application, you need to handle various exceptions that may occur during the inventory management process. Design and implement exception handling for the following scenarios:

1. **InputMismatchException Handling:** When the user inputs data for book quantity, there is a possibility of encountering an InputMismatchException if the input provided is not a valid integer. Implement exception handling to catch and handle this exception gracefully. Display an error message informing the user about the incorrect input format and prompt them to enter the quantity again.
2. **NumberFormatException Handling:** When processing book prices, there is a risk of encountering a NumberFormatException if the price entered by the user cann be parsed as a valid decimal number. Implement exception handling to catch and handle this exception. Display an error message indicating that the price format invalid and prompt the user to enter the price again.
3. **ArrayIndexOutOfBoundsException Handling:** During the inventory update process, there is a possibility of encountering an ArrayIndexOutOfBoundsException if the user attempts to access an array element with an invalid index. Implement exception handling to catch and handle this exception. Display an error message indicating that the specified index is out of bounds and prompt the user to enter a valid index.

Analysis:

1. **InputMismatchException Handling:**
   - Use a try-catch block to catch the InputMismatchException.
   - Display an error message informing the user about the incorrect input format.
   - Prompt the user to enter the quantity again.
2. **NumberFormatException Handling:**
   - Use a try-catch block to catch the NumberFormatException.
   - Display an error message indicating that the price format is invalid.
   - Prompt the user to enter the price again.
3. **ArrayIndexOutOfBoundsException Handling:**
   - Use a try-catch block to catch the ArrayIndexOutOfBoundsException.
   - Display an error message indicating that the specified index is out of bounds.
   - Prompt the user to enter a valid index.

Code:

```java
import java.util.InputMismatchException;

public class Bookstore {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] inventory = {10, 20, 30, 40, 50};

        try {
            System.out.print("Enter the book quantity: ");
            int quantity = scanner.nextInt();
            System.out.println("Book quantity: " + quantity);
        } catch (InputMismatchException e) {
            System.out.println("Error: Invalid input format. Please enter a valid integer.");
            scanner.nextLine(); // Clear the input buffer
        }

        try {
            System.out.print("Enter the book price: ");
            double price = Double.parseDouble(scanner.next());
            System.out.println("Book price: $" + price);
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid price format. Please enter a valid decimal number.");
        }

        try {
            System.out.print("Enter the index to update inventory: ");
            int index = scanner.nextInt();
            inventory[index] = 0;
            System.out.println("Inventory updated successfully.");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Specified index is out of bounds. Please enter a valid index.");
        }
    }
}
```

Output:

```
Enter the book quantity: abc
Error: Invalid input format. Please enter a valid integer.
Enter the book price: 1sc
Error: Invalid price format. Please enter a valid decimal number.
Enter the index to update inventory: 10
Error: Specified index is out of bounds. Please enter a valid index.
```

---

2. Question 2

Problem Statement:

You are developing a simple banking application to manage customer accounts. One of the requirements is to ensure that the withdrawal amount from an account do not exceed the available balance. Implement a custom exception called InsufficientBalanceException to handle cases where the withdrawal amount exceeds the available balance. Your task is to modify the existing Account class to include exception handling for withdrawals.

Your implementation should adhere to the following specifications:

1. Define a custom exception class named InsufficientBalanceException that extends the Exception class. This class should have a parameterized constructor that accepts a message string.
2. Modify the Account class to include exception handling for withdrawals:
    - When a withdrawal is attempted, check if the withdrawal amount is greater than the available balance.
    - If the withdrawal amount exceeds the available balance, throw an InsufficientBalanceException with an appropriate error message.
    - If the withdrawal amount is valid, deduct the amount from the available balance.
3. In the main method or a separate testing class, create an instance of the Account class with an initial balance. Test the withdrawal functionality by attempting to withdraw an amount that exceeds the available balance. Handle the InsufficientBalanceException appropriately by displaying an error message

Analysis:

1. **Custom Exception Class:**
    - Define a custom exception class named InsufficientBalanceException that extends the Exception class.
    - Implement a parameterized constructor that accepts a message string.
2. **Account Class Modification:**
    - Modify the Account class to include exception handling for withdrawals.
    - Check if the withdrawal amount exceeds the available balance.
    - If the withdrawal amount is greater than the available balance, throw an InsufficientBalanceException with an appropriate error message.
    - If the withdrawal amount is valid, deduct the amount from the available balance.
3. **Testing:**
    - Create an instance of the Account class with an initial balance.
    - Test the withdrawal functionality by attempting to withdraw an amount that exceeds the available balance.

- Handle the InsufficientBalanceException appropriately by displaying an error message.

Code:

```
class Account {
    private double balance;

    public Account(double balance) {
        this.balance = balance;
    }

    public void withdraw(double amount) throws InsufficientBalanceException {
        if (amount > balance) {
            throw new InsufficientBalanceException("Insufficient balance. Cannot withdraw $" + amount);
        } else {
            balance -= amount;
            System.out.println("Withdrawal successful. Remaining balance: $" + balance);
        }
    }
}


class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String message) {
        super(message);
    }
}

public class Main {
    public static void main(String[] args) {
        Account account = new Account(100);

        try {
            account.withdraw(150);
        } catch (InsufficientBalanceException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Insufficient balance. Cannot withdraw $150.0
```

---

3. Question 3

Problem Statement:

Define an employee class with properties Employee code, name, date of birth and date of appointment. The Employee code must be a positive integer number.

- Write a java program to read the above details and validate the employee code. If the employee code is not in the format specified , then raise an exception called InvalidEmpNumberException.
- Verify if the date of birth is before the data of appointment. If it is not so then raise an exception called InvalidDateOfJoinException. If it is correct, then create the Employee object and display the details of employees and the number of years of experience

Analysis:

1. **Employee Class:**
    - Define an Employee class with properties Employee code, name, date of birth, and date of appointment.

   - The Employee code must be a positive integer number.
2. **Input Validation:**
   - Read the employee details and validate the employee code.
   - If the employee code is not in the specified format, raise an InvalidEmpNumberException.
   - Verify if the date of birth is before the date of appointment.
   - If the date of birth is not before the date of appointment, raise an InvalidDateOfJoinException.
3. **Employee Object Creation:**
   - If the input is valid, create an Employee object with the provided details.
   - Display the details of the employee and calculate the number of years of experience.

Code:

```java
import java.util.Scanner;
import java.time.Period;

public class Employee {
    private int empCode;
    private String name;
    private String dob;
    private String doj;

    public Employee(int empCode, String name, String dob, String doj) {
        this.empCode = empCode;
        this.name = name;
        this.dob = dob;
        this.doj = doj;
    }

    public void displayDetails() {
        System.out.println("Employee Code: " + empCode);
        System.out.println("Name: " + name);
        System.out.println("Date of Birth: " + dob);
        System.out.println("Date of Appointment: " + doj);
    }

    public void calculateExperience() {
        Period period = Period.between(LocalDate.parse(dob, DateTimeFormatter.ofPattern("dd/MM/yyyy")), LocalDate.parse(doj, Date
        System.out.println("Experience: " + period.getYears() + " years");
    }
}

class InvalidEmpNumberException extends Exception {
    public InvalidEmpNumberException(String message) {
        super(message);
    }
}

class InvalidDateOfJoinException extends Exception {
    public InvalidDateOfJoinException(String message) {
        super(message);
    }
}


public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee Code: ");
        int empCode = scanner.nextInt();
        scanner.nextLine(); // Clear the input buffer
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Date of Birth (dd/MM/yyyy): ");
        String dob = scanner.nextLine();
        System.out.print("Enter Date of Appointment (dd/MM/yyyy): ");
        String doj = scanner.nextLine();

        try {
            if (empCode <= 0) {
                throw new InvalidEmpNumberException("Invalid Employee Code. Employee code must be a positive integer.");
            }

            LocalDate dobDate = LocalDate.parse(dob, DateTimeFormatter.ofPattern("dd/MM/yyyy"));
            LocalDate dojDate = LocalDate.parse(doj, DateTimeFormatter.ofPattern("dd/MM/yyyy"));

            if (dobDate.isAfter(dojDate)) {
                throw new InvalidDateOfJoinException("Invalid Date of Appointment. Date of birth must be before Date of Appointme
            }

            Employee employee = new Employee(empCode, name, dob, doj);
            employee.displayDetails();
            employee.calculateExperience();
        } catch (InvalidEmpNumberException | InvalidDateOfJoinException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Enter Employee Code: -11
Enter Name: Shabari
Enter Date of Birth (dd/MM/yyyy): 20-11-2003
Enter Date of Appointment (dd/MM/yyyy): 14-05-2024
Invalid Employee Code. Employee code must be a positive integer.
```

---

4. Question 4

Problem Statement:

Create a java class for handling an Exception called ‘PayOutOfBoundsException’ and throw the exception when the transaction amount exceeds the limit or the amour is insufficient. (Maximum transaction limit is 30000).Create a class called ‘AccountManagement’ with two methods named ‘checkForDebit’ and ‘withdrawAmount’ that uses PayOutOfBoundsException.(Keep Current balance as 80000).

Analysis:

1. **PayOutOfBoundsException:**
    - Create a custom exception class called PayOutOfBoundsException that extends the Exception class.
    - Implement a parameterized constructor that accepts a message string.
2. **AccountManagement Class:**
    - Create a class named AccountManagement with two methods: checkForDebit and withdrawAmount.
    - checkForDebit method should check if the transaction amount exceeds the limit or the amount is insufficient and throw PayOutOfBoundsException.
    - withdrawAmount method should handle the PayOutOfBoundsException and display an appropriate error message.
3. **Testing:**
    - Create an instance of the AccountManagement class with a current balance of 80000.
    - Test the withdrawAmount method by attempting to withdraw an amount that exceeds the limit or is insufficient.

Code:

```
public class AccountManagement {
    private double currentBalance = 80_00_000.00;
    private final double _Max_Transaction_Limit = 30_000.00;

    public void checkForDebit(double amount ) throws PayOutOfBoundsException{
        if (amount >= _Max_Transaction_Limit || amount >= currentBalance) {
            throw new PayOutOfBoundsException("Transaction amount exceeds the limit or insufficient balance.");
        }
    }

    public void withdraw(double amount){
        try{
            checkForDebit(amount);
            currentBalance -= amount;
        } catch (PayOutOfBoundsException e) {
            System.out.println(e.getMessage());
        }
    }

}

class PayOutOfBoundsException extends Exception {
    public PayOutOfBoundsException(String message) {
        super(message);
    }
}

public class Main {
    public static void main(String[] args) {
        AccountManagement accountManagement = new AccountManagement();
        accountManagement.withdraw(40000);
    }
}
```

Output:

```
Transaction amount exceeds the limit or insufficient balance.
```

---

5. Question 5

Problem Statement:

Create a class called Invoice that a Grocery store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables—a part number (type integer), a part description (type String), quantity of the item being purchased (type integer) and a price per item (double). Provide method constructor with four arguments. Write a test application to create an instance and validate the input obtained using Scanner object. Ensure that part number is value greater than 0, part description is not null string, quantity of the item and price per item is value greater than 0.

> **Note :** The InputMismatchException is thrown when attempting to retrieve a value using the Scanner class that doesn't match the expected pattern or type

Analysis:

1. **Invoice Class:**
    - Create an Invoice class with instance variables for part number, part description, quantity, and price per item.
    - Implement a constructor that accepts four arguments to initialize the instance variables.
2. **Input Validation:**
    - Write a test application to create an instance of the Invoice class and validate the input obtained using the Scanner object.
    - Ensure that the part number is greater than 0, the part description is not a null string, the quantity of the item and price per item are greater than 0.
3. **Exception Handling:**

- Handle the InputMismatchException that is thrown when attempting to retrieve a value using the Scanner class that doesn't match the expected pattern or type.
- Display an error message indicating that the input format is invalid and prompt the user to enter the correct value.

Code:

```java
import java.util.Scanner;
import java.util.InputMismatchException;

public class Invoice {
    private int partNumber;
    private String partDescription;
    private int quantity;
    private double pricePerItem;

    public Invoice(int partNumber, String partDescription, int quantity, double pricePerItem) {
        this.partNumber = partNumber;
        this.partDescription = partDescription;
        this.quantity = quantity;
        this.pricePerItem = pricePerItem;
    }

    public void displayInvoice() {
        System.out.println("Part Number: " + partNumber);
        System.out.println("Part Description: " + partDescription);
        System.out.println("Quantity: " + quantity);
        System.out.println("Price Per Item: $" + pricePerItem);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter Part Number: ");
            int partNumber = scanner.nextInt();
            scanner.nextLine(); // Clear the input buffer
            System.out.print("Enter Part Description: ");
            String partDescription = scanner.nextLine();
            System.out.print("Enter Quantity: ");
            int quantity = scanner.nextInt();
            System.out.print("Enter Price Per Item: ");
            double pricePerItem = scanner.nextDouble();

            if (partNumber <= 0 || partDescription.isEmpty() || quantity <= 0 || pricePerItem <= 0) {
                throw new InputMismatchException("Invalid input. Please enter valid values.");
            }

            Invoice invoice = new Invoice(partNumber, partDescription, quantity, pricePerItem);
            invoice.displayInvoice();
        } catch (InputMismatchException e) {
            System.out.println("Error: Invalid input format. Please enter valid values.");
        }
    }
}
```

Output:

```
Enter Part Number: 1293q
Error: Invalid input format. Please enter valid values.
```