

C-DAC Mumbai

OOPJ Lab Assignment

Problem 1: Salary Split

Scenario: You are developing a payroll system for a company. The HR department wants to distribute a bonus equally among employees in a department. However, sometimes a department might have zero employees due to restructuring.

Task: Create a method that divides a bonus amount among employees and handles the case when the number of employees is zero.

Sample Input:

10000
0

Expected Output:

Error: Division by zero not allowed

Problem 2: Exam Scores

Scenario: A teacher is using a digital gradebook system to access student scores. Sometimes they might accidentally try to access the score of a student number that doesn't exist in the class roster.

Task: Create a program that stores exam scores in an array and safely accesses student scores by index.

Sample Input:

3
78 90 85
5

Expected Output:

Invalid index accessed

```
System.out.println("Enter the Number of Students: ");
int n = in.nextInt();
System.out.println("Enter the marks: ");
int[] marks = new int[n];
for(int i = 0; i < marks.length; i++){
    marks[i] = in.nextInt();
}

int index = in.nextInt();
try{
    System.out.println(marks[index]);
} catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Invalid index accessed.");
}
```

Problem 3: Age Input

Scenario: A registration form for an online course asks for the user's age. Sometimes users accidentally enter text instead of numbers, causing the system to crash.

Task: Create a registration system that safely converts age input from string to integer.

Sample Input:

eighteen

Expected Output:

Invalid number format

```
try{
    int age = Integer.parseInt(input);
    System.out.println("Age is: " + age);
} catch(NumberFormatException e){
    System.out.println("Invalid Number Format");
}
```

Problem 4: Employee Data

Scenario: An HR system needs to calculate hourly wage by dividing an employee's salary by working hours. The system must handle both invalid employee indices and division by zero.

Task: Create a method with nested try-catch blocks to handle multiple exception scenarios.

Sample Input:

2
5000 6000
0
5

Expected Output:

Division by zero
or
Invalid index

Problem 5: Online Shopping

Scenario: An e-commerce platform processes orders by calculating the total price (quantity \times unit price). The system needs to handle invalid quantities and accessing non-existent products.

Task: Create an order processing method that handles multiple exception types.

Sample Input:

0
3
299.99 499.99 199.99
5

Expected Output:

Arithmetic Exception caught
or
Array Index Exception

Problem 6: Age Restriction

Scenario: A professional workshop registration system only allows participants who are 18 years or older. The system needs a custom exception for age validation.

Task: Create a custom exception class and use it to validate user age during registration.

Sample Input:

16

Expected Output:

AgeNotValidException: Age must be \geq 18

Problem 7: Student List

Scenario: A school management system tries to load a student list from a file at the beginning of each semester. Sometimes the file might not exist or be corrupted.

Task: Simulate file reading operation and handle FileNotFoundException.

Sample Input:

student_list.txt

Expected Output:

File not found

Problem 8: Payment Processing

Scenario: A payment gateway system processes transactions and needs to clean up database connections (just a scenario, database knowledge not required) regardless of whether the payment succeeds or fails.

Task: Create a payment processing method that uses finally block for cleanup operations.

Sample Input:

(No input required)

Expected Output:

Exception occurred: Payment failed

Cleanup done

```
try{
    throw new Exception("Payment failed.");
} catch(Exception e) {
    SOP("Exception occur." + e.getMessage());
} finally {
    System.out.println("Cleanup done");
}
```

Problem 9: Marks Validation

Scenario: An online examination system needs to validate that marks entered by teachers are within valid range (0-100). Negative marks should not be allowed.

Task: Create a marks validation method that throws an exception for invalid marks.

Sample Input:

-5

Expected Output:

Invalid marks

```
try{
    if(marks > 0 && marks <= 100) {
        System.out.println("Valid Marks")
    } else {
        throw new Exception("Invalid marks.");
    }
} catch(Exception e) {
    System.out.println(e.getMessage());
}
```

Problem 10: Greeting Message

Scenario: A learning management system generates personalized greeting messages for students. The system starts with a basic greeting and adds course-specific information.

Task: Use StringBuilder to create a personalized greeting message.

Sample Input:

Initial Text: Hello

Text to insert: CDAC

Insert Index: 6

Text to append: Java Student

Expected Output:

Hello CDAC Java Student

```
StringBuilder sb = new StringBuilder(in.nextLine());
String textToInsert = in.nextLine();
int index = in.nextInt(); in.nextLine();
String appendText = in.nextLine();
try {
    sb.insert(index, " " + textToInsert);
    sb.append(" ").append(appendText);
    System.out.println(sb);
} catch (StringIndexOutOfBoundsException e) {
    System.out.println("Invalid index for insertion."); }
}
```

Problem 11: Notification Update

Scenario: A university notification system needs to update announcements when exam schedules change. The system should efficiently replace old information with new information.

Task: Use StringBuilder to update notification messages.

Sample Input:

Original text: Exam postponed

Text to find: postponed

Replacement Text: rescheduled

Expected Output:

Exam rescheduled

```
StringBuilder sb = new StringBuilder(in.nextLine()); //original text
System.out.println("Text to Find: ");
String textToFind = in.nextLine();
System.out.println("Replacement Text: ");
String replaceText = in.nextLine();
String result = sb.toString().replace(textToFind, replaceText);
sb = new StringBuilder(result);
System.out.println(sb);
```

Problem 12: Remove Extra Text

Scenario: An automated message system sometimes adds extra text that needs to be removed before sending messages to students.

Task: Use StringBuilder to clean up message content.

Sample Input:

Original Text: Please read - Do not disturb

Exact substring to delete: - Do not disturb

Expected Output:

Please read

Problem 13: Order Number Display

Scenario: An e-commerce system generates invoice numbers and needs to display them in reverse order for verification purposes.

Task: Use StringBuilder to reverse order numbers.

Sample Input:

INV2025

Expected Output:

5202VNI

Problem 14: Report Title

Scenario: A report generation system needs to modify document titles by adding department names and updating formatting.

Task: Use StringBuilder method chaining to efficiently modify report titles.

Sample Input:

Original title: Annual Report

Department Name: CDAC

Expected Output:

Annual CDAC Report

Problem 15: Meeting Notification

Scenario: A corporate meeting scheduler needs to build complete meeting notifications by adding time and location details to basic meeting announcements.

Task: Use StringBuffer to create detailed meeting notifications.

Sample Input:

Base text: Meeting:

Text to append: Friday at 5 PM

Expected Output:

Meeting: Friday at 5 PM

Problem 16: Room Allocation Update

Scenario: A facility management system assigns rooms to different activities and needs to insert building information into existing room numbers.

Task: Use StringBuffer to update room allocation information.

Sample Input:

Original text: 101

Text to insert: New Building

Insert index: 0

Expected Output:

New Building 101

Problem 17: Remove Outdated Information

Scenario: An academic system maintains course information that includes year details. When information becomes outdated, the year needs to be removed.

Task: Use StringBuffer to remove outdated information.

Sample Input:

Original text: CDAC Kharghar 2024

Exact substring to delete: 2024

Expected Output:

CDAC Kharghar

Problem 18: Ticket Number Verification

Scenario: A ticketing system generates verification codes by reversing ticket numbers for security purposes.

Task: Use StringBuffer to create verification codes.

Sample Input:

12345

Expected Output:

54321

```
String ticketNumber = "12345";  
StringBuffer sb = new StringBuffer(ticketNumber);  
String verificationCode = sb.reverse().toString();
```

Problem 19: Message Update System

Scenario: A communication system needs to update message status from "Old Notice" to "Updated Notice" when information is refreshed.

Task: Use StringBuffer to update message status.

Sample Input:

Original text: Old Notice

Text to find: Old

Replacement text: Updated

Expected Output:

Updated Notice

Problem 20: Bank Account Security

Scenario: A banking system needs to ensure that once a bank account ID is assigned, it cannot be changed for security and audit purposes.

Task: Demonstrate the use of final variables in a banking context.

Sample Input:

Account ID: 101

Expected Output:

Account ID = 101 (cannot be changed)

Problem 21: Data Processing Cleanup

Scenario: A data processing system handles user form submissions and must always close database connections and clean up resources, whether the processing succeeds or fails.

Task: Use finally block to ensure proper resource cleanup.

Sample Input:

(No input required)

Expected Output:

Exception occurred: Invalid input

Data processing completed

Problem 22: Student Object Cleanup

Scenario: A student management system creates student objects during registration. When these objects are no longer needed, the system should clean up resources before garbage collection.

Task: Override finalize method to demonstrate cleanup during garbage collection.

Sample Input:

Student Name: Amit

Expected Output:

Student object for Amit is being garbage collected

Problem 23: Employee Age Management

Scenario: An HR system stores employee ages in a database. The system needs to convert primitive int values to Integer objects for database storage and collection operations.

Task: Demonstrate autoboxing by converting primitive int to Integer object.

Sample Input:

30

Expected Output:

Integer object: 30

`int obj = 30;
Integer object = obj;`

Problem 24: Salary Calculation

Scenario: A payroll system retrieves employee ages from a database as Integer objects but needs primitive int values for mathematical calculations.

Task: Demonstrate unboxing by extracting primitive values from wrapper objects.

Sample Input:

25

Expected Output:

int value: 25

Autoboxing:
Integer ageObject = 25;
int age = ageObject;

Problem 25: Payment Processing

Scenario: An e-commerce system receives payment amounts as strings from web forms and needs to convert them to integers for financial calculations.

Task: Parse string input to integer and perform calculations.

Sample Input:

Amount as string: 1000

Additional amount to add: 500

Expected Output:

1000 + 500 = 1500

int amount = Integer.parseInt(amountStr);
int total = amount + additionalAmount;

Problem 26: Salary Storage

Scenario: A financial system needs to convert primitive double salary values to Double objects for storage in collections and database operations.

Task: Use valueOf method to convert primitives to wrapper objects.

Sample Input:

45000.5

Expected Output:

Double object: 45000.5

double salary = in.nextDouble();
Double salaryObject = Double.valueOf(salary);
