OOPJ Lab Assignment-8

ANSWERS

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

```java
import java.util.Scanner;

public class SalarySplit {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int bonus = sc.nextInt();
        int employees = sc.nextInt();

        try {
            int share = bonus / employees;
            System.out.println("Each employee gets: " + share);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero not allowed");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac SalarySplit.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java SalarySplit
5000
0
```

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

```java
import java.util.Scanner;

public class ExamScores {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int scores[] = new int[n];
        for (int i = 0; i < n; i++) {
            scores[i] = sc.nextInt();
        }
        int index = sc.nextInt();

        try {
            System.out.println("Score: " + scores[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid index accessed");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac ExamScores.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java ExamScores
4
65 78 90 81
6
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

```java
import java.util.Scanner;

public class AgeInput {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String ageStr = sc.next();

        try {
            int age = Integer.parseInt(ageStr);
            System.out.println("Age: " + age);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac AgeInput.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java AgeInput
twelve
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

```java
import java.util.Scanner;

public class EmployeeData {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int salaries[] = new int[n];
        for (int i = 0; i < n; i++) {
            salaries[i] = sc.nextInt();
        }
        int index = sc.nextInt();
        int hours = sc.nextInt();

        try {
            try {
                int wage = salaries[index] / hours;
                System.out.println("Hourly wage: " + wage);
            } catch (ArithmeticException e) {
                System.out.println("Division by zero");
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid index");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac EmployeeData.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java EmployeeData
2
5000 6000
0
5
Hourly wage: 1000
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac EmployeeData.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java EmployeeData
3
1000 2000 3000
3
0
Invalid index
```

Problem 5: Online Shopping
Scenario: An e-commerce platform processes orders by calculating the total price (quantity × 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

```java
import java.util.Scanner;

public class OnlineShopping {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int quantity = sc.nextInt();
        int n = sc.nextInt();
        double prices[] = new double[n];
        for (int i = 0; i < n; i++) {
            prices[i] = sc.nextDouble();
        }
        int index = sc.nextInt();

        try {
            double total = quantity * prices[index];
            System.out.println("Total price: " + total);
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception caught");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Exception");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac OnlineShopping.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java OnlineShopping
0
3
39.99 49.99 19.99
4
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 ≥ 18

```java
import java.util.Scanner;

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

public class AgeRestriction {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int age = sc.nextInt();

        try {
            if (age < 18) {
                throw new AgeNotValidException("Age must be ≥ 18");
            }
            System.out.println("Registration successful");
        } catch (AgeNotValidException e) {
            System.out.println("AgeNotValidException: " + e.getMessage());
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac AgeRestriction.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java AgeRestriction
14
AgeNotValidException: Age must be ≥ 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

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class StudentList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String fileName = sc.next();

        try {
            Scanner fileScanner = new Scanner(new File(fileName));
            System.out.println("File loaded successfully");
        } catch (FileNotFoundException e) {
            System.out.println("File not found");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac StudentList.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java StudentList
student_list.txt
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

```java
public class PaymentProcessing {
    public static void main(String[] args) {
        try {
            throw new Exception("Payment failed");
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.getMessage());
        } finally {
            System.out.println("Cleanup done");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac PaymentProcessing.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java PaymentProcessing
Exception occurred: Payment failed
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

```java
import java.util.Scanner;

public class MarksValidation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int marks = sc.nextInt();

        try {
            if (marks < 0 || marks > 100) {
                throw new Exception("Invalid marks");
            }
            System.out.println("Marks entered: " + marks);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac MarksValidation.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java MarksValidation
-20
Invalid marks
```

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

```java
import java.util.Scanner;

public class GreetingMessage {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String initial = sc.nextLine();
        String insertText = sc.nextLine();
        int index = sc.nextInt();
        sc.nextLine();
        String appendText = sc.nextLine();

        StringBuilder sb = new StringBuilder(initial);
        sb.insert(index, " " + insertText);
        sb.append(" " + appendText);

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac GreetingMessage.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java GreetingMessage
Namaste
INDIA
7
Kaise Ho
Namaste INDIA Kaise Ho
```

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

```java
import java.util.Scanner;

public class NotificationUpdate {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String original = sc.nextLine();
        String find = sc.nextLine();
        String replace = sc.nextLine();

        StringBuilder sb = new StringBuilder(original);
        int index = sb.indexOf(find);
        if (index != -1) {
            sb.replace(index, index + find.length(), replace);
        }

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac NotificationUpdate.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java NotificationUpdate
Assignment Pending
Pending
Completed
Assignment Completed
```

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

```java
import java.util.Scanner;

public class RemoveExtraText {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String original = sc.nextLine();
        String toDelete = sc.nextLine();

        StringBuilder sb = new StringBuilder(original);
        int index = sb.indexOf(toDelete);
        if (index != -1) {
            sb.delete(index, index + toDelete.length());
        }

        System.out.println(sb.toString().trim());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac RemoveExtraText.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java RemoveExtraText
Boarding - Delayed
- Delayed
Boarding
```

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

```java
import java.util.Scanner;

public class OrderNumberDisplay {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String order = sc.nextLine();

        StringBuilder sb = new StringBuilder(order);
        sb.reverse();

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac OrderNumberDisplay.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java OrderNumberDisplay
INV123456
654321VNI
```

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

```java
import java.util.Scanner;

public class ReportTitle {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String original = sc.nextLine();
        String dept = sc.nextLine();

        StringBuilder sb = new StringBuilder(original);
        sb.insert(original.indexOf("Report"), dept + " ");

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac ReportTitle.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java ReportTitle
Weekly Report
Attendance
Weekly Attendance 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

```java
import java.util.Scanner;

public class MeetingNotification {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String base = sc.nextLine();
        String appendText = sc.nextLine();

        StringBuffer sb = new StringBuffer(base);
        sb.append(" " + appendText);

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac MeetingNotification.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java MeetingNotification
Meeting:
Saturday at 12 PM
Meeting: Saturday at 12 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

```java
import java.util.Scanner;

public class RoomAllocationUpdate {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String original = sc.nextLine();
        String insertText = sc.nextLine();
        int index = sc.nextInt();

        StringBuffer sb = new StringBuffer(original);
        sb.insert(index, insertText + " ");

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac RoomAllocationUpdate.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java RoomAllocationUpdate
25
Room
0
Room 25
```

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

```java
import java.util.Scanner;

public class RemoveOutdatedInformation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String original = sc.nextLine();
        String toDelete = sc.nextLine();

        StringBuffer sb = new StringBuffer(original);
        int index = sb.indexOf(toDelete);
        if (index != -1) {
            sb.delete(index, index + toDelete.length());
        }

        System.out.println(sb.toString().trim());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac RemoveOutdatedInformation.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java RemoveOutdatedInformation
CDAC Kharghar 2024
2024
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

```java
import java.util.Scanner;

public class TicketNumberVerification {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String ticket = sc.nextLine();

        StringBuffer sb = new StringBuffer(ticket);
        sb.reverse();

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac TicketNumberVerification.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java TicketNumberVerification
24680
08642
```

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

```java
import java.util.Scanner;

public class MessageUpdateSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String original = sc.nextLine();
        String find = sc.nextLine();
        String replace = sc.nextLine();

        StringBuffer sb = new StringBuffer(original);
        int index = sb.indexOf(find);
        if (index != -1) {
            sb.replace(index, index + find.length(), replace);
        }

        System.out.println(sb.toString());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac MessageUpdateSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java MessageUpdateSystem
Traffic Sign: Red
Red
Green
Traffic Sign: Green
```

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)

```java
public class BankAccountSecurity {
    public static void main(String[] args) {
        final int accountId = 101;
        System.out.println("Account ID = " + accountId + " (cannot be changed)");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac BankAccountSecurity.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java BankAccountSecurity
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

```java
public class DataProcessingCleanup {
    public static void main(String[] args) {
        try {
            throw new Exception("Invalid input");
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.getMessage());
        } finally {
            System.out.println("Data processing completed");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac DataProcessingCleanup.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java DataProcessingCleanup
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

```java
@SuppressWarnings("removal")
public class StudentObjectCleanup {

    static class Student {
        private String name;

        public Student(String name) {
            this.name = name;
        }

        @Override
        @SuppressWarnings("removal")
        protected void finalize() throws Throwable {
            try {
                System.out.println("Student object for " + name + " is being garbage collected");
            } finally {
                super.finalize();
            }
        }
    }

    public static void main(String[] args) {
        Student s1 = new Student("Amit");

        s1 = null;

        System.gc();

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac StudentObjectCleanup.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java StudentObjectCleanup
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

```java
import java.util.Scanner;

public class EmployeeAgeManagement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int age = sc.nextInt();

        Integer ageObj = age;
        System.out.println("Integer object: " + ageObj);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac EmployeeAgeManagement.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java EmployeeAgeManagement
45
Integer object: 45
```

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

```java
import java.util.Scanner;

public class SalaryCalculation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Integer ageObj = sc.nextInt();

        int age = ageObj;
        System.out.println("int value: " + age);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac SalaryCalculation.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java SalaryCalculation
20
int value: 20
```

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

```java
import java.util.Scanner;

public class PaymentProcessingConversion {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String amountStr = sc.nextLine();
        int add = sc.nextInt();

        int amount = Integer.parseInt(amountStr);
        System.out.println(amount + " + " + add + " = " + (amount + add));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac PaymentProcessingConversion.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java PaymentProcessingConversion
20000
10000
20000 + 10000 = 30000
```

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

```java
import java.util.Scanner;

public class SalaryStorage {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double salary = sc.nextDouble();

        Double salaryObj = Double.valueOf(salary);
        System.out.println("Double object: " + salaryObj);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 8> javac SalaryStorage.java
PS C:\Users\baenu\Test\OOPJ Assignment 8> java SalaryStorage
10.75
Double object: 10.75
```