**C-DAC Mumbai**

**OOPJ Lab**

**Assignment - 4**

**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.

->import java.util.Scanner;

public class SalarySplit {

static void distributeBonus(double bonus, int employees) {

try {

if (employees == 0) {

throw new ArithmeticException("Division by zero not allowed");

}

double bonusPerEmployee = bonus / employees;

System.out.println("Bonus per employee: " + bonusPerEmployee);

} catch (ArithmeticException e) {

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

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter total bonus amount: ");

double bonus = sc.nextDouble();

System.out.print("Enter number of employees: ");

int employees = sc.nextInt();

distributeBonus(bonus, employees);

sc.close();

}

}

**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.

->import java.util.Scanner;

public class ExamScores {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of students: ");

int n = sc.nextInt();

int[] scores = new int[n];

System.out.println("Enter scores:");

for (int i = 0; i < n; i++) {

scores[i] = sc.nextInt();

}

System.out.print("Enter student index to access: ");

int index = sc.nextInt();

try {

System.out.println("Score of student " + index + ": " + scores[index]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Invalid index accessed");

}

sc.close();

}

}

**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

->import java.util.Scanner;

public class AgeInput {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter your age: ");

String ageInput = sc.nextLine();

try {

int age = Integer.parseInt(ageInput);

System.out.println("Age entered: " + age);

} catch (NumberFormatException e) {

System.out.println("Invalid number format");

}

sc.close();

}

}

**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.

->

import java.util.Scanner;

public class EmployeeData {

static void calculateHourlyWage(double[] salaries, int index, int hours) {

try {

try {

double hourlyWage = salaries[index] / hours;

System.out.println("Hourly wage: " + hourlyWage);

} catch (ArithmeticException e) {

System.out.println("Division by zero");

}

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Invalid index");

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of employees: ");

int n = sc.nextInt();

double[] salaries = new double[n];

System.out.println("Enter salaries:");

for (int i = 0; i < n; i++) {

salaries[i] = sc.nextDouble();

}

System.out.print("Enter employee index: ");

int index = sc.nextInt();

System.out.print("Enter working hours: ");

int hours = sc.nextInt();

calculateHourlyWage(salaries, index, hours);

sc.close();

}

}

**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.

->import java.util.Scanner;

public class OnlineShopping {

static void processOrder(int quantity, double[] prices, int index) {

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");

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter quantity: ");

int quantity = sc.nextInt();

System.out.print("Enter number of products: ");

int n = sc.nextInt();

double[] prices = new double[n];

System.out.println("Enter product prices:");

for (int i = 0; i < n; i++) {

prices[i] = sc.nextDouble();

}

System.out.print("Enter product index: ");

int index = sc.nextInt();

processOrder(quantity, prices, index);

sc.close();

}

}

**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

->

import java.util.Scanner;

class AgeNotValidException extends Exception {

public AgeNotValidException(String message) {

super(message);

}

}

public class AgeRestriction {

static void validateAge(int age) throws AgeNotValidException {

if (age < 18) {

throw new AgeNotValidException("Age must be ≥ 18");

} else {

System.out.println("Registration Successful!");

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter age: ");

int age = sc.nextInt();

try {

validateAge(age);

} catch (AgeNotValidException e) {

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

}

sc.close();

}

}

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.

->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);

System.out.print("Enter file name: ");

String fileName = sc.nextLine();

try {

File file = new File(fileName);

Scanner fileReader = new Scanner(file); // may throw FileNotFoundException

System.out.println("File found! Reading data...");

while (fileReader.hasNextLine()) {

System.out.println(fileReader.nextLine());

}

fileReader.close();

} catch (FileNotFoundException e) {

System.out.println("File not found");

}

sc.close();

}

}

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.

->public class PaymentProcessing {

static void processPayment() {

try {

// Simulating payment failure

throw new Exception("Payment failed");

} catch (Exception e) {

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

} finally {

System.out.println("Cleanup done");

}

}

public static void main(String[] args) {

processPayment();

}

}

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.

->import java.util.Scanner;

public class MarksValidation {

static void validateMarks(int marks) throws Exception {

if (marks < 0 || marks > 100) {

throw new Exception("Invalid marks");

} else {

System.out.println("Marks accepted: " + marks);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter marks: ");

int marks = sc.nextInt();

try {

validateMarks(marks);

} catch (Exception e) {

System.out.println(e.getMessage());

}

sc.close();

}

}

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

->import java.util.Scanner;

public class GreetingMessage {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Initial Text: ");

String initial = sc.nextLine();

System.out.print("Text to insert: ");

String insertText = sc.nextLine();

System.out.print("Insert Index: ");

int index = sc.nextInt();

sc.nextLine(); // consume newline

System.out.print("Text to append: ");

String appendText = sc.nextLine();

StringBuilder sb = new StringBuilder(initial);

sb.insert(index, " " + insertText);

sb.append(" " + appendText);

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

sc.close();

}

}

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

->import java.util.Scanner;

public class NotificationUpdate {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Original text: ");

String original = sc.nextLine();

System.out.print("Text to find: ");

String find = sc.nextLine();

System.out.print("Replacement text: ");

String replace = sc.nextLine();

StringBuilder sb = new StringBuilder(original);

int start = sb.indexOf(find);

if (start != -1) {

sb.replace(start, start + find.length(), replace);

}

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

sc.close();

}

}

Problem 12: Remove Extra Text Scenario: An automated message system sometimes adds extra text that needs to be removed before sending messages to students.

->import java.util.Scanner;

public class RemoveExtraText {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Original Text: ");

String original = sc.nextLine();

System.out.print("Exact substring to delete: ");

String toDelete = sc.nextLine();

StringBuilder sb = new StringBuilder(original);

int start = sb.indexOf(toDelete);

if (start != -1) {

sb.delete(start, start + toDelete.length());

}

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

sc.close();

}

}

Problem 13: Order Number Display Scenario: An e-commerce system generates invoice numbers and needs to display them in reverse order for verification purposes.

->import java.util.Scanner;

public class OrderNumberDisplay {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter order number: ");

String order = sc.nextLine();

StringBuilder sb = new StringBuilder(order);

System.out.println(sb.reverse().toString());

sc.close();

}

}

Problem 14: Report Title Scenario: A report generation system needs to modify document titles by adding department names and updating formatting

->import java.util.Scanner;

public class ReportTitle {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Original title: ");

String title = sc.nextLine();

System.out.print("Department Name: ");

String dept = sc.nextLine();

StringBuilder sb = new StringBuilder(title);

int spaceIndex = sb.indexOf(" ");

sb.insert(spaceIndex, " " + dept);

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

sc.close();

}

}

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

->import java.util.Scanner;

public class MeetingNotification {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Base text: ");

String base = sc.nextLine();

System.out.print("Text to append: ");

String append = sc.nextLine();

StringBuffer sb = new StringBuffer(base);

sb.append(" " + append);

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

sc.close();

}

}

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.

->import java.util.Scanner;

public class RoomAllocationUpdate {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Original text: ");

String original = sc.nextLine();

System.out.print("Text to insert: ");

String insertText = sc.nextLine();

System.out.print("Insert index: ");

int index = sc.nextInt();

StringBuffer sb = new StringBuffer(original);

sb.insert(index, insertText + " ");

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

sc.close();

}

}

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

->import java.util.Scanner;

public class RemoveOutdatedInfo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Original text: ");

String original = sc.nextLine();

System.out.print("Exact substring to delete: ");

String deleteText = sc.nextLine();

StringBuffer sb = new StringBuffer(original);

int start = sb.indexOf(deleteText);

if (start != -1) {

sb.delete(start, start + deleteText.length());

}

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

sc.close();

}

}

Problem 18: Ticket Number Verification Scenario: A ticketing system generates verification codes by reversing ticket numbers for security purposes

->import java.util.Scanner;

public class TicketVerification {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter ticket number: ");

String ticket = sc.nextLine();

StringBuffer sb = new StringBuffer(ticket);

System.out.println(sb.reverse().toString());

sc.close();

}

}

Problem 19: Message Update System Scenario: A communication system needs to update message status from "Old Notice" to "Updated Notice" when information is refreshed

->import java.util.Scanner;

public class MessageUpdateSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Original text: ");

String original = sc.nextLine();

System.out.print("Text to find: ");

String find = sc.nextLine();

System.out.print("Replacement text: ");

String replace = sc.nextLine();

StringBuffer sb = new StringBuffer(original);

int start = sb.indexOf(find);

if (start != -1) {

sb.replace(start, start + find.length(), replace);

}

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

sc.close();

}

}

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

->public class BankAccountSecurity {

public static void main(String[] args) {

final int ACCOUNT\_ID = 101; // cannot be changed

System.out.println("Account ID = " + ACCOUNT\_ID + " (cannot be changed)");

// Uncommenting the below line will give a compile-time error

// ACCOUNT\_ID = 202;

}

}

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.

->public class DataProcessingCleanup {

static void processData() {

try {

// Simulate an error during data processing

throw new Exception("Invalid input");

} catch (Exception e) {

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

} finally {

System.out.println("Data processing completed");

}

}

public static void main(String[] args) {

processData();

}

}

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

->public class StudentObjectCleanup {

static class Student {

String name;

Student(String name) {

this.name = name;

}

@Override

protected void finalize() throws Throwable {

System.out.println("Student object for " + name + " is being garbage collected");

}

}

public static void main(String[] args) {

Student s1 = new Student("Amit");

System.out.println("Created student: " + s1.name);

// Make object eligible for garbage collection

s1 = null;

// Request Garbage Collection

System.gc();

// Give JVM some time to run GC (not guaranteed, but usually works for demo)

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

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.

->import java.util.Scanner;

public class EmployeeAgeManagement {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter employee age: ");

int age = sc.nextInt(); // primitive int

// Autoboxing: automatically converting int to Integer object

Integer ageObj = age;

System.out.println("Integer object: " + ageObj);

sc.close();

}

}

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.

->import java.util.Scanner;

public class SalaryCalculation {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter employee age (as Integer): ");

Integer ageObj = sc.nextInt(); // This will still be autoboxed into Integer

// Unboxing: converting Integer object to primitive int

int age = ageObj;

System.out.println("int value: " + age);

sc.close();

}

}

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

->import java.util.Scanner;

public class PaymentProcessing1 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Amount as string: ");

String amountStr = sc.nextLine();

System.out.print("Additional amount to add: ");

int additional = sc.nextInt();

try {

int amount = Integer.parseInt(amountStr); // Parsing String → int

int total = amount + additional;

System.out.println(amount + " + " + additional + " = " + total);

} catch (NumberFormatException e) {

System.out.println("Invalid number format");

}

sc.close();

}

}

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.

->import java.util.Scanner;

public class SalaryStorage {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter salary (double): ");

double salary = sc.nextDouble();

// Using valueOf to convert primitive double to Double object

Double salaryObj = Double.valueOf(salary);

System.out.println("Double object: " + salaryObj);

sc.close();

}

}