



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Experiment 1

Student Name: Swastik

Branch: BE-CSE

Semester: 6th

Subject Name: Project Based Learning
in Java with Lab

UID: 22BCS11755

Section/Group: 902_DL_B

Date of Performance: 11/01/2025

Subject Code: 22CSH-359

1. **Aim:** Given the following table containing information about employees of an organization, develop a small java application, which accepts employee id from the command prompt and displays the following details as output:

Emp No Emp Name Department Designation and Salary

You may assume that the array is initialized with the following details:

Emp No.	Emp Name	Join Date	Desig Code	Dept	Basic	HRA	IT
1001	Ashish	01/04/2009	e	R&D	20000	8000	3000
1002	Sushma	23/08/2012	c	PM	30000	12000	9000
1003	Rahul	12/11/2008	k	Acct	10000	8000	1000
1004	Chahat	29/01/2013	r	Front Desk	12000	6000	2000
1005	Ranjan	16/07/2005	m	Engg	50000	20000	20000
1006	Suman	1/1/2000	e	Manu facturing	23000	9000	4400
1007	Tanmay	12/06/2006	c	PM	29000	12000	10000

Salary is calculated as Basic+HRA+DA-IT. (DA details are given in the Designation table)

Designation details :

Designation Code	Designation	DA
e	Engineer	20000
c	Consultant	32000
k	Clerk	12000
r	Receptionist	15000
m	Manager	40000

Use Switch-Case to print Designation in the output and to find the value of DA for a particular employee.

2. Objective:

i. Assuming that your class name is Project1, and you execute your code as java Project1 1003, it should display the following output : Emp No. Emp Name Department Designation Salary

1003 Rahul Acct Clerk 29000 ii.

java Project1 123

There is no employee with empid : 123

3. Implementation/Code:

```
import java.util.*;
```

```
class Project {  
    public static void main(String[] args) {
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

```
String[] empNo = {"1001", "1002", "1003", "1004", "1005", "1006",  
"1007"};  
String[] empName = {"Ashish", "Sushma", "Rahul", "Chahat",  
"Ranjan", "Suman", "Tanmay"};  
String[] joinDate = {"01/04/2009", "23/08/2012", "12/11/2008",  
"29/01/2013", "16/07/2005", "1/1/2000", "12/06/2006"};  
char[] designationCode = {'e', 'c', 'k', 'r', 'm', 'e', 'c'};  
String[] department = {"R&D", "PM", "Acct", "Front Desk",  
"Engg", "Manufacturing", "PM"};  
  
int[] basic = {20000, 30000, 10000, 12000, 50000, 23000, 29000};  
int[] hra = {8000, 12000, 8000, 6000, 20000, 9000, 12000};  
int[] it = {3000, 9000, 1000, 2000, 20000, 4400, 10000};  
  
Scanner scanner = new Scanner(System.in);  
System.out.print("Enter Employee Number: ");  
String userInput = scanner.nextLine();  
scanner.close();  
  
boolean exist = false;  
int index = -1;  
  
for (int i = 0; i < empNo.length; i++) {  
    if (Objects.equals(empNo[i], userInput)) {  
        exist = true;  
        index = i;  
        break;  
    }  
}  
  
String empId = "", name = "", date = "", depart = "";  
char dCode = 'e';
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

```
int basicSalary = 0, h = 0, iT = 0, da = 0;  
String designation = "";
```

```
if (index == -1 && !exist) {  
    System.out.println("There is no employee with empid: " +  
userInput);  
} else {  
    empId = empNo[index];  
    name = empName[index];  
    date = joinDate[index];  
    dCode = designationCode[index];  
    depart = department[index];  
    basicSalary = basic[index];  
    h = hra[index];  
    iT = it[index];  
}  
  
switch (dCode) {  
    case 'e':  
        designation = "Engineer";  
        da = 20000;  
        break;  
  
    case 'c':  
        designation = "Consultant";  
        da = 32000;  
        break;  
  
    case 'k':  
        designation = "Clerk";  
        da = 12000;  
        break;
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

```
        case 'r':
            designation = "Receptionist";
            da = 15000;
            break;

        default:
            designation = "Manager";
            da = 40000;
    }

    int salary = basicSalary + h + da - iT;

    if (exist) {
        System.out.println("Emp No.   " + "Emp Name   " +
            "Department   " + "Designation   " + "Salary   ");
        System.out.println(" " + empId + "   " + name + "   " +
            depart + "   " + designation + "   " + salary);
    }
}
```



4. Output:

```
PS D:\free time> cd "d:\free time\" ; if ($?) { javac Project1.java } ; if ($?) { java Project1 }
Enter Employee ID: 1005
Emp No  Emp Name      Department      Designation      Salary
1005    Ranjan  Engg    Manager 90000
PS D:\free time> 
```

5. Learning Outcomes:

- i. Understand how to map employee details (like designation codes to roles) using efficient logic and structures.
- ii. Learn to identify and address input mismatches or invalid entries through proper validation and error messages.
- iii. Gain skills in presenting data in a well-structured and readable format for better user understanding.



Experiment 1.2

Student Name: Swastik

Branch: CSE

Semester: 6th

Subject: Java

UID: 22BCS11755

Section: 902_DL_B

DOP:13-01-2025

Subject Code: 22CSH-359

Aim: Design and implement a simple inventory control system for a small video rental store

Objective: To design and implement a user-friendly inventory control system for a small video rental store, enabling efficient management of video inventory, including functionalities for adding, renting, and returning videos.

Algorithm:

- **Define Classes:**

- **Video:** To represent each video, with attributes such as video ID, title, genre, and availability status.
- **Inventory:** To manage the list of videos, including adding and removing videos from the inventory.
- **Customer:** To represent customers, with attributes such as customer ID, name, and rented videos.
- **RentalSystem:** To control the process of renting and returning videos.

- **Video Class:**

- Define the video with attributes such as `videoID`, `title`, `genre`, and `isAvailable`.
- Define methods to mark the video as rented and returned.

- **Inventory Class:**

- Maintain a list of videos (`ArrayList<Video>`).
- Implement methods to add new videos, display available videos, and check if a video is available.

- **Customer Class:**

- Define a list to store rented videos.
- Implement methods to rent a video (if available) and return it.

- **RentalSystem Class:**

- Handle the main functionality: list available videos, allow customers to rent and return videos, and display the inventory status.

Code:

```
import java.util.ArrayList;
import java.util.Scanner;

class Video {
    private String title;
    private boolean isAvailable;

    public Video(String title) {
        this.title = title;
        this.isAvailable = true;
    }

    public String getTitle() {
        return title;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void rent() {
        if (isAvailable) {
            isAvailable = false;
        } else {
            System.out.println("Error: Video is already rented out.");
        }
    }

    public void returnVideo() {
        if (!isAvailable) {
            isAvailable = true;
        } else {
            System.out.println("Error: Video was not rented.");
        }
    }

    @Override
    public String toString() {
        return "Title: " + title + " | Available: " + (isAvailable ? "Yes" : "No");
    }
}

class VideoStore {
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
private ArrayList<Video> inventory;

public VideoStore() {
    inventory = new ArrayList<>();
}

public void addVideo(String title) {
    for (Video video : inventory) {
        if (video.getTitle().equalsIgnoreCase(title)) {
            System.out.println("Error: Video already exists in the inventory.");
            return;
        }
    }
    inventory.add(new Video(title));
    System.out.println("Video added successfully: " + title);
}

public void listInventory() {
    if (inventory.isEmpty()) {
        System.out.println("No videos in inventory.");
    } else {
        System.out.println("Inventory:");
        for (int i = 0; i < inventory.size(); i++) {
            System.out.println((i + 1) + ". " + inventory.get(i));
        }
    }
}

public void rentVideo(String title) {
    for (Video video : inventory) {
        if (video.getTitle().equalsIgnoreCase(title)) {
            if (video.isAvailable()) {
                video.rent();
                System.out.println("You rented: " + title);
            } else {
                System.out.println("Video is currently unavailable.");
            }
        }
    }
    return;
}

System.out.println("Error: Video not found in inventory.");
}

public void returnVideo(String title) {
    for (Video video : inventory) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        if (video.getTitle().equalsIgnoreCase(title)) {
            if (!video.isAvailable()) {
                video.returnVideo();
                System.out.println("You returned: " + title);
            } else {
                System.out.println("Error: Video was not rented.");
            }
            return;
        }
        System.out.println("Error: Video not found in inventory.");
    }
}

public class VideoRentalSystem {
    public static void main(String[] args) {
        VideoStore store = new VideoStore();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n--- Video Rental Store ---");
            System.out.println("1. Add Video");
            System.out.println("2. List Inventory");
            System.out.println("3. Rent Video");
            System.out.println("4. Return Video");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            int choice = -1;
            if (scanner.hasNextInt()) {
                choice = scanner.nextInt();
            } else {
                System.out.println("Invalid choice. Please enter a number.");
                scanner.next();
                continue;
            }
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter video title to add: ");
                    String titleToAdd = scanner.nextLine().trim();
                    store.addVideo(titleToAdd);
                    break;
                case 2:
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        store.listInventory();
        break;
    case 3:
        System.out.print("Enter video title to rent: ");
        String titleToRent = scanner.nextLine().trim();
        store.rentVideo(titleToRent);
        break;
    case 4:
        System.out.print("Enter video title to return: ");
        String titleToReturn = scanner.nextLine().trim();
        store.returnVideo(titleToReturn);
        break;
    case 5:
        System.out.println("Exiting the system. Goodbye!");
        scanner.close();
        return;
    default:
        System.out.println("Invalid choice. Please try again.");
    }
}
}
```

Output:

```
PS D:\Waheguru\java-ds\final\2D - Arrays> cd "d:\Waheguru\java-ds\final\2D - Arrays\" ; if ($?) { javac VideoRentalSystem.java
($?) { java VideoRentalSystem }

--- Video Rental Store ---
1. Add Video
2. List Inventory
3. Rent Video
4. Return Video
5. Exit
Enter your choice: 1
Enter video title to add: Joban
Video added successfully: Joban

--- Video Rental Store ---
1. Add Video
2. List Inventory
3. Rent Video
4. Return Video
5. Exit
Enter your choice: 2
Inventory:
1. Title: Joban | Available: Yes

--- Video Rental Store ---
1. Add Video
2. List Inventory
3. Rent Video
4. Return Video
5. Exit
Enter your choice: 3
Enter video title to rent: jobanjeet Singh
```

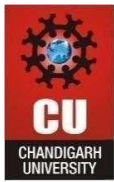


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Learning Outcomes:

- **Object-Oriented Design:** Learn to create and use classes for real-world entities.
- **Core Programming Skills:** Practice loops, conditionals, and methods for inventory operations.
- **Data Structure Usage:** Use `ArrayList` to manage dynamic data effectively.
- **User-Friendly Systems:** Design intuitive interfaces and handle errors smoothly.



Experiment 1.3

Name: Swastik

Branch: CSE

Semester: 6th

Subject: Java

UID: 22BCS11755

Section: 902_DL_B

DOP: 27/01/2025

Subject Code: 22CSH-359

1. Aim: Calculate interest based on the type of the account and the status of the account holder. The rates of interest changes according to the amount (greater than or less than 1 crore), age of account holder (General or Senior citizen) and number of days if the type of account is FD or RD.

2. Objective: The objective is to calculate interest for an account based on its type (e.g., FD, RD, Savings) and the account holder's status (General or Senior Citizen). The interest rate varies depending on factors such as the account balance (greater than or less than 1 crore), the account holder's age, and the duration (number of days) for fixed or recurring deposits. The system dynamically determines the applicable interest rate and computes the interest accordingly.

3. Algorithm:

1. Input: Account type, account holder status (General/Senior Citizen), account balance, and duration (if FD/RD).
2. Check Account Type:
3. If Savings Account:
4. Use predefined interest rates based on balance (>1 crore or <1 crore) and status.
5. If FD/RD:
6. Use interest rates based on duration, balance (>1 crore or <1 crore), and status.
7. Determine Interest Rate:
8. Fetch the applicable rate from the rate table based on balance, status, and duration (if applicable).
9. Calculate Interest:
10. For Savings Account:
11. $\text{Interest} = \text{Balance} \times \text{Interest Rate} \times \text{Duration (in years)}$
12. $\text{Interest} = \text{Balance} \times \text{Interest Rate} \times 1 \text{Duration (in years)}$
13. For FD/RD:
14. $\text{Interest} = \text{Principal} \times \text{Interest Rate} \times \text{Duration (in days)} / 365$
15. $\text{Interest} = \text{Principal} \times \text{Interest Rate} \times 365 \text{Duration (in days)}$
16. Output: Return the calculated interest.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Code:

```
import java.util.Scanner;

abstract class Account {
    double interestRate;
    double amount;

    Account(double amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Amount must be greater than zero.");
        }
        this.amount = amount;
    }

    abstract double calculateInterest();
}

// FDAccount class
class FDAccount extends Account {
    int noOfDays;
    int ageOfACHolder;

    FDAccount(double amount, int noOfDays, int ageOfACHolder) {
        super(amount);
        if (noOfDays <= 0 || ageOfACHolder <= 0) {
            throw new IllegalArgumentException("Number of days and age must be greater than zero.");
        }
        this.noOfDays = noOfDays;
        this.ageOfACHolder = ageOfACHolder;
    }

    @Override
    double calculateInterest() {
        if (amount < 1_00_00_000) {
            if (noOfDays >= 7 && noOfDays <= 14) interestRate = (ageOfACHolder >= 60) ? 5.00 : 4.50;
            else if (noOfDays >= 15 && noOfDays <= 29) interestRate = (ageOfACHolder >= 60) ? 5.25 : 4.75;
            else if (noOfDays >= 30 && noOfDays <= 45) interestRate = (ageOfACHolder >= 60) ? 6.00 : 5.50;
            else if (noOfDays >= 46 && noOfDays <= 60) interestRate = (ageOfACHolder >= 60) ? 7.50 : 7.00;
            else if (noOfDays >= 61 && noOfDays <= 184) interestRate = (ageOfACHolder >= 60) ? 8.00 : 7.50;
            else if (noOfDays >= 185 && noOfDays <= 365) interestRate = (ageOfACHolder >= 60) ? 8.50 : 8.00;
            else {
                if (noOfDays >= 7 && noOfDays <= 14) interestRate = 6.50;
                else if (noOfDays >= 15 && noOfDays <= 29) interestRate = 6.75;
                else if (noOfDays >= 30 && noOfDays <= 45) interestRate = 6.75;
                else if (noOfDays >= 46 && noOfDays <= 60) interestRate = 8.00;
                else if (noOfDays >= 61 && noOfDays <= 184) interestRate = 8.50;
                else if (noOfDays >= 185 && noOfDays <= 365) interestRate = 10.00;
            }
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}
    return amount * interestRate / 100;
}
}

// SBAccount class
class SBAccount extends Account {
    String accountType;

    SBAccount(double amount, String accountType) {
        super(amount);
        if (!accountType.equals("Normal") && !accountType.equals("NRI")) {
            throw new IllegalArgumentException("Invalid account type.");
        }
        this.accountType = accountType;
        this.interestRate = accountType.equals("Normal") ? 4.00 : 6.00;
    }

    @Override
    double calculateInterest() {
        return amount * interestRate / 100;
    }
}

// RDAccount class
class RDAccount extends Account {
    int noOfMonths;
    int ageOfACHolder;

    RDAccount(double amount, int noOfMonths, int ageOfACHolder) {
        super(amount);
        if (noOfMonths <= 0 || ageOfACHolder <= 0) {
            throw new IllegalArgumentException("Number of months and age must be greater than zero.");
        }
        this.noOfMonths = noOfMonths;
        this.ageOfACHolder = ageOfACHolder;
    }

    @Override
    double calculateInterest() {
        if (noOfMonths == 6) interestRate = (ageOfACHolder >= 60) ? 8.00 : 7.50;
        else if (noOfMonths == 9) interestRate = (ageOfACHolder >= 60) ? 8.25 : 7.75;
        else if (noOfMonths == 12) interestRate = (ageOfACHolder >= 60) ? 8.50 : 8.00;
        else if (noOfMonths == 15) interestRate = (ageOfACHolder >= 60) ? 8.75 : 8.25;
        else if (noOfMonths == 18) interestRate = (ageOfACHolder >= 60) ? 9.00 : 8.50;
        else if (noOfMonths == 21) interestRate = (ageOfACHolder >= 60) ? 9.25 : 8.75;
        return amount * interestRate / 100;
    }
}

// Main class to run the program
public class Main {
    public static void main(String[] args) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
Scanner scanner = new Scanner(System.in);
while (true) {
    System.out.println("Select the option:");
    System.out.println("1. Interest Calculator –SB");
    System.out.println("2. Interest Calculator –FD");
    System.out.println("3. Interest Calculator –RD");
    System.out.println("4. Exit");
    int choice = scanner.nextInt();

    if (choice == 4) break;

    try {
        switch (choice) {
            case 1:
                System.out.println("Enter the Average amount in your account:");
                double sbAmount = scanner.nextDouble();
                System.out.println("Enter the account type (Normal/NRI):");
                String accountType = scanner.next();
                SBAccount sbAccount = new SBAccount(sbAmount, accountType);
                System.out.println("Interest gained: Rs. " + sbAccount.calculateInterest());
                break;
            case 2:
                System.out.println("Enter the FD amount:");
                double fdAmount = scanner.nextDouble();
                System.out.println("Enter the number of days:");
                int noOfDays = scanner.nextInt();
                System.out.println("Enter your age:");
                int fdAge = scanner.nextInt();
                FDAccount fdAccount = new FDAccount(fdAmount, noOfDays, fdAge);
                System.out.println("Interest gained is: Rs. " + fdAccount.calculateInterest());
                break;
            case 3:
                System.out.println("Enter the RD amount:");
                double rdAmount = scanner.nextDouble();
                System.out.println("Enter the number of months:");
                int noOfMonths = scanner.nextInt();
                System.out.println("Enter your age:");
                int rdAge = scanner.nextInt();
                RDAccount rdAccount = new RDAccount(rdAmount, noOfMonths, rdAge);
                System.out.println("Interest gained is: Rs. " + rdAccount.calculateInterest());
                break;
            default:
                System.out.println("Invalid option. Please select a valid option.");
                break;
        }
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }
}
scanner.close();
}
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Output:

```
Select the option:
1. Interest Calculator -SB
2. Interest Calculator -FD
3. Interest Calculator -RD
4. Exit
1
Enter the Average amount in your account:
10000
Enter the account type (Normal/NRI):
Normal
Interest gained: Rs. 400.0
```

Learning Outcomes:

- **Object-Oriented Design:** Learn to create and use classes for real-world entities.
- **Core Programming Skills:** Practice loops, conditionals, and methods for inventory operations.
- **Data Structure Usage:** Use `ArrayList` to manage dynamic data effectively.
- **User-Friendly Systems:** Design intuitive interfaces and handle errors smoothly.



Experiment 5

Student Name: Swastik

UID: 22BCS11755

Branch: BE-CSE

Section/Group: 902_DL_B

Semester: 6th

Date of Performance: 24/02/2025

Subject Name: PBLJ Lab

Subject Code: 22CSH-359

1. Aim:

Implement and manage data using Java Collections and Exception Handling.

2. Problem Statements:

- **Easy Level:** Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).
- **Medium Level:** Create a Java program to serialize and deserialize a Student object. The program should:
 - Serialize a Student object (containing ID, Name, and GPA) and save it to a file.
 - Deserialize the object from the file and display the student details.
 - Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.
- **Hard Level:** Create a menu-based Java application with the following options:
 1. Add an Employee
 2. Display All
 3. Exit

If option 1 is selected, the application should gather details of the employee like employee name, employee ID, designation, and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected, the application should exit.



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

3. Implementation/Code:

Problem 1.1: Sum of Integers Using Autoboxing and Unboxing

```
import java.util.*;

public class SumCalculator {
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Autounboxing happens here
        }
        return sum;
    }

    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        System.out.println("Sum: " + calculateSum(numbers));
    }
}
```

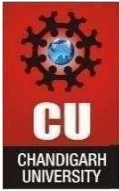
Problem 1.2: Serialization and Deserialization of a Student Object

```
import java.io.*;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name;
    double gpa;

    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", GPA: " + gpa;
    }
}
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}

public class StudentSerialization {
    public static void main(String[] args) {
        Student student = new Student(101, "Alice", 3.9);
        String filename = "student.ser";

        // Serialization
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filename))) {
            out.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (IOException e) {
            System.err.println("IOException: " + e.getMessage());
        }

        // Deserialization
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))) {
            Student deserializedStudent = (Student) in.readObject();
            System.out.println("Deserialized Student: " + deserializedStudent);
        } catch (FileNotFoundException e) {
            System.err.println("FileNotFoundException: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("ClassNotFoundException: " + e.getMessage());
        }
    }
}
```

Problem 1.3: Employee Management System (Menu-Based Application)

```
import java.io.*;
import java.util.*;

class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name, designation;
    double salary;
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public Employee(int id, String name, String designation, double salary) {  
    this.id = id;  
    this.name = name;  
    this.designation = designation;  
    this.salary = salary;  
}
```

@Override

```
public String toString() {  
    return "ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " +  
salary;  
}  
}
```

```
public class EmployeeManagement {  
    private static final String FILE_NAME = "employees.ser";
```

```
    public static void addEmployee(Employee emp) {  
        List<Employee> employees = loadEmployees();  
        employees.add(emp);  
        saveEmployees(employees);  
    }
```

```
    public static List<Employee> loadEmployees() {  
        try (ObjectInputStream in = new ObjectInputStream(new  
FileInputStream(FILE_NAME))) {  
            return (List<Employee>) in.readObject();  
        } catch (IOException | ClassNotFoundException e) {  
            return new ArrayList<>();  
        }  
    }
```

```
    public static void saveEmployees(List<Employee> employees) {  
        try (ObjectOutputStream out = new ObjectOutputStream(new  
FileOutputStream(FILE_NAME))) {  
            out.writeObject(employees);  
        } catch (IOException e) {  
            System.err.println("Error saving employees: " + e.getMessage());  
        }  
    }
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public static void displayEmployees() {
    List<Employee> employees = loadEmployees();
    if (employees.isEmpty()) {
        System.out.println("No employees found.");
    } else {
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("\n1. Add Employee\n2. Display All\n3. Exit\nEnter choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                System.out.print("Enter ID: ");
                int id = scanner.nextInt();
                scanner.nextLine();
                System.out.print("Enter Name: ");
                String name = scanner.nextLine();
                System.out.print("Enter Designation: ");
                String designation = scanner.nextLine();
                System.out.print("Enter Salary: ");
                double salary = scanner.nextDouble();
                addEmployee(new Employee(id, name, designation, salary));
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                scanner.close();
                System.exit(0);
        }
    }
}
```

4. Output:

```
PS D:\today java> cd "d:\today java\" ; if ($?) {  
sum : 150  
PS D:\today java> █
```

(Fig. 1 - Sum of Integers Output)

```
PS D:\today java> cd "d:\today java\" ; if ($?) { javac StudentSerialization  
Student object serialized to student.ser  
Deserialized Student: Student{id=15377, name='Jobajeet Singh', gpa=7.5}  
PS D:\today java> █
```

(Fig. 2 - Student Serialization & Deserialization Output)

```
PS D:\today java> cd "d:\today java\" ; if ($?) { javac Main.java } ; if ($?) { java Main }  
Menu:  
1. Add an Employee  
2. Display All  
3. Exit  
Enter your choice: 1  
Enter employee name: jobanjeet singh  
Enter employee id: 15377  
Enter employee designation: Amritsar  
Enter employee salary: 200000  
Menu:  
1. Add an Employee  
2. Display All  
3. Exit  
Enter your choice: 2  
Employee{name='jobanjeet Singh', id=15377, designation='Amritsar', salary=200000.0}  
Error reading from file: invalid type code: AC  
Menu:
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

(Fig. 3 - Employee Management System Output)

5. Learning Outcome:

1. Understand autoboxing and unboxing in Java.
2. Learn how to parse and work with wrapper classes.
3. Gain hands-on experience with serialization and deserialization of objects.
4. Implement exception handling for file operations in Java.
5. Develop a menu-based Java application with file handling.

1.



Experiment 5

Student Name: Swastik

UID: 22BCS11755

Branch: BE-CSE

Section/Group: 902_DL_B

Semester: 6th

Date of Performance: 24/02/2025

Subject Name: PBLJ Lab

Subject Code: 22CSH-359

1. Aim:

Implement and manage data using Java Collections and Exception Handling.

2. Problem Statements:

- **Easy Level:** Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).
- **Medium Level:** Create a Java program to serialize and deserialize a Student object. The program should:
 - Serialize a Student object (containing ID, Name, and GPA) and save it to a file.
 - Deserialize the object from the file and display the student details.
 - Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.
- **Hard Level:** Create a menu-based Java application with the following options:
 1. Add an Employee
 2. Display All
 3. Exit

If option 1 is selected, the application should gather details of the employee like employee name, employee ID, designation, and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected, the application should exit.



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

3. Implementation/Code:

Problem 1.1: Sum of Integers Using Autoboxing and Unboxing

```
import java.util.*;

public class SumCalculator {
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Autounboxing happens here
        }
        return sum;
    }

    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        System.out.println("Sum: " + calculateSum(numbers));
    }
}
```

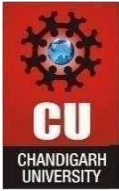
Problem 1.2: Serialization and Deserialization of a Student Object

```
import java.io.*;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name;
    double gpa;

    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", GPA: " + gpa;
    }
}
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}

public class StudentSerialization {
    public static void main(String[] args) {
        Student student = new Student(101, "Alice", 3.9);
        String filename = "student.ser";

        // Serialization
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filename))) {
            out.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (IOException e) {
            System.err.println("IOException: " + e.getMessage());
        }

        // Deserialization
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))) {
            Student deserializedStudent = (Student) in.readObject();
            System.out.println("Deserialized Student: " + deserializedStudent);
        } catch (FileNotFoundException e) {
            System.err.println("FileNotFoundException: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("ClassNotFoundException: " + e.getMessage());
        }
    }
}
```

Problem 1.3: Employee Management System (Menu-Based Application)

```
import java.io.*;
import java.util.*;

class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name, designation;
    double salary;
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public Employee(int id, String name, String designation, double salary) {  
    this.id = id;  
    this.name = name;  
    this.designation = designation;  
    this.salary = salary;  
}
```

@Override

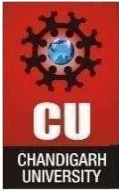
```
public String toString() {  
    return "ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " +  
salary;  
}  
}
```

```
public class EmployeeManagement {  
    private static final String FILE_NAME = "employees.ser";
```

```
    public static void addEmployee(Employee emp) {  
        List<Employee> employees = loadEmployees();  
        employees.add(emp);  
        saveEmployees(employees);  
    }
```

```
    public static List<Employee> loadEmployees() {  
        try (ObjectInputStream in = new ObjectInputStream(new  
FileInputStream(FILE_NAME))) {  
            return (List<Employee>) in.readObject();  
        } catch (IOException | ClassNotFoundException e) {  
            return new ArrayList<>();  
        }  
    }
```

```
    public static void saveEmployees(List<Employee> employees) {  
        try (ObjectOutputStream out = new ObjectOutputStream(new  
FileOutputStream(FILE_NAME))) {  
            out.writeObject(employees);  
        } catch (IOException e) {  
            System.err.println("Error saving employees: " + e.getMessage());  
        }  
    }
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public static void displayEmployees() {
    List<Employee> employees = loadEmployees();
    if (employees.isEmpty()) {
        System.out.println("No employees found.");
    } else {
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("\n1. Add Employee\n2. Display All\n3. Exit\nEnter choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                System.out.print("Enter ID: ");
                int id = scanner.nextInt();
                scanner.nextLine();
                System.out.print("Enter Name: ");
                String name = scanner.nextLine();
                System.out.print("Enter Designation: ");
                String designation = scanner.nextLine();
                System.out.print("Enter Salary: ");
                double salary = scanner.nextDouble();
                addEmployee(new Employee(id, name, designation, salary));
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                scanner.close();
                System.exit(0);
        }
    }
}
```

4. Output:

```
PS D:\today java> cd "d:\today java\" ; if ($?) {  
sum : 150  
PS D:\today java> █
```

(Fig. 1 - Sum of Integers Output)

```
PS D:\today java> cd "d:\today java\" ; if ($?) { javac StudentSerialization  
Student object serialized to student.ser  
Deserialized Student: Student{id=15377, name='Jobajeet Singh', gpa=7.5}  
PS D:\today java> █
```

(Fig. 2 - Student Serialization & Deserialization Output)

```
PS D:\today java> cd "d:\today java\" ; if ($?) { javac Main.java } ; if ($?) { java Main }  
Menu:  
1. Add an Employee  
2. Display All  
3. Exit  
Enter your choice: 1  
Enter employee name: jobanjeet singh  
Enter employee id: 15377  
Enter employee designation: Amritsar  
Enter employee salary: 200000  
Menu:  
1. Add an Employee  
2. Display All  
3. Exit  
Enter your choice: 2  
Employee{name='jobanjeet Singh', id=15377, designation='Amritsar', salary=200000.0}  
Error reading from file: invalid type code: AC  
Menu:
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

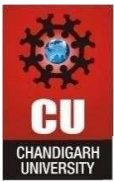
Discover. Learn. Empower.

(Fig. 3 - Employee Management System Output)

5. Learning Outcome:

1. Understand autoboxing and unboxing in Java.
2. Learn how to parse and work with wrapper classes.
3. Gain hands-on experience with serialization and deserialization of objects.
4. Implement exception handling for file operations in Java.
5. Develop a menu-based Java application with file handling.

1.



Experiment 2.1

Student Name:Swastik
Branch: CSE
Semester: 6th
Subject: Java

UID: 22BCS11755
Section: 902_DL_B
DOP:13-01-2025
Subject Code: 22CSH-359

1. Aim: Implement and manage data using Java Collections and Threads.

2. Problem Statements:

- **Easy Level:** Implement an ArrayList to store employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.
- **Medium Level:** Create a program to collect and store all the cards to assist users in finding all the cards in a given symbol using the Collection interface.
- **Hard Level:** Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

3. Implementation/Code:

Problem 1.1: Employee Management System

```
import java.util.*;
class Employee {
    int id;
    String name;
    double salary;

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    @Override
    public String toString() {
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

public class EmployeeManagement {
    private static List<Employee> employees = new ArrayList<>();

    public static void addEmployee(int id, String name, double salary) {
        employees.add(new Employee(id, name, salary));
    }

    public static void updateEmployee(int id, String newName, double newSalary) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                emp.name = newName;
                emp.salary = newSalary;
                return;
            }
        }
    }

    public static void removeEmployee(int id) {
        employees.removeIf(emp -> emp.id == id);
    }

    public static Employee searchEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.id == id) return emp;
        }
        return null;
    }
}
```

Problem 1.2: Card Collection Management

```
import java.util.*;

class CardCollection {
    private List<Card> cards = new ArrayList<>();

    class Card {
        String symbol;
        String cardName;

        Card(String symbol, String cardName) {
            this.symbol = symbol;
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        this.cardName = cardName;
    }

    @Override
    public String toString() {
        return cardName;
    }
}

public void addCard(String symbol, String cardName) {
    cards.add(new Card(symbol, cardName));
}

public List<String> getCardsBySymbol(String symbol) {
    List<String> result = new ArrayList<>();
    for (Card card : cards) {
        if (card.symbol.equals(symbol)) {
            result.add(card.cardName);
        }
    }
    return result;
}

public static void main(String[] args) {
    CardCollection collection = new CardCollection();

    collection.addCard("Hearts", "Ace of Hearts");
    collection.addCard("Hearts", "2 of Hearts");
    collection.addCard("Diamonds", "Ace of Diamonds");

    System.out.println("Hearts: " + collection.getCardsBySymbol("Hearts"));
    System.out.println("Diamonds: " + collection.getCardsBySymbol("Diamonds"));
}
```

Problem 1.3: Ticket Booking System with Synchronized Threads

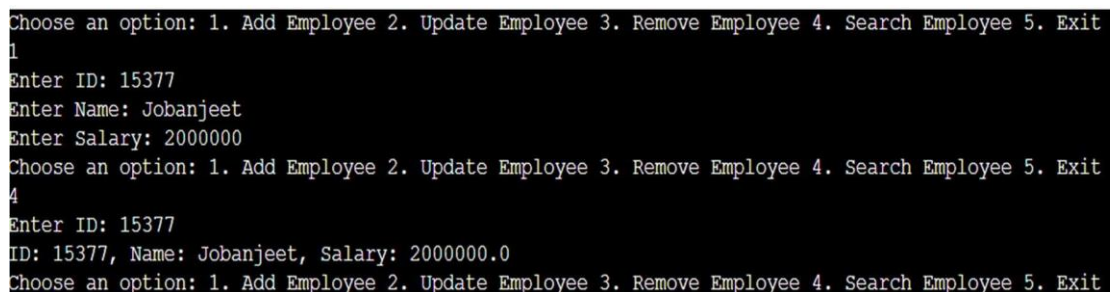
```
class TicketBookingSystem {
    private static int availableSeats = 10;
    private static Lock lock = new ReentrantLock();

    public void bookSeat(String customer) {
        lock.lock();
        try {
```

```
        if (availableSeats > 0) {  
            System.out.println(customer + " booked a seat. Remaining seats: " +  
(--availableSeats));  
        } else {  
            System.out.println("No seats available for " + customer);  
        }  
    } finally {  
        lock.unlock();  
    }  
}  
}
```

```
public class TicketBooking {  
    public static void main(String[] args) {  
        TicketBookingSystem system = new TicketBookingSystem();  
        Thread vip = new Thread(() -> system.bookSeat("VIP Customer"));  
        Thread normal = new Thread(() -> system.bookSeat("Normal  
Customer"));  
        vip.setPriority(Thread.MAX_PRIORITY);  
        normal.setPriority(Thread.MIN_PRIORITY);  
        vip.start();  
        normal.start();  
    }  
}
```

4. Output:



```
Choose an option: 1. Add Employee 2. Update Employee 3. Remove Employee 4. Search Employee 5. Exit  
1  
Enter ID: 15377  
Enter Name: Jobanjeet  
Enter Salary: 2000000  
Choose an option: 1. Add Employee 2. Update Employee 3. Remove Employee 4. Search Employee 5. Exit  
4  
Enter ID: 15377  
ID: 15377, Name: Jobanjeet, Salary: 2000000.0  
Choose an option: 1. Add Employee 2. Update Employee 3. Remove Employee 4. Search Employee 5. Exit
```

(Fig. 1 - Employee Management Output)

```
Cards of Heart: [A, 10]
Cards of Spade: [K]
Cards of Club: []

...Program finished with exit code 0
Press ENTER to exit console.
```

(Fig. 2 - Card Collection Output)

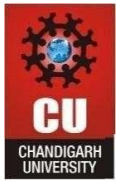
```
VIP Customer booked a seat. Remaining seats: 9
Normal Customer booked a seat. Remaining seats: 8

...Program finished with exit code 0
Press ENTER to exit console. █
```

(Fig. 3 - Ticket Booking System Output)

5. Learning Outcome:

1. Learn to manage collections dynamically using ArrayList and HashMap.
2. Understand thread synchronization and priority handling in Java.
3. Develop real-world applications with efficient data management techniques.



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 7

Student Name:Swastik

UID: 22BCS11755

Branch: CSE

Section: 902_DL_B

Semester: 6th

DOP: 19/3/25

Subject: Java

Subject Code: 22CSH-359

Aim: Create Java applications with JDBC for database connectivity, CRUD operations, and MVC architecture.

Objective: To Create Java applications with JDBC for database connectivity, CRUD operations, and MVC architecture.

Easy Level:

Create a Java program to connect to a MySQL database and fetch data from a single table. The program should:

Use DriverManager and Connection objects.

Retrieve and display all records from a table named Employee with columns EmpID, Name, and Salary.

Code:

```
import java.sql.*; import
```

```
java.util.Scanner;
```

```
public class EmployeeDatabase {
```

```
    private static final String DB_URL = "jdbc:mysql://localhost:3808/test";
```

```
    private static final String USERNAME = "root";    private static final
```

```
    String PASSWORD = "*****";
```

```
    public static void main(String[] args) {
```

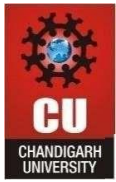
```
        Scanner scanner = new Scanner(System.in);
```

```
        while (true) {
```

```
            System.out.println("\n=== Employee Management System ===");
```

```
            System.out.println("1) View Employee List");
```

```
            System.out.println("2) Exit");
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
System.out.print("Select an option: ");

int option = scanner.nextInt();

if (option == 1) {
fetchEmployees();
} else if (option == 2) {
    System.out.println("Goodbye!");
break;
} else {
    System.out.println("Invalid choice! Please try again.");
}
}

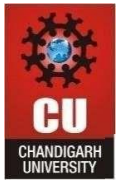
scanner.close();
}

private static void fetchEmployees() {
    String query = "SELECT EmpID, Name, Salary FROM Employee";

    try (Connection conn = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query)) {

        System.out.println("\nEmployee Details:");
        System.out.println("ID | Name | Salary");
        System.out.println("-----");

        while (rs.next()) {
            System.out.printf("%d | %s | %.2f%n", rs.getInt("EmpID"), rs.getString("Name"),
rs.getDouble("Salary"));
        }
    }
}
```



```
        } catch (SQLException ex) {  
            System.err.println("Database connection error: " + ex.getMessage());  
        }  
    }  
}
```

Medium Level:

Build a program to perform CRUD operations (Create, Read, Update, Delete) on a database table Product with columns:

ProductID, ProductName, Price, and Quantity.

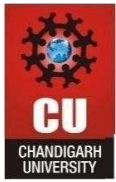
The program should include:

Menu-driven options for each operation.

Transaction handling to ensure data integrity.

Code:

```
import java.sql.*; import  
java.util.Scanner;  
  
public class ProductManager {  
    private static final String DB_URL = "jdbc:mysql://localhost:3808/test";  
    private static final String USER = "root";  
    private static final String PASSWORD = "*****";  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        boolean running = true;  
  
        while (running) {  
            System.out.println("\n===== Product Management =====");  
            System.out.println("1) Add Product");  
            System.out.println("2) View Products");  
            System.out.println("3) Update Product");  
        }  
    }  
}
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        System.out.println("4) Delete Product");

        System.out.println("5) Exit");

        System.out.print("Choose an option: ");

        int choice = scanner.nextInt();

scanner.nextLine(); // Clear newline buffer

        switch (choice) {

            case 1 -> addProduct(scanner);

        case 2 -> viewProducts();           case 3 -
        > updateProduct(scanner);           case 4
        -> deleteProduct(scanner);           case 5

        -> {

                System.out.println("Exiting application...");

running = false;

                }

            default -> System.out.println("Invalid option! Try again.");

                }

        }

        scanner.close();

    }

    private static void addProduct(Scanner scanner) {

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

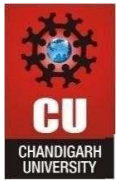
        String name = scanner.nextLine();

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

        double price = scanner.nextDouble();

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

        quantity = scanner.nextInt();
```

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
String sql = "INSERT INTO Product (ProductName, Price, Quantity) VALUES (?, ?, ?)"; try (Connection
conn = DriverManager.getConnection(DB_URL, USER, PASSWORD); PreparedStatement stmt =

conn.prepareStatement(sql)) {

    stmt.setString(1, name);

    stmt.setDouble(2, price);    stmt.setInt(3,
quantity);

    int rowsInserted = stmt.executeUpdate();
if (rowsInserted > 0) {

    System.out.println("Product added successfully!");

    } else {

    System.out.println("Failed to add product.");

    }

} catch (SQLException ex) {

    System.err.println("Error adding product: " + ex.getMessage());

    }

}

private static void viewProducts() {

    String sql = "SELECT * FROM Product";

    try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASSWORD);

        Statement stmt = conn.createStatement();

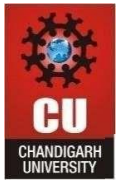
        ResultSet rs = stmt.executeQuery(sql)) {

        System.out.println("\nProduct List:");

        System.out.println("ID | Name | Price | Quantity");

        System.out.println("-----");

        while (rs.next()) {
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        System.out.printf("%d | %s | %.2f | %d%n",
rs.getInt("ProductID"),
rs.getString("ProductName"),
rs.getDouble("Price"),          rs.getInt("Quantity"));
    }
    } catch (SQLException ex) {
        System.err.println("Error retrieving products: " + ex.getMessage());
    }
}

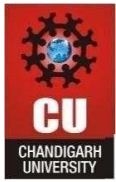
private static void updateProduct(Scanner scanner) {
System.out.print("Enter product ID to update: ");
int id = scanner.nextInt();    scanner.nextLine(); //
Clear buffer

    System.out.print("Enter new product name: ");
    String name = scanner.nextLine();
System.out.print("Enter new price: ");
double price = scanner.nextDouble();
System.out.print("Enter new quantity: ");    int
quantity = scanner.nextInt();

    String sql = "UPDATE Product SET ProductName=?, Price=?, Quantity=? WHERE ProductID=?";

    try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASSWORD);
PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, name);
stmt.setDouble(2, price);    stmt.setInt(3,
quantity);
        stmt.setInt(4, id);
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {
    System.out.println("Product updated successfully!");
} else {
    System.out.println("Product ID not found.");
}
} catch (SQLException ex) {
    System.err.println("Error updating product: " + ex.getMessage());
}
}

private static void deleteProduct(Scanner scanner) {
    System.out.print("Enter product ID to delete: ");    int
    id = scanner.nextInt();

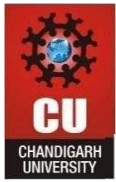
    String sql = "DELETE FROM Product WHERE ProductID=?";

    try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASSWORD);
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, id);

        int rowsDeleted = stmt.executeUpdate();

        if (rowsDeleted > 0) {
            System.out.println("Product deleted successfully!");
        } else {
            System.out.println("Product ID not found.");
        }
    } catch (SQLException ex) {
        System.err.println("Error deleting product: " + ex.getMessage());
    }
}
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}  
  
}
```

Hard Level:

Develop a Java application using JDBC and MVC architecture to manage student data. The application should:

Use a Student class as the model with fields like StudentID, Name, Department, and Marks.

Include a database table to store student data.

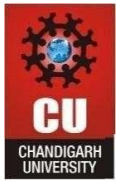
Allow the user to perform CRUD operations through a simple menu-driven view.

Implement database operations in a separate controller class.

Code:

Model

```
public class Student {  
  
    private int id;    private  
    String fullName;    private  
    String dept;    private int  
    score;  
  
    public Student(int id, String fullName, String dept, int score) {  
        this.id = id;  
        this.fullName = fullName;  
        this.dept = dept;    this.score  
        = score;  
    }  
  
    // Getters and Setters    public int  
    getId() { return id; }  
  
    public void setId(int id) { this.id = id; }  
    public String getFullName() { return  
    fullName; }  
}
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public void setFullName(String fullName) { this.fullName = fullName; }
```

```
public String getDept() { return dept; }
```

```
public void setDept(String dept) { this.dept = dept; }
```

```
public int getScore() {  
return score;  
}
```

```
public void setScore(int score) {  
this.score = score;  
}
```

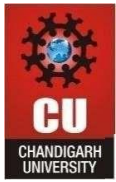
```
@Override  
public String toString() {  
    return "Student ID: " + id + ", Name: " + fullName + ", Department: " + dept + ", Score: " + score;  
}  
}
```

View

```
import java.util.List; import  
java.util.Scanner;
```

```
public class StudentView {  
    private final StudentController studentController = new StudentController();  
    private final Scanner inputScanner = new Scanner(System.in);
```

```
    public void showMenu() {  
        int option;  
do {  
    System.out.println("\n=== Student Management Portal ===");  
    System.out.println("1. Register Student");
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        System.out.println("2. Display All Students");

        System.out.println("3. Modify Student Details");

        System.out.println("4. Remove Student");

        System.out.println("5. Exit");

    System.out.print("Select an option: ");

    option = inputScanner.nextInt();

    inputScanner.nextLine(); // Consume newline


    switch (option) {

        case 1:

            registerStudent();

            break;

        case 2:

            listStudents();

            break;

        case 3:

            modifyStudent();

            break;

        case 4:

            removeStudent();

            break;

        case 5:

            System.out.println("Closing application...");

            break;

        default:

            System.out.println("Invalid option, please try again.");

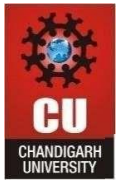
    }

    } while (option != 5);

}


private void registerStudent() {

    System.out.print("Enter Student Name: ");
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
String fullName = inputScanner.nextLine();

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

String department = inputScanner.nextLine();

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

int score = inputScanner.nextInt();


Student newStudent = new Student(0, fullName, department, score);
studentController.addStudent(newStudent);

}


private void listStudents() {

    List<Student> studentList = studentController.getAllStudents();
    if (studentList.isEmpty()) {

        System.out.println("No student records available.");

    } else {

        System.out.println("\n--- Student Records ---");
        for (Student student : studentList) {

            System.out.println(student);

        }

    }

}


private void modifyStudent() {

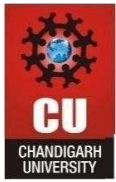
    System.out.print("Enter Student ID to update: ");

    int studentId = inputScanner.nextInt();
    inputScanner.nextLine(); // Consume newline

    System.out.print("Enter Updated Name: ");

    String updatedName = inputScanner.nextLine();

    System.out.print("Enter Updated Department: ");
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
String updatedDepartment = inputScanner.nextLine();

System.out.print("Enter Updated Marks: ");    int
updatedScore = inputScanner.nextInt();

Student updatedStudent = new Student(studentId, updatedName, updatedDepartment, updatedScore);
studentController.updateStudent(updatedStudent);

}

private void removeStudent() {

    System.out.print("Enter Student ID to remove: ");

    int studentId = inputScanner.nextInt();
    studentController.deleteStudent(studentId);

}
}
```

Controller

```
import java.sql.*; import
java.util.ArrayList; import
java.util.List;

public class StudentController {

    private static final String DB_URL = "jdbc:mysql://localhost:3306/javadb";
    private static final String DB_USER = "root";    private static final String
    DB_PASSWORD = "karan.111";

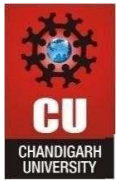
    public void insertStudent(Student student) {

        String sql = "INSERT INTO Students (Name, Department, Marks) VALUES (?, ?, ?)";

        try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

            PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            connection.setAutoCommit(false);
```

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        preparedStatement.setString(1, student.getName());

        preparedStatement.setString(2, student.getDepartment());

        preparedStatement.setInt(3, student.getMarks());

        preparedStatement.executeUpdate();        connection.commit();

        System.out.println("Student successfully registered!");

    } catch (SQLException ex) {
ex.printStackTrace();
    }
}

public List<Student> fetchAllStudents() {
    List<Student> studentList = new ArrayList<>();

    String sql = "SELECT * FROM Students";

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

        Statement statement = connection.createStatement();

        ResultSet resultSet = statement.executeQuery(sql)) {

        while (resultSet.next()) {

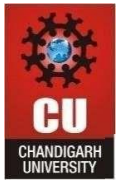
            studentList.add(new Student(resultSet.getInt("StudentID"),

resultSet.getString("Name"),
resultSet.getString("Department"),
resultSet.getInt("Marks")));

        }

    } catch (SQLException ex) {
ex.printStackTrace();
    }

    return studentList;
}
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public void modifyStudent(Student student) {

    String sql = "UPDATE Students SET Name=?, Department=?, Marks=? WHERE StudentID=?";

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
        PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

        connection.setAutoCommit(false);

        preparedStatement.setString(1, student.getName());
        preparedStatement.setString(2, student.getDepartment());
        preparedStatement.setInt(3, student.getMarks());
        preparedStatement.setInt(4, student.getStudentID());

        int affectedRows = preparedStatement.executeUpdate();

        if (affectedRows > 0) {

            connection.commit();

            System.out.println("Student details updated!");

        } else {

            System.out.println("No record found with the given Student ID.");

        }

    } catch (SQLException ex) {

        ex.printStackTrace();

    }

}

public void removeStudent(int studentID) {

    String sql = "DELETE FROM Students WHERE StudentID=?";

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
        PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

        connection.setAutoCommit(false);

        preparedStatement.setInt(1, studentID);

        int affectedRows = preparedStatement.executeUpdate();

        if (affectedRows > 0) {

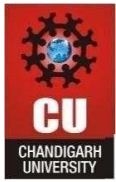
            connection.commit();

            System.out.println("Student record deleted!");

        }

    }

}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        } else {  
            System.out.println("No record found with the given Student ID.");  
        }  
  
    } catch (SQLException ex) {  
ex.printStackTrace();  
    }  
}  
}
```

Main

```
public class StudentApplication {  
    public static void main(String[] args) {  
        StudentView studentView = new StudentView();  
        studentView.showMenu();  
    }  
}
```

Output:

```
C:\Users\123sa\Desktop\Coding\JAVA\Class\exp 7>javac -cp ".;mysql-connector-j-9.2.0.jar" MySQLConnectionCode.java  
C:\Users\123sa\Desktop\Coding\JAVA\Class\exp 7>java -cp ".;mysql-connector-j-9.2.0.jar" MySQLConnectionCode  
Menu:  
1. Display Employees  
2. Exit  
Enter your choice: 1  
  
EmpID | Name | Salary  
-----  
1 | Saket Agarwal | 55000.0  
2 | Ram | 32000.5  
3 | Dam | 41000.75  
4 | Pam | 53000.25  
  
Menu:  
1. Display Employees  
2. Exit  
Enter your choice: 2  
Exiting...
```



1.1 Easy Problem

```
C:\Users\123sa\Desktop\Coding\JAVA\Class\exp 7>javac -cp ".;mysql-connector-j-9.2.0.jar" ProductCRUD.java
C:\Users\123sa\Desktop\Coding\JAVA\Class\exp 7>java -cp ".;mysql-connector-j-9.2.0.jar" ProductCRUD

--- Product Management System ---
1. Add Product
2. View Products
3. Update Product
4. Delete Product
5. Exit
Enter your choice: 2

ProductID | ProductName | Price | Quantity
-----
1 | Laptop | 75000.0 | 10
2 | Mouse | 1500.0 | 50
3 | Keyboard | 2500.0 | 30

--- Product Management System ---
1. Add Product
2. View Products
3. Update Product
4. Delete Product
5. Exit
Enter your choice: 4
Enter Product ID to delete: 3
Product deleted successfully!

--- Product Management System ---
1. Add Product
2. View Products
3. Update Product
4. Delete Product
5. Exit
Enter your choice: 2

ProductID | ProductName | Price | Quantity
-----
1 | Laptop | 75000.0 | 10
2 | Mouse | 1500.0 | 50

--- Product Management System ---
1. Add Product
2. View Products
3. Update Product
4. Delete Product
5. Exit
Enter your choice: 5
Exiting...
```

1.2 Medium Problem

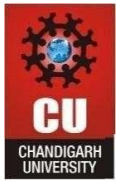
```
C:\Users\123sa\Desktop\Coding\JAVA\Class\exp 7>java -cp ".;mysql-connector-j-9.2.0.jar" StudentMain

--- Student Management System ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2

Student List:
ID: 1, Name: Saket, Dept: Computer Science, Marks: 95
ID: 2, Name: Ram, Dept: Electronics, Marks: 78
ID: 3, Name: Dam, Dept: Mechanical, Marks: 92

--- Student Management System ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 5
Exiting...
```

1.3 Hard Problem



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Learning Outcomes:

1. Integrating Java with Databases – Learn how Java applications interact with databases to store and retrieve data efficiently.
2. Enhancing Data Security – Explore best practices for securing database connections and preventing SQL injection attacks in Java applications.
3. Optimizing Query Performance – Understand how to write efficient SQL queries and use indexing to improve database performance.
4. Building Scalable Applications – Learn how to design a Java-based system that can handle increasing data loads while maintaining performance.



DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment -8

Student Name:Swastik

Branch: BE-CSE

Semester:6th

Subject Name: Project-Based Learning in
Java with Lab

UID:22BCS11755

Section/Group: 902_DL_B

Date of Performance:17/03/2025

Subject Code: 22CSH-359

7.1.1.Aim: To develop a servlet that accepts user credentials from an HTML form and displays a personalized welcome message on successful login.

7.1.2 Objective: Learn form handling with Servlets
Understand HTTP request/response handling
Practice doPost() method

7.1.3 Code:

```
<!DOCTYPE html>
<html>
<head><title>Login</title></head>
<body>
  <form action="LoginServlet" method="post">
    Username: <input type="text" name="username"><br>
    Password: <input type="password" name="password"><br>
    <input type="submit" value="Login">
  </form>
</body>
</html>
```

```
import java.io.*; import
javax.servlet.*;
import javax.servlet.http.*;
```

```
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {        String user =
    request.getParameter("username");
        String pass = request.getParameter("password");
```

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();

if ("admin".equals(user) && "1234".equals(pass)) {
    out.println("<h2>Welcome, " + user + "!</h2>");
} else {
    out.println("<h2>Login Failed. Invalid credentials.</h2>");
}
}
```

```
<web-app>
<servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
</web-app>
```

Output:

- 1) On correct login: Welcome, Sarthak !
- 2) On failure: Login Failed. Invalid credentials.

7.2.1 Aim: To build a servlet integrated with JDBC that displays all employees and enables search by employee ID.

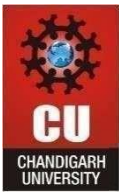
Objective: 1) Use JDBC with Servlet 2)

Fetch and display records

- 3) Implement search functionality

7.2.2 Code:

```
<!DOCTYPE html>
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
<html>
<head><title>Search Employee</title></head>
<body>
  <form action="EmployeeServlet" method="post">
    Enter Employee ID: <input type="text" name="empId">
    <input type="submit" value="Search">
  </form>
</body>
</html>
```

```
import java.io.*; import
javax.servlet.*; import
javax.servlet.http.*; import
java.sql.*;
```

```
public class EmployeeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

```
        String empId = request.getParameter("empId");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
```

```
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
            DriverManager.getConnection("jdbc:mysql://localhost:3306/company", "root", "password");
```

```
            String query = "SELECT * FROM employees WHERE emp_id=?";
            PreparedStatement ps = con.prepareStatement(query);
            ps.setString(1, empId);
            ResultSet rs = ps.executeQuery();
```

```
            if (rs.next()) {
                out.println("<h2>Employee Details</h2>");
                out.println("ID: " + rs.getInt(1) + "<br>");
                out.println("Name: " + rs.getString(2) + "<br>");
                out.println("Department: " + rs.getString(3));
```



```
        } else {  
            out.println("No employee found with ID " + empId);  
        }  
  
        con.close();    }  
catch (Exception e) {  
    out.println("Error: " + e.getMessage());  
}  
}  
}
```

7.2.3 Output:

1) Enter an employee ID → Shows details if found.

2) Not found → "No employee found with ID X"

7.3.1 Aim: To develop a JSP-based student portal that accepts attendance data and saves it to the database using a servlet.

Objective: 1) Combine JSP for UI and Servlets for logic

2) Perform INSERT using JDBC

3) Build a real-world web flow

Code:

```
<%@ page language="java" %>  
<html>  
<head><title>Student Attendance</title></head>  
<body>  
    <h2>Mark Attendance</h2>  
    <form action="AttendanceServlet" method="post">  
        Roll No: <input type="text" name="roll"><br>  
        Name: <input type="text" name="name"><br>  
        Status: <select name="status">  
            <option>Present</option>
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
<option>Absent</option>
</select><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

```
import java.io.*; import
javax.servlet.*; import
javax.servlet.http.*; import
java.sql.*;

public class AttendanceServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {        String roll =
    request.getParameter("roll");
        String name = request.getParameter("name");
        String status = request.getParameter("status");

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
            DriverManager.getConnection("jdbc:mysql://localhost:3306/student_portal", "root",
            "password");

            String query = "INSERT INTO attendance (roll_no, name, status) VALUES (?, ?,
            ?)";
            PreparedStatement ps = con.prepareStatement(query);
            ps.setString(1, roll);        ps.setString(2, name);
            ps.setString(3, status);

            int i = ps.executeUpdate();
            if (i > 0) {
                out.println("<h3>Attendance marked successfully for " + name + "!</h3>");
            }
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

```
        con.close();  
    } catch (Exception e) {  
        out.println("Error: " + e.getMessage());  
    }  
}  
}
```

```
CREATE TABLE attendance (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    roll_no VARCHAR(20),  
    name VARCHAR(100),  
    status VARCHAR(10)  
);
```

OUTPUT

Form submission → "Attendance marked successfully for John!" And the data is stored in the database.



Experiment – 9

Student Name: Swastik
Branch: BE-CSE
Semester: 6
Subject Name: PBLJ

UID: 22BCS11755
Section/Group: 902_DL_B
Date : 24/02/2025

9.1.1.Aim: To demonstrate dependency injection using Spring Framework With Java-based configuration.

9.1.2 Objective:

Define Course and Student classes.

Use Configuration and Bean annotations to inject dependencies.

Load Spring context and print student details.

9.1.3 Code:

// Course.java

```
public class Course {  
    private String courseName;  
    private String duration;  
  
    public Course(String courseName, String duration) {  
        this.courseName = courseName;  
        this.duration = duration;  
    }  
  
    public String getCourseName() { return courseName; }  
    public String getDuration() { return duration; }  
  
    @Override  
    public String toString() {  
        return "Course: " + courseName + ", Duration: " + duration;  
    }  
}
```

// Student.java

```
public class Student {  
    private String name;  
    private Course course;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public Student(String name, Course course) {
    this.name = name;
    this.course = course;
}

public void showDetails() {
    System.out.println("Student: " + name);
    System.out.println(course);
}
} // AppConfig.java
import org.springframework.context.annotation.*;

@Configuration
public class AppConfig {
    @Bean
    public Course course() {
        return new Course("Java", "3 months");
    }

    @Bean
    public Student student() {
        return new Student("Aman", course());
    }
} // MainApp.java
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
        AnnotationConfigApplicationContext(AppConfig.class);
        Student student = context.getBean(Student.class);
        student.showDetails();
    }
}
```

Output:

```
Student: Jobanjeet Singh
Course: Java, Duration: 3 months
```

9.2.1 Aim: To perform CRUD operations on a Student entity using Hibernate ORM with MySQL.

Objective: Define Course and Student classes.

Use Configuration and Bean annotations to inject dependencies.

Load Spring context and print student details.

9.2.2 Code:

```
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">password</property>
    <property
name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <mapping class="Student"/>
  </session-factory>
</hibernate-configuration>
```

```
import javax.persistence.*;
```

Entity

```
public class Student {
    Id
    GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private int age;

    public Student() {}
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Getters, setters, toString
}
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        sessionFactory = new Configuration().configure().buildSessionFactory();
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}

import org.hibernate.*;

public class MainCRUD {
    public static void main(String[] args) {
        Session session = HibernateUtil.getSessionFactory().openSession();

        // Create
        Transaction tx = session.beginTransaction();
        Student s1 = new Student("Aman", 22);
        session.save(s1);
        tx.commit();

        // Read
        Student student = session.get(Student.class, 1);
        System.out.println(student);

        // Update
        tx = session.beginTransaction();
        student.setAge(23);
        session.update(student);
        tx.commit();

        // Delete
        tx = session.beginTransaction();
        session.delete(student);
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
tx.commit();  
  
session.close();  
}  
}
```

9.2.3 Output:

```
Student{id=1, name= Joban'  age=22}  
Updated age to 23  
Deleted student with id 1
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

9.3.1 Aim: To implement a banking system using Spring and Hibernate that ensures transaction consistency during fund transfers.

Objective:

Integrate Spring + Hibernate.

Handle transactions atomically (rollback on failure).

Demonstrate success and failure cases.

Code:

```
import javax.persistence.*;
```

Entity

```
public class Account { @Id  
    private int accountId; private  
    String holderName; private  
    double balance;
```

```
    // Constructors, getters, setters  
}
```

```
import javax.persistence.*;  
import java.util.Date;
```

@Entity

```
public class BankTransaction { @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private int txnId;  
    private int fromAcc;  
    private int toAcc; private  
    double amount;  
    private Date txnDate = new Date();
```

```
    // Constructors, getters, setters  
}  
import org.hibernate.*;  
import org.springframework.transaction.annotation.Transactional;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public class BankService {
    private SessionFactory sessionFactory;

    public BankService(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional
    public void transferMoney(int fromId, int toId, double amount) { Session
        session = sessionFactory.getCurrentSession();

        Account from = session.get(Account.class, fromId); Account
        to = session.get(Account.class, toId);

        if (from.getBalance() < amount) {
            throw new RuntimeException("Insufficient Balance");
        }

        from.setBalance(from.getBalance() - amount);
        to.setBalance(to.getBalance() + amount);

        session.update(from);
        session.update(to);

        BankTransaction txn = new BankTransaction(fromId, toId, amount); session.save(txn);
    }
}

@Configuration
@EnableTransactionManagement
public class AppConfig {
    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource ds = new DriverManagerDataSource();
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/testdb"); ds.setUsername("root");
        ds.setPassword("password");
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return ds;
    }
```

@Bean

```
public LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean lsf = new LocalSessionFactoryBean();
    lsf.setDataSource(dataSource()); lsf.setPackagesToScan("your.package");
    Properties props = new Properties();
    props.put("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");
    props.put("hibernate.hbm2ddl.auto", "update"); lsf.setHibernateProperties(props);
    return lsf;
}
```

@Bean

```
public HibernateTransactionManager transactionManager(SessionFactory sf) { return new
    HibernateTransactionManager(sf);
}
```

@Bean

```
public BankService bankService(SessionFactory sf) { return
    new BankService(sf);
}
}
```

```
public class MainApp {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx = new
        AnnotationConfigApplicationContext(AppConfig.class);
        BankService service = ctx.getBean(BankService.class);

        try {
            service.transferMoney(101, 102, 500);
            System.out.println("Transaction Successful!");
        } catch (Exception e) {
            System.out.println("Transaction Failed: " + e.getMessage());
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        ctx.close();  
    }  
}
```

OUTPUT

```
Transaction Successful!
```

```
OR
```

```
Transaction Failed: Insufficient Balance
```