Experiment 1

Student Name: Sonu Choubey **UID: 22BCS14466**
Branch: CSE **Section: 22BCS_KRG_IOT_3B**
Semester: 6th **DOP: 14/01/25**
Subject: Java **Subject Code:22CSH-359**

1. **Aim:** Create an application to save employee information using arrays.
2. **Objective:** To develop a functional application that effectively utilizes arrays to store, manage, and retrieve employee information, enabling efficient data organization and manipulation within the application.
3. **Code:**

```
import java.util.*;
class Employee {
String name;
int id;
String department;
int salary;
public Employee(String n, int i, String d, int s) {
this.name = n;
this.id = i;
this.department = d;
this.salary = s;
}}
public class EmployeeProject {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number of employees:");
int n = sc.nextInt();
Employee[] employees = new Employee[n];
for (int i = 0; i < n; i++) {
System.out.println("Enter details for employee " + (i + 1) + "-");
System.out.print("Name: ");
String name = sc.next();
System.out.print("ID: ");
int id = sc.nextInt();
System.out.print("Department: ");
```

```
String department = sc.next();
System.out.print("Salary: ");
int salary = sc.nextInt();
sc.nextLine();
employees[i] = new Employee(name, id, department, salary);
}
 System.out.println("Enter the employee ID you want to check information
of:");
 int checkId = sc.nextInt();
boolean found = false;
for (Employee emp : employees) {
if (emp.id == checkId) {
 System.out.println("Employee Details:");
 System.out.println("Name: " + emp.name);
 System.out.println("Department: " + emp.department);
 System.out.println("Salary: " + emp.salary);
found = true;
 break;  } }
if (!found) {
 System.out.println("No employee found with ID: " + checkId);
}
}
}
```

4. Output:

```
PROBLEMS 4   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\22BCS\Downloads\PBLJ\com.student1>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMes
J\com.student1\bin' 'EmployeeProject'
Enter the number of employees:
3
Enter details for employee 1-
Name: Sukhleen
ID: 1102
Department: CSE
Salary: 100000
Enter details for employee 2-
Name: Priya
ID: 1104
Department: ECE
Salary: 90000
Enter details for employee 3-
Name: Rahul
ID: 1134
Department: Arts
Salary: 70000
Enter the employee ID you want to check information of:
1102
Employee Details:
Name: Sukhleen
Department: CSE
Salary: 100000
PS C:\Users\22BCS\Downloads\PBLJ\com.student1>
```

5. **Learning Outcomes:**
   - Learn how to define and use classes and objects, such as the Employee class for organizing and encapsulating employee details
   - Gain insight into using arrays to store multiple objects and efficiently retrieve specific data using loops.
   - Understand how to handle cases where user input does not match expected data, ensuring a robust program.
   - Learn how to traverse data structures (e.g., arrays) to implement search functionality.
   - Develop skills to create user-interactive applications that handle input and display information dynamically

Experiment 2

Student Name: Sonu Choubey **UID: 22BCS14466**
Branch: CSE **Section: 22BCS_KRG_IOT_3B**
Semester: 6th **DOP:14/01/25**
Subject: Java **Subject Code: 22CSH-359**

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

2. **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.
3. **Code:**

```
package exp2;
class Video {
private String title;
private boolean checkedOut;
private double averageRating;
private int ratingCount;
public Video(String title) {
this.title = title;
this.checkedOut = false;
this.averageRating = 0.0;
this.ratingCount = 0;
}
public String getTitle() {
return title;
}
public boolean isCheckedOut() {
return checkedOut;
}
public double getAverageRating() {
return averageRating;
}
public void checkOut() {
if (!checkedOut) {
checkedOut = true;
System.out.println(title + " has been checked out.");
} else {
System.out.println(title + " is already checked out.");
```

```
 }
 }
 public void returnVideo() {
 if (checkedOut) {
 checkedOut = false;
 System.out.println(title + " has been returned.");
 } else {
 System.out.println(title + " was not checked out.");
 }
 }
 public void receiveRating(int rating) {
 if (rating < 1 || rating > 5) {
System.out.println("Invalid rating. Please give a rating between 1 and 5.");
return;
}
averageRating = ((averageRating * ratingCount) + rating) / (++ratingCount);
System.out.println("Rating received for " + title + ": " + rating);
}
@Override
public String toString() {
return "Title: " + title + ", Checked Out: " + checkedOut + ", Average Rating: " +
String.format("%.2f", averageRating);
}
}
class VideoStore {
private Video[] inventory;
private int count;
public VideoStore() {
inventory = new Video[10]; // Initialize the inventory array
count = 0;
}
public void addVideo(String title) {
System.out.println("Attempting to add video: " + title);
if (count < inventory.length) {
inventory[count++] = new Video(title);
System.out.println(title + " has been added to the inventory.");
} else {
System.out.println("Inventory is full. Cannot add more videos.");
}
}
```

```
public void checkOut(String title) {
Video video = findVideo(title);
if (video != null) {
video.checkOut();
} else {
System.out.println("Video not found in inventory.");
}
}
public void returnVideo(String title) {
Video video = findVideo(title);
if (video != null) {
video.returnVideo();
} else {
System.out.println("Video not found in inventory.");
}
}
public void receiveRating(String title, int rating) {
Video video = findVideo(title);
if (video != null) {
video.receiveRating(rating);
} else {
System.out.println("Video not found in inventory.");
}
}
public void listInventory() {
System.out.println("Video Store Inventory:");
for (int i = 0; i < count; i++) {
System.out.println(inventory[i]);
}
}
private Video findVideo(String title) {
for (int i = 0; i < count; i++) {
if (inventory[i].getTitle().equalsIgnoreCase(title)) {
return inventory[i];
}
}
return null;
}
}
public class VideoStoreLauncher {
public static void main(String[] args) {
```

```
VideoStore store = new VideoStore();
System.out.println("Adding videos...");
store.addVideo("The Matrix");
store.addVideo("Godfather II");
store.addVideo("Star Wars Episode IV: A New Hope");
System.out.println("\nReceiving ratings...");
store.receiveRating("The Matrix", 5);
store.receiveRating("The Matrix", 4);
store.receiveRating("Godfather II", 5);
store.receiveRating("Star Wars Episode IV: A New Hope", 3);
store.receiveRating("Star Wars Episode IV: A New Hope", 4);
System.out.println("\nChecking out and returning videos...");
store.checkOut("The Matrix");
store.returnVideo("The Matrix");
store.checkOut("Godfather II");
// List inventory
System.out.println("\nListing inventory...");
store.listInventory();
}
}
```

4. **Output**:

```
PS C:\Users\22BCS\Downloads\PBLJ\com.student1>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsI
J\com.student1\bin' 'exp2.VideoStoreLauncher'
Adding videos...
Attempting to add video: The Matrix
The Matrix has been added to the inventory.
Attempting to add video: Godfather II
Godfather II has been added to the inventory.
Attempting to add video: Star Wars Episode IV: A New Hope
Star Wars Episode IV: A New Hope has been added to the inventory.

Receiving ratings...
Rating received for The Matrix: 5
Rating received for The Matrix: 4
Rating received for Godfather II: 5
Rating received for Star Wars Episode IV: A New Hope: 3
Rating received for Star Wars Episode IV: A New Hope: 4

Checking out and returning videos...
The Matrix has been checked out.
The Matrix has been returned.
Godfather II has been checked out.

Listing inventory...
Video Store Inventory:
Title: The Matrix, Checked Out: false, Average Rating: 4.50
Title: Godfather II, Checked Out: true, Average Rating: 5.00
Title: Star Wars Episode IV: A New Hope, Checked Out: false, Average Rating: 3.50
PS C:\Users\22BCS\Downloads\PBLJ\com.student1>
```

5. Learning Outcomes:

- **OOP Concepts:** Learn encapsulation, abstraction, and modularity in class design.
- **Array Management:** Manage object collections using arrays with basic operations.
- **Method Design:** Implement and validate methods for functionality and error handling.
- **Class Relationships:** Build cohesive programs by linking and managing classes effectively.
- **Debugging Skills:** Develop problem-solving abilities by analyzing and fixing runtime errors.

Experiment 3

Name: Sonu Choubey **UID: 22BCS14466**
Branch: BE-CSE **Section/Group: 22BCS_KRG_IOT_3B**
Semester: 6th **Date of Performance:19/02/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

1. **Aim:** Create an application to calculate interest for FDs, RDs based on certain conditions using inheritance.

2. **Objective:** 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.

3. **Implementation/Code:**

```
package Java;
import java.util.Scanner;
abstract class Account {
double amount;
abstract double calculateInterest();
}
class SBAccount extends
Account {
SBAccount(double amount) {
this.amount = amount; }
double calculateInterest() {
return amount * 0.04;
}
}
class FDAccount extends
Account {     int days, age;
```

```
FDAccount(double amount, int days, int
age) {
this.amount = amount;
this.days = days;
this.age = age;
}
 double calculateInterest()
{
 if (days < 7) {
 System.out.println("Invalid Number of days. Please enter correct values.");
return 0;
}

boolean aboveOneCr = amount >= 1_00_00_000;  //

 if (days >= 7 && days <= 14) rate = aboveOneCr ? (age <
60 ? 0.045 : 0.05) : (age < 60 ? 0.04 : 0.045);
else if (days >= 15 && days <= 29) rate = aboveOneCr ? (age < 60 ? 0.0475 : 0.0525) :
(age < 60 ? 0.0425 : 0.0475);
else if (days >= 30 && days <= 45) rate = aboveOneCr ? (age < 60 ? 0.055 : 0.06) : (age
< 60 ? 0.05 : 0.055);
else if (days >= 46 && days <= 60) rate = aboveOneCr ? (age < 60 ? 0.07 : 0.075) : (age
< 60 ? 0.065 : 0.07);
 else if (days >= 61 && days <= 90) rate = aboveOneCr ? (age < 60 ? 0.075 : 0.08) (age
< 60 ? 0.07 : 0.075);
else if (days >= 91 && days <= 180) rate = aboveOneCr ? (age < 60 ? 0.08 : 0.085) : (age
< 60 ? 0.075 : 0.08);
else rate = aboveOneCr ? (age < 60 ? 0.085 : 0.09) : (age < 60 ? 0.08 : 0.085);
return amount * rate;
 }
}
class RDAccount extends Account {
int months;
RDAccount(double amount, int months) {
this.amount = amount;
this.months = months; }
```

```
double calculateInterest() {
if (months <= 0) {
System.out.println("Invalid Number of months. Please enter correct values.");
return 0;
 }
 boolean aboveOneCr = amount >= 1_00_00_000;
double rate = aboveOneCr ? 0.075 : 0.07;  // 7.5% for >= 1Cr, 7% for < 1Cr
  return amount * rate;
  }
}

public class exp3 {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
while (true) {
  System.out.println("\nSelect 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");
  System.out.print("Enter your choice: ");
  int      choice = sc.nextInt();
if (choice == 4) break;
System.out.print("Enter the present amount in your accunt: ");
double amount = sc.nextDouble();
if (choice == 1) {  // SB Account
System.out.println("Interest gained: Rs. " + new
SBAccount(amount).calculateInterest());
}
else if (choice == 2) {  // FD Account
System.out.print("Enter duration in days: ");
int days = sc.nextInt();
if (days <= 0) {
System.out.println("Invalid Number of days. Please enter correct values.");
continue;
```

```
}
System.out.print("Enter age: ");
int age = sc.nextInt();
System.out.println("Interest gained: Rs. " + new FDAccount(amount, days, age).calculateInterest());
 }
 else if (choice == 3) {  // RD Account
System.out.print("Enter duration in months: ");
int months = sc.nextInt();
 if (months <= 0) {
 System.out.println("Invalid Number of months. Please enter correct values."); continue;
 }
System.out.println("Interest gained: Rs. " + new RDAccount(amount, months).calculateInterest());
} else {
System.out.println("Invalid choice. Please try again.");
 }        }
 sc.close();  }}
```

4. **Output**

```
Select the option:
1. Interest Calculator - SB
2. Interest Calculator - FD
3. Interest Calculator - RD
4. Exit
Enter your choice: 1
Enter the present amount in your accunt: 10000
Interest gained: Rs. 400.0

Select the option:
1. Interest Calculator - SB
2. Interest Calculator - FD
3. Interest Calculator - RD
4. Exit
Enter your choice: 2
Enter the present amount in your accunt: 10000
Enter duration in days: 70
Enter age: 21
Interest gained: Rs. 700.0000000000001

Select the option:
1. Interest Calculator - SB
2. Interest Calculator - FD
3. Interest Calculator - RD
4. Exit
Enter your choice: 3
Enter the present amount in your accunt: 10000
Enter duration in months: 8
Interest gained: Rs. 700.0000000000001

Select the option:
1. Interest Calculator - SB
2. Interest Calculator - FD
3. Interest Calculator - RD
4. Exit
Enter your choice: 4
```

5. Learning Outcome

a) Learned to use abstract classes to define a common interface for subclasses (Account as the base class).
b) Learned how to override abstract methods (calculateInterest) in subclasses (SBAccount, FDAccount, RDAccount) to provide account-specific implementations.Learned how to manage a collection of objects (video inventory) using arrays.
c) Understood how to use conditional statements to determine the appropriate interest rates based on factors like Account, Tenure, Account Holder Age.
d) Gained insights into building applications with real-world relevance, such as an interest calculator for different bank accounts

Experiment 4

Student Name: Sonu Choubey **UID:** 22BCS14466
Branch: BE CSE **Section/Group:**22BCSKRG_IOT3B
Semester: 6th **Date of Performance:**19/02/2025
Subject Name: Programming in java lab **Subject Code:** 22ITH-359

1. Array List Implementation for Employee Details

(Easy) Aim :

To implement an Array List to store employee details such as ID, Name, and Salary and allow users to perform CRUD operations like adding, updating, removing, and searching employees.

Objective :

- To understand the use of the ArrayList class in Java.
- To perform basic operations (add, update, remove, search) on an ArrayList. • To learn how to create a class representing an employee and use it in an ArrayList.

Implementation/Code :

```
package employee;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;
class Employee {
private int id;
private String name;
private double salary;
public Employee(int id, String name, double salary) {
this.id = id; this.name = name; this.salary = salary;
}
```

```
public int getId() {
return id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name= name;
}
public double getSalary() {
return salary;
}
public void setSalary(double salary) {
this.salary = salary;
}
@Override
public String toString() { return "ID: " + id + ", Name: " +
name + ", Salary: " + salary;
}
}
public class EmployeeManagementSystem { private static final
Scanner scanner = new Scanner(System.in);
private static final List<Employee> employees = new ArrayList<>();
public static void main(String[] args) {
while (true) {
System.out.println("\nEmployee Management System");
System.out.println("1. Add Employee");
System.out.println("2. Update Employee");
System.out.println("3. Remove Employee");
System.out.println("4. Search Employee");
```

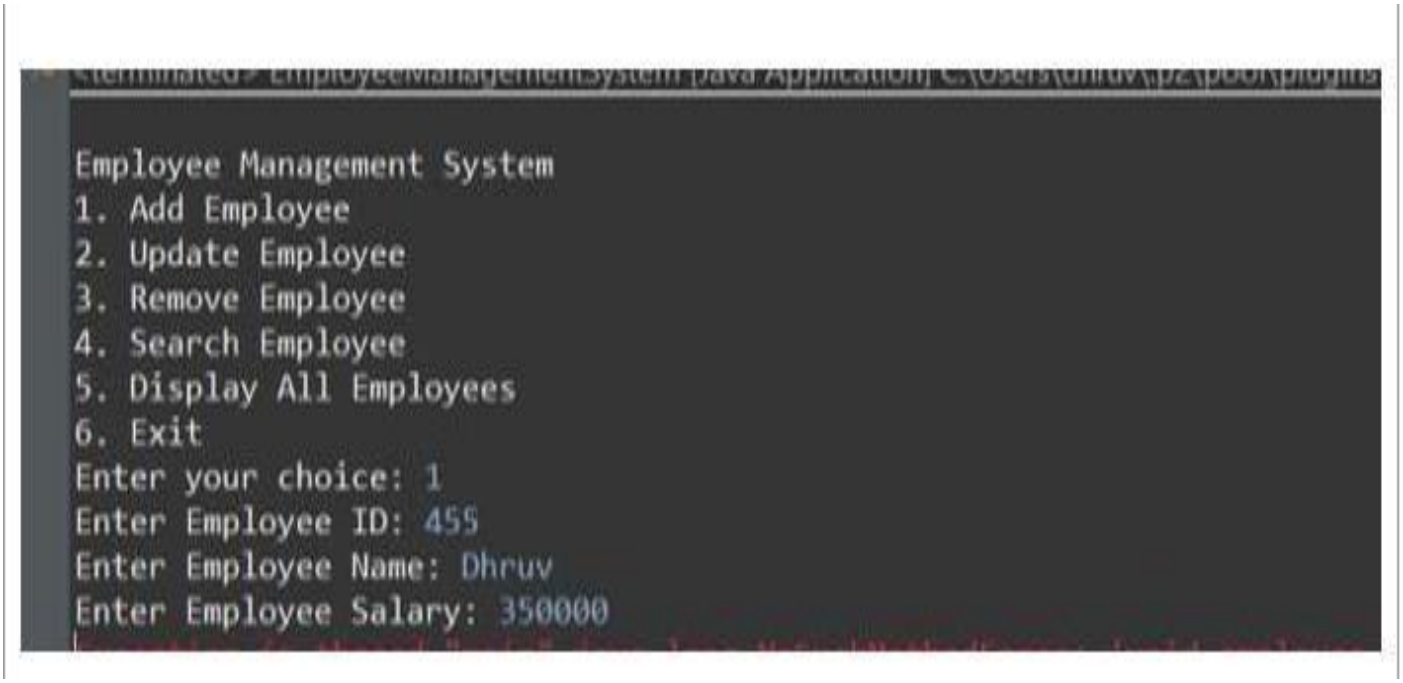
```
System.out.println("5. Display All Employees");
System.out.println("6. Exit"); System.out.print("Enter
your choice: "); int choice = scanner.nextInt();
scanner.nextLine();
switch (choice) { case 1 ->
addEmployee(); case 2 ->
updateEmployee(); case 3 ->
removeEmployee(); case 4 ->
searchEmployee(); case 5 ->
displayEmployees(); case 6 -
> exitProgram();
default -> System.out.println("Invalid choice. Please try again.");
}
}
}
private static void addEmployee() {
System.out.print("Enter Employee ID: ");
int id = scanner.nextInt();
scanner.nextLine();
System.out.print("Enter Employee Name: ");
String name = scanner.nextLine();
System.out.print("Enter Employee Salary: ");
double salary = scanner.nextDouble();
employees.add(new Employee(id, name, salary));
System.out.println("Employee added successfully.");
}
private static void updateEmployee() {
System.out.print("Enter Employee ID to update:
"); int id = scanner.nextInt(); scanner.nextLine();
for (Employee emp : employees) {
if (emp.getId()
== id) {
System.out.print("Enter new Name: ");
emp.setName(scanner.nextLine());
System.out.print("Enter new Salary: ");
emp.setSalary(scanner.nextDouble());
System.out.println("Employee updated successfully.");
 return;
```

```
}
}
System.out.println("Employee not found.");
}
private static void removeEmployee() {
System.out.print("Enter Employee ID to remove: ");
int id = scanner.nextInt();
Iterator<Employee> iterator = employees.iterator();
while (iterator.hasNext()) {
 if (iterator.next().getId() == id) {
iterator.remove();
System.out.println("Employee removed successfully."); return;
}
}
System.out.println("Employee not found.");
}

private static void searchEmployee() {
System.out.print("Enter Employee ID to search:
");
 int id = scanner.nextInt(); for (Employee emp :
employees) {
 if (emp.getId() == id) {
System.out.println(emp);
return;
}
}
System.out.println("Employee not found.");
}
private static void displayEmployees() { if
(employees.isEmpty()) {
System.out.println("No employees to display.");
} else {
employees.forEach(System.out::println);
}
}
```

```
private static void exitProgram() {
System.out.println("Exiting program.");
scanner.close();
System.exit(0);
}
}
```

Output :



2. Card Collection System Using Collection Interface (Medium)

Aim :

To create a program that collects and stores all cards in a collection to assist users in finding all the cards of a given symbol using the Collection interface.

Objective :

- To learn how to use the Collection interface in Java.
- To understand how to manage a collection of objects and filter based on certain attributes. • To practice the use of iteration and filtering in Java collections.

Implementation/Code :

```
package exp4; import
java.util.*;
import java.util.stream.Collectors;
class Card
{ private final String suit;
private final String rank;
 public Card(String suit, String rank)
{
this.suit = suit;
this.rank
= rank;
}
public String getSuit() {
return suit; }
@Override
public String toString() {
 return rank + " of " + suit;
} }
public class CardCollectionSystem {
public static void main(String[] args)
{ List<Card> deck = Arrays.asList(
new Card("Hearts", "Ace"),
new Card("Diamonds", "King"),
new Card("Clubs", "Queen"),
```

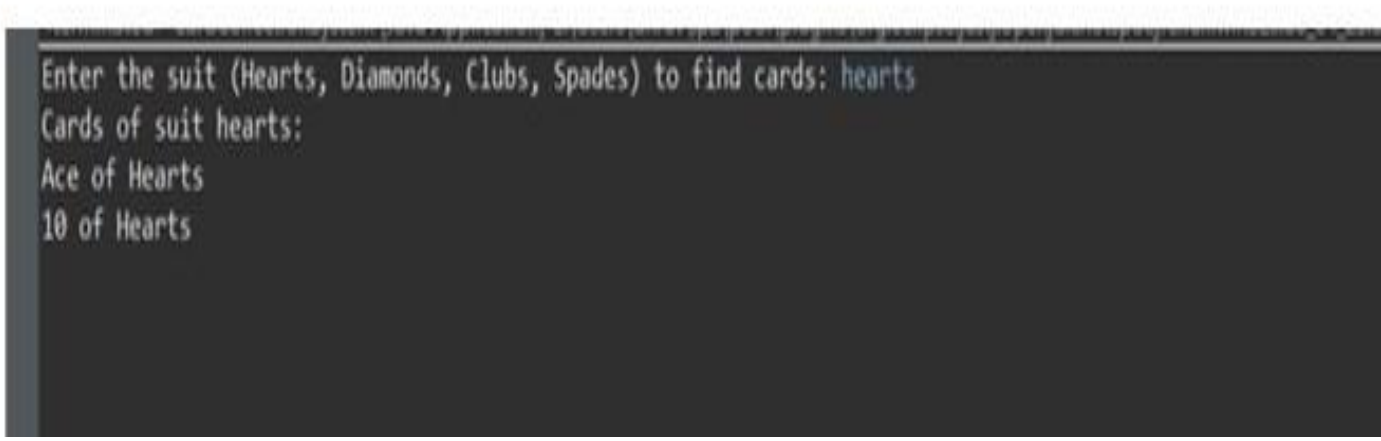
```
new Card("Spades", "Jack"),
new Card("Hearts", "10"),
 new Card("Diamonds", "2"),
new Card("Spades", "Ace")

);
try (Scanner scanner = new Scanner(System.in)) {
System.out.print("Enter the suit (Hearts, Diamonds, Clubs, Spades) to find cards: "); String
suit = scanner.nextLine().trim();

List<Card> filteredCards = deck.stream()
.filter(card -> card.getSuit().equalsIgnoreCase(suit)) .collect(Collectors.toList());

if (filteredCards.isEmpty()) {
System.out.println("No cards found for the suit: " + suit);
} else {
System.out.println("Cards of suit " + suit + ":"); filteredCards.forEach(System.out::println);
}
}
}
}
```

Output :



3. Ticket Booking System with Synchronized Threads(Hard)

Aim :

To develop a ticket booking system where multiple threads ensure no double booking occurs, using synchronized methods, and simulate VIP booking by using thread priorities.

Objective :

- To understand thread synchronization in Java to prevent data inconsistency.
- To implement thread priorities for VIP bookings.
- To learn how to manage concurrent access to shared resources in a multithreaded environment.

Implementation/Code :

```
package exp4;
import java.util.concurrent.locks.ReentrantLock;
class TicketBookingSystem { private
int availableSeats = 10;
private final ReentrantLock lock = new ReentrantLock();
public void bookTicket(String customerName) {
lock.lock();
try {
if (availableSeats > 0) {
System.out.println(customerName + " booked a seat. Remaining seats: " + (availableSeats - 1));
availableSeats--;
} else {
System.out.println(customerName + " attempted to book, but no seats available."); }
} finally { lock.unlock();
} }
public int getAvailableSeats() { return
availableSeats;
}
}
class BookingThread extends Thread {
private final TicketBookingSystem system;
private final String customerName;
```

```
public BookingThread(TicketBookingSystem system, String customerName) { this.system
= system;
this.customerName = customerName;
}
@Override
public void run() {
system.bookTicket(customerName);
} }
public class TicketBookingApp {
public static void main(String[] args) {
TicketBookingSystem system = new TicketBookingSystem();
Thread vipThread = new BookingThread(system, "VIP Customer");
vipThread.setPriority(Thread.MAX_PRIORITY);
Thread regularThread1 = new BookingThread(system, "Regular Customer 1");
Thread regularThread2 = new BookingThread(system, "Regular Customer 2");
vipThread.start(); regularThread1.start();
regularThread2.start();
try { vipThread.join();
regularThread1.join();
regularThread2.join();
} catch (InterruptedException e) {
System.err.println("Thread interrupted: " + e.getMessage());
}  }
}
```

Output :

```
<terminated> TicketBookingApp [Java Application] C:\Users\dhruv\.p2\pool\plugins
VIP Customer booked a seat. Remaining seats: 9
Regular Customer 2 booked a seat. Remaining seats: 8
Regular Customer 1 booked a seat. Remaining seats: 7
```

Learning Outcomes :

- Understanding synchronization in Java and its importance in multithreaded applications.
- Gaining knowledge of thread priorities and how to manage them.
- Learning how to safely share and modify data across multiple threads.

Experiment 5

Student Name: Sonu Choubey **UID:22BCS14466**
Branch: CSE **Section:22BCS_KRG_IOT_3B**
Semester: 6th **DOP:25/02/25**
Subject: PBLJ **Subject Code:22CSH-359**

1. **Aim:** 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()).

2. **Objective:** Demonstrate **autoboxing** and **unboxing** in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

3. **Code:**

```
import java.util.ArrayList;
import java.util.List;
public class AutoboxingExample {
public static void main(String[] args) {
String[] numberStrings = {"10", "20", "30", "40", "50"};
List<Integer> numbers = parseStringArrayToIntegers(numberStrings);
int sum = calculateSum(numbers);
System.out.println("The sum of the numbers is: " + sum);
}
 public static List<Integer> parseStringArrayToIntegers(String[] strings) {
 List<Integer> integerList = new ArrayList<>();
 for (String str : strings) {
 integerList.add(Integer.parseInt(str));
 }
 return integerList;
  }
public static int calculateSum(List<Integer> numbers) {
int sum = 0;
 for (Integer num : numbers) {
 sum += num;
  }
  return sum;
  }
 }
```

4. **Output**:

```
The sum of the numbers is: 150


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

5. **Learning Outcomes:**

- Understand the concept of **autoboxing and unboxing** in Java and how primitive types are automatically converted to their wrapper classes and vice versa.
- Learn how to **convert string values into Integer objects** using Integer.parseInt() and store them in a list.
- Gain experience in **working with ArrayLists** to store and manipulate a collection of numbers dynamically.
- Develop proficiency in **iterating through collections** and performing arithmetic operations like summation.

Experiment 5.2

1. **Aim:** 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.

2. **Objective:** The objective is to serialize and deserialize a Student object, store and retrieve its id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

3. **Code:**

```
import java.io.*;
class Student implements Serializable {
private static final long serialVersionUID = 1L;
private int id;
private String name;
private double gpa;
public Student(int id, String name, double gpa) {
this.id = id;
this.name = name;
this.gpa = gpa;
 }
```

```
    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "', gpa=" + gpa + "}";
    }
}
public class StudentSerialization {
private static final String FILE_NAME = "student.ser";
public static void main(String[] args) {
Student student = new Student(1, "Anwar", 7.8);
serializeStudent(student);
deserializeStudent();
 }
public static void serializeStudent(Student student) {
 try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) { oos.writeObject(student);
System.out.println("Student object serialized successfully.");
 } catch (FileNotFoundException e) {
 System.err.println("File not found: " + e.getMessage());
 } catch (IOException e) {
  System.err.println("IOException occurred: " + e.getMessage());
  }
  }
   public static void deserializeStudent() {
   try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
   Student student = (Student) ois.readObject();
   System.out.println("Deserialized Student: " + student);
  } catch (FileNotFoundException e) {
   System.err.println("File not found: " + e.getMessage());
  } catch (IOException e) {
   System.err.println("IOException occurred: " + e.getMessage());
  } catch (ClassNotFoundException e) {
   System.err.println("Class not found: " + e.getMessage());
   }
   }
   }
```

4. Output

```
Student object serialized successfully.
Deserialized Student: Student{id=1, name='Anwar', gpa=7.8}


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

Experiment 5.3

1. Aim: 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.

2. Objective: The objective is to develop a menu-based Java application that allows users to **add employee details**, **store them in a file**, and **display all stored employee records**, with an option to exit the program.

3. Code:

```
import java.io.*;
import java.util.*;
class Employee implements Serializable {
private static final long serialVersionUID = 1L;
private int id;
private String name;
private String designation;
private double salary;

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 "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ",
Salary: " + salary;
    }
}
public class EmployeeManagementSystem {     private static final
String FILE_NAME = "employees.ser";         private static
List<Employee> employees = new ArrayList<>();
public static void addEmployee() {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter Employee ID: ");
int id = scanner.nextInt();
scanner.nextLine();
System.out.print("Enter Employee Name: ");
String name = scanner.nextLine();
System.out.print("Enter Designation: ");
String designation = scanner.nextLine();
System.out.print("Enter Salary: ");
double salary = scanner.nextDouble();
```

```
Employee employee = new Employee(id, name, designation, salary);
employees.add(employee);
saveEmployees();
System.out.println("Employee added successfully!");
}
public static void displayAllEmployees() {
loadEmployees();
if (employees.isEmpty()) {
System.out.println("No employees found.");
} else {
for (Employee employee : employees) {
System.out.println(employee);                                    }
}
}
private static void saveEmployees() {
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
oos.writeObject(employees);
} catch (IOException e) {
System.err.println("Error saving employees: " + e.getMessage());
}
}

@SuppressWarnings("unchecked")
private static void loadEmployees() {
try (ObjectInputStream ois = new
ObjectInputStream(new
FileInputStream(FILE_NAME))) {
employees = (List<Employee>) ois.readObject();
} catch (FileNotFoundException e) {
employees = new ArrayList<>();
} catch (IOException | ClassNotFoundException e) {
System.err.println("Error loading employees: " + e.getMessage());
}
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
while (true) {
System.out.println("\nEmployee Management System");
System.out.println("1. Add an Employee");
System.out.println("2. Display All Employees");
System.out.println("3. Exit");
System.out.print("Enter your choice: ");
int choice = scanner.nextInt();
scanner.nextLine();

switch (choice) {
```

```
case 1:
      addEmployee();
      break;
case 2:
      displayAllEmployees();
      break;
 case 3:
        System.out.println("Exiting...");
        return;
 default:
        System.out.println("Invalid choice! Please try again.");
     }
     }
     }
    }
```

4. Output:

```
Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 132
Enter Employee Name: Anwar
Enter Designation: HR
Enter Salary: 75000
Employee added successfully!

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 125
Enter Employee Name: Vedant
Enter Designation: Director
Enter Salary: 100000
Employee added successfully!

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2
Employee ID: 132, Name: Anwar, Designation: HR, Salary: 75000.0
Employee ID: 125, Name: Vedant, Designation: Director, Salary: 100000.0
```

5. Learning Outcomes:

- Understand file handling and serialization in Java to store and retrieve objects persistently.
- Learn how to implement a menu-driven console application using loops and conditional statements.

- Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.
- Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
- Develop skills in list manipulation and user input handling using ArrayList and Scanner.

Experiment 6

Name: Sonu Choubey **UID: 22BCS14466**
Branch: BE-CSE **Section/Group: KRG_IOT_3B**
Semester: 6th **Date of Performance:28/02/25**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

EASY:

1. **Aim:** Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

2. **Implementation/Code:**

```
package Java;
import java.util.*;

class Emp {
    String name;
    int age;
    double salary;

    Emp(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    public String toString() {
        return name + " - Age: " + age + ", Salary: " + salary;
    }
}
public class EmployeeSorter {
    public static void main(String[] args) {
        List<Emp> employees = Arrays.asList(
            new Emp("Pragyan", 30, 50000),
            new Emp("Gorisha", 25, 60000),
            new Emp("Manreet", 35, 55000)
        );
        employees.sort(Comparator.comparing((Emp e) -> e.name).thenComparing(e -> e.age)
.thenComparing(e -> e.salary));
        employees.forEach(System.out::println);
    }
}
```

3. **Output:**

```
<terminated> EmployeeSorter [Java Application] C
Gorisha - Age: 25, Salary: 60000.0
Manreet - Age: 35, Salary: 55000.0
Pragyan - Age: 30, Salary: 50000.0
```

MEDIUM:

1. **Aim:** Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

2. **Implementation/Code:**

```
package Java;
import java.util.*;
import java.util.stream.*;

class  Student  {
   String   name;
   double marks;

   Student(String name, double marks) {
      this.name = name;
      this.marks = marks;
   }
}
public class StudentFilter {
   public static void main(String[] args) {
      List<Student> students = Arrays.asList(
         new Student("Reena", 80),
         new Student("Boby", 70),
         new Student("Tina", 85),
         new Student("Dev", 60),
         new Student("Radha", 90)
      );

      List<Student> filteredStudents = students.stream().filter(s -> s.marks > 75).sorted
(Comparator.comparingDouble(s -> -s.marks)).collect(Collectors.toList());

      System.out.println("Students scoring above 75%:");
      filteredStudents.forEach(s -> System.out.println(s.name + " - Marks: " + s.marks));
   }
}
```

3. **Output:**

```
<terminated> StudentFilter [Java Applic
Students scoring above 75%:
Radha - Marks: 90.0
Tina - Marks: 85.0
Reena - Marks: 80.0
```

HARD:

1. **Aim:** Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

2. **Implementation/Code:**

```
package Java;
import java.util.*;
import java.util.stream.*;

class Product {
    String name, category;
    double price;

    public Product(String name, String category, double price) {
        this.name = name;
        this.category = category;
        this.price = price;
    }
    @Override
    public String toString() {
        return name + " ($" + price + ")";
    }
}
public class ProductProcessor {
    public static void main(String[] args) {
        List<Product> products = List.of(
            new Product("Laptop", "Electronics", 1200.0),
            new Product("Phone", "Electronics", 800.0),
            new Product("Tablet", "Electronics", 600.0),
            new Product("Shoes", "Fashion", 100.0),
            new Product("Jacket", "Fashion", 150.0),
            new Product("T-shirt", "Fashion", 50.0)
```

```
        );

        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));
        System.out.println("Products grouped by category:");
        groupedByCategory.forEach((category, productList) -> {
            System.out.println(category + ":");
            productList.forEach(product -> System.out.println("  " + product));
        });

        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category,
                    Collectors.maxBy(Comparator.comparingDouble(p -> p.price))));
        System.out.println("\nMost expensive product in each category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.orElse(null)));

        double averagePrice = products.stream()
            .collect(Collectors.averagingDouble(p -> p.price));
        System.out.println("\nAverage price of all products: " + averagePrice);
    }
}
```

3. **Output:**

```
<terminated> ProductProcessor [Java Application] C:\Users\Lenovo\.
Products grouped by category:
Fashion:
  Shoes ($100.0)
  Jacket ($150.0)
  T-shirt ($50.0)
Electronics:
  Laptop ($1200.0)
  Phone ($800.0)
  Tablet ($600.0)

Most expensive product in each category:
Fashion: Jacket ($150.0)
Electronics: Laptop ($1200.0)

Average price of all products: 483.3333333333333
```

4. Learning Outcome

a) Understanding Lambda Expressions – Learn how to use lambda expressions to simplify function definitions and make code more concise.
b) Sorting with Lambda and Comparator – Utilize Comparator.comparing() and thenComparing() for multi-criteria sorting of objects.
c) Using Java Streams for Data Processing – Gain proficiency in filtering, sorting, mapping, and collecting data using Java's Stream API.
d) Filtering Data with Stream API – Use filter() to extract specific elements from collections based on given conditions.
e) Grouping Data Using Collectors – Understand how to use groupingBy() to categorize and structure data effectively.
f) Finding Max and Min Values in a Dataset – Use maxBy() and minBy() to determine the most expensive or least expensive items in a category.
g) Calculating Aggregates Using Streams – Apply averagingDouble() to compute the average price or marks of a dataset.

Experiment 7

Student Name: Sonu Choubey **UID:22BCS14466**
Branch: CSE **Section:22BCS_KRG_IOT_3B**
Semester: 6th **DOP:25/03/25**
Subject: PBLJ **Subject Code:22CSH-359**

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

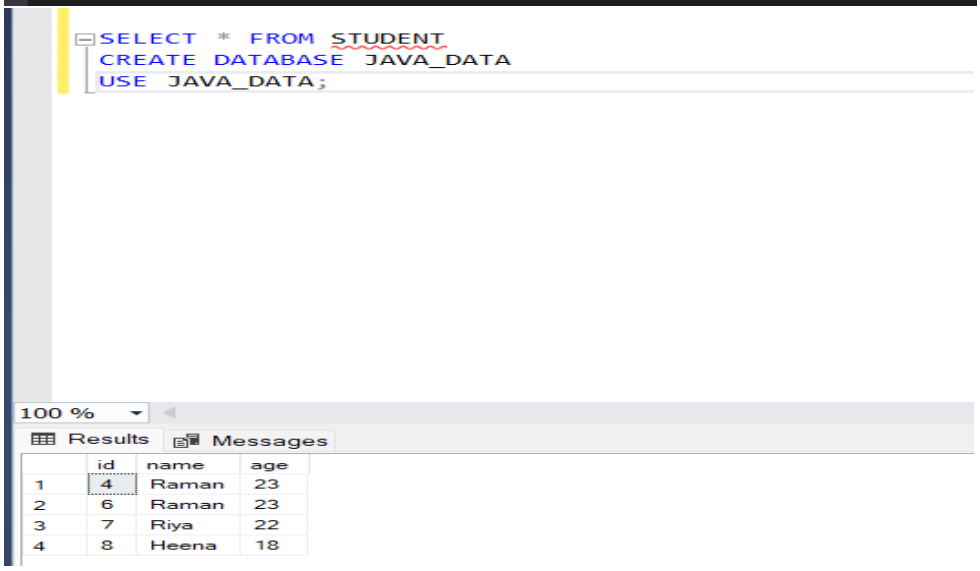
2. **Objective:** To create Java applications with JDBC for database connectivity, CRUD operations, and MVC architecture.

3. **Code:**

```
package com;
import java.sql.*;
public class App2 {
public static void main(String[] args) {
String url=
"jdbc:sqlserver://localhost:1433;databaseName=JAVA_DATA;encrypt=true;trustServerCert
ificate=true;integratedSecurity=true";
// Establish connection
 String username = "hp\\22BCS";
 String password = "1122";
 try{
 Connection conn=DriverManager.getConnection(url, username, password);
 System.out.println("Connected to database");
 //Create the statement
  Statement stmt=conn.createStatement();
  //create table
  String newtable="create table student("
  +"id int IDENTITY(1,1) PRIMARY KEY,"
  +"name varchar(50),"
  +"age int)";
  /stmt.executeUpdate(newtable);
   System.out.println("Table created successfully");
   //insert table
   String insertquery="insert into student(name,age) VALUES
('sukh',21),('Raman',23),('Riya',22),('Heena',18)";
```

```
stmt.executeUpdate(insertquery);
//update data
String updatequery="update student set age=20 where name='Sukh'";
stmt.executeUpdate(updatequery);
//delete data
String deletequery="delete from student where name='sukh'";
stmt.executeUpdate(deletequery);
//read data
String selectQuery="select * from student";
ResultSet rs=stmt.executeQuery(selectQuery);
while(rs.next()){
System.out.println("ID:"+rs.getInt("id")+"name:"+rs.getString("name")+"age:"+rs.get
Int("age"));
}
} catch(SQLException e){
System.out.println(e);
}
}
}
```

4. Output:



5. Learning Outcomes:

- Learn how to establish a connection between a Java application and a relational database using JDBC.
- Gain proficiency in executing SQL queries, retrieving results, and handling database transactions effectively.
- Implement Create, Read, Update, and Delete (CRUD) functionalities using JDBC.
- Apply best practices for handling exceptions, managing connections, and optimizing database interactions.

Experiment -8

Student Name: Sonu Choubey
UID: 22BCS14466
Branch: BE-CSE
Section/Group:KRG_IOT_3B
Semester:6th
Date of Performance:17/03/2025
Subject Name: Project-Based Learning in Java with Lab
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>
<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>
        <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>");
            }

            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: Sonu Choubey **UID:22BCS14466**
Branch: BE-CSE **Section/Group:KRG_IOT_3B**
Semester:6th **Date of Performance:17/03/2025**
Subject Name: Project-Based Learning in Java with Lab **Subject Code: 22CSH-359**

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;     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: Sarthak
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;
    }

    // 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);
```

```
        tx.commit();

        session.close();
    }
}
```

9.2.3 Output:

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

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;
```

```
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");
        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());
}
```

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

```
Transaction Successful!
OR
Transaction Failed: Insufficient Balance
```

```
}
```

OUTPUT