

DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 4

Student Name: Sameer Gautam

Branch: CSE

Semester: 6th

Subject: Java

UID: 22BCS10410

Section: 618/B

DOP: 21/2/2025

Subject Code: 22CSH-359

- 1. Aim:** Write a the given program of three different levels easy , medium , hard using mentioned collections.
- 2. Objective:** Use of Collections in Java. LinkedList, HashMap, HashSet in Java. Multithreading in Java. Thread Synchronization. Thread Priority, Thread lifecycle
- 3. Easy Level:** Write a Java program to implement an that storesemployee details (ID, Name, and Salary). Allow users toadd, update, remove, and search employees.

Code:

```
package College;

import java.util.*;

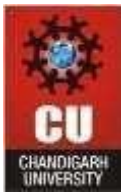
class Employee {
    int id;
    String name;
    double salary;

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

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

public class EmployeeManagement {
    static List<Employee> employees = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.hasNextInt() ? scanner.nextInt() : 0;
            scanner.nextLine(); // Consume newline
            switch (choice) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
            }
        }
    }
}
```



DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        case 5 -> displayEmployees();
        case 6 -> System.exit(0);
        default -> System.out.println("Invalid choice!");
    }
}

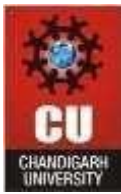
static void addEmployee() {
    System.out.print("Enter ID, Name, Salary: ");
    if (scanner.hasNextInt()) {
        int id = scanner.nextInt();
        String name = scanner.next();
        double salary = scanner.hasNextDouble() ? scanner.nextDouble() : 0;
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added!");
    } else scanner.next();
}

static void updateEmployee() {
    System.out.print("Enter Employee ID to update: ");
    int id = scanner.hasNextInt() ? scanner.nextInt() : -1;
    scanner.nextLine();
    for (Employee emp : employees) {
        if (emp.id == id) {
            System.out.print("Enter new Name and Salary: ");
            emp.name = scanner.next();
            emp.salary = scanner.hasNextDouble() ? scanner.nextDouble() : emp.salary;
            System.out.println("Employee updated!");
            return;
        }
    }
    System.out.println("Employee not found!");
}

static void removeEmployee() {
    System.out.print("Enter Employee ID to remove: ");
    int id = scanner.hasNextInt() ? scanner.nextInt() : -1;
    employees.removeIf(emp -> emp.id == id);
    System.out.println("Employee removed!");
}

static void searchEmployee() {
    System.out.print("Enter Employee ID to search: ");
    int id = scanner.hasNextInt() ? scanner.nextInt() : -1;
    employees.stream().filter(emp -> emp.id == id).forEach(System.out::println);
}

static void displayEmployees() {
    employees.forEach(System.out::println);
}
}
```



DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        if
(items.contains(itemToSearch)) {

    System.out.println("Item found in
the list.");
        } else {

    System.out.println("Item not found
in the list.");
        }
        break;

    case 3:
        System.out.print("Enter
the item to delete : ");
        String itemToDelete =
sc.nextLine();
        if
(items.remove(itemToDelete)) {

    System.out.println("Deleted
successfully");
        } else {

    System.out.println("Item does not
exist.");
        }
        break;

    case 4:
        System.out.println("The
Items in the list are :");
        for (String item : items)
        {

    System.out.println(item);
        }
        break;

    case 5:

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

    default:

    System.out.println("Invalid choice.
Please try again.");
        }
        } while (choice != 5);
    }
}
}
```



DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

PROBLEMS 27 OUTPUT DEBUG CONSOLE TERMINAL PORTS

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit

Choose an option: 1

Enter ID: 11537

Enter Name: Harsh

Enter Salary: 100000

Employee added successfully!

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit

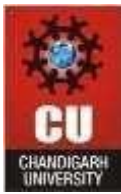
Choose an option: 5

ID: 10410, Name: Sameer, Salary: 20000.0

ID: 11537, Name: Harsh, Salary: 100000.0

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit

Choose an option: █



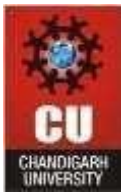
4. Medium Level: Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using the Collection interface.

Code: package College;

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

```
class Card {  
    String name;  
    double balance;  
  
    Card(String name, double balance) {  
        this.name = name;  
        this.balance = balance;  
    }  
  
    @Override  
    public String toString() {  
        return "Card: " + name + ", Balance: " + balance;  
    }  
}
```

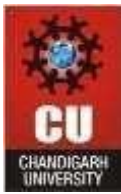
```
public class CardCollectionApp {  
    private static final int CARD_LIMIT = 5;  
  
    public static void main(String[] args) {  
        List<Card> cards = new ArrayList<>();  
        Scanner scanner = new Scanner(System.in);  
  
        while (true) {  
            System.out.println("\n1. Add Card 2. Find Card 3. Display All 4. Exit");  
            System.out.print("Choose an option: ");  
            int choice = scanner.nextInt();  
            scanner.nextLine(); // Consume newline  
  
            switch (choice) {  
                case 1 -> {  
                    if (cards.size() >= CARD_LIMIT) {  
                        System.out.println("Card limit reached! Cannot add more.");  
                    } else {  
                        System.out.print("Enter card name: ");  
                        String cardName = scanner.nextLine();  
                        System.out.print("Enter balance: ");  
                        double balance = scanner.nextDouble();  
                        scanner.nextLine();  
                        cards.add(new Card(cardName, balance));  
                        System.out.println("Card added!");  
                    }  
                }  
                case 2 -> {
```



DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
System.out.print("Enter card name to find: ");
String searchCard = scanner.nextLine();
boolean found = false;
for (Card card : cards) {
    if (card.name.equalsIgnoreCase(searchCard)) {
        System.out.println("Card found: " + card);
        found = true;
        break;
    }
}
if (!found) {
    System.out.println("Card not found.");
}
}
case 3 -> {
    if (cards.isEmpty()) {
        System.out.println("No cards available.");
    } else {
        for (Card card : cards) {
            System.out.println(card);
        }
    }
}
case 4 -> {
    System.out.println("Exiting...");
    System.exit(0);
}
default -> System.out.println("Invalid choice! Try again.");
}
}
}
```



DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
1. Add Card  2. Find Card  3. Display All  4. Exit
```

```
Choose an option: 1
```

```
Enter card name: Sameer
```

```
Enter balance: 2000000
```

```
Card added!
```

```
1. Add Card  2. Find Card  3. Display All  4. Exit
```

```
Choose an option: 1
```

```
Enter card name: Harsh
```

```
Enter balance: 5000000
```

```
Card added!
```

```
1. Add Card  2. Find Card  3. Display All  4. Exit
```

```
Choose an option: 3
```

```
Card: Sameer, Balance: 2000000.0
```

```
Card: Harsh, Balance: 5000000.0
```

```
1. Add Card  2. Find Card  3. Display All  4. Exit
```

```
Choose an option: █
```

5. 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.

Code : package College;

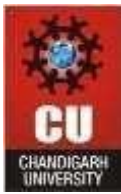
```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;
import java.util.concurrent.locks.ReentrantLock;

class TicketBookingSystem {
    private final boolean[] seats;
    private final ReentrantLock lock = new ReentrantLock();

    public TicketBookingSystem() {
        seats = new boolean[20];
    }

    public void bookSeat(String user, int seatNumber, boolean isVIP) {
        if (seatNumber < 1 || seatNumber > seats.length) {
            System.out.println(user + ": Invalid seat number!");
            return;
        }

        lock.lock();
```



DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        try {
            if (!seats[seatNumber - 1]) {
                seats[seatNumber - 1] = true;
                System.out.println(user + " booked seat " + seatNumber);
            } else {
                System.out.println(user + ": Seat " + seatNumber + " is already booked!");
            }
        } finally {
            lock.unlock();
        }
    }
}

class UserThread extends Thread {
    private final TicketBookingSystem system;
    private final String userName;
    private final int seatNumber;
    private final boolean isVIP;

    public UserThread(TicketBookingSystem system, String userName, int seatNumber, boolean
isVIP) {
        this.system = system;
        this.userName = userName;
        this.seatNumber = seatNumber;
        this.isVIP = isVIP;
    }

    public boolean isVIP() {
        return isVIP;
    }

    @Override
    public void run() {
        system.bookSeat(userName, seatNumber, isVIP);
    }
}

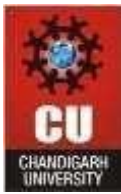
public class TicketBookingApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        TicketBookingSystem system = new TicketBookingSystem();

        System.out.print("Enter the number of users: ");
        int numUsers = scanner.nextInt();
        scanner.nextLine();

        UserThread[] users = new UserThread[numUsers];

        for (int i = 0; i < numUsers; i++) {
            System.out.print("Enter username: ");
            String userName = scanner.nextLine();

            System.out.print("Enter seat number to book: ");
```

DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int seatNumber = scanner.nextInt();

System.out.print("Is the user VIP? (true/false): ");
boolean isVIP = scanner.nextBoolean();
scanner.nextLine();

users[i] = new UserThread(system, userName, seatNumber, isVIP);
}

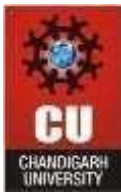
Arrays.sort(users, Comparator.comparing(UserThread::isVIP).reversed());

for (UserThread user : users) {
    user.start();
}

for (UserThread user : users) {
    try {
        user.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

scanner.close();
}
}
```

```
361\bin' 'College.TicketBookingApp'
Enter the number of users: 3
Enter username: Sameer
Enter seat number to book: 1
Is the user VIP? (true/false): True
Enter username: Harsh
Enter seat number to book: 2
Is the user VIP? (true/false): False
Enter username: Kriti
Enter seat number to book: 3
Is the user VIP? (true/false): False
Sameer booked seat 1
Harsh booked seat 2
Kriti booked seat 3
PS C:\Users\samro\Desktop\Codes> █
```



6. Learning Outcome :

- **Understanding Multithreading** – Learn how to create and manage multiple threads for concurrent execution in Java.
- **Synchronization & Locks** – Gain experience in using `ReentrantLock` to handle concurrent access to shared resources safely.
- **Thread Prioritization** – Implement sorting of threads based on priority (VIP users first) using `Comparator`.
- **User Interaction & Input Handling** – Improve skills in handling user input efficiently and validating seat reservations dynamically.