



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 4

Student Name: Kushagar

Branch: CSE

Semester: 6th

Subject: Java

UID:22BCS10429

Section:22BCS_IOT-642A

DOP:25/02/25

Subject Code:22CSH-359

Aim: Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

Objective: Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

Algorithm:

1.) Define Data Structure:

- Create an Employee class/structure with:
- Integer id
- String name
- Double salary

2.) Initialize Data Storage:

- Create an empty list (e.g., ArrayList<Employee>) to hold employee objects.

3.) Main Loop:

- Repeat until the user chooses to exit: 1) Display Menu Options:
- "1. Add Employee"
- "2. Update Employee"
- "3. Remove Employee"
- "4. Search Employee"
- "5. Display All Employees"
- "0. Exit"

2) Input Choice:

- Read the user's menu option (e.g., as an integer).

4.) Process User Choice:

• If choice is 1 (Add Employee):

1. Prompt the user to enter Employee ID.
2. Prompt the user to enter Employee Name.
3. Prompt the user to enter Employee Salary.

4. Create a new Employee object with the provided details.
5. Add the new employee to the list.
6. Display a success message.
- **If choice is 2 (Update Employee):**
 1. Prompt the user to enter the Employee ID to update.
 2. Search for the employee in the list using the given ID.
 3. If the employee exists:
 - Prompt the user to enter the new Name.
 - Prompt the user to enter the new Salary.
 - Update the employee's name and salary.
 - Display a success message.
 4. **Else:**
 - Display a "not found" message.
 - **If choice is 3 (Remove Employee):**
 1. Prompt the user to enter the Employee ID to remove.
 2. Search for the employee in the list using the given ID.
 3. If the employee exists:
 - Remove the employee from the list.
 - Display a success message.
 4. **Else:**
 - Display a "not found" message.
 - **If choice is 4 (Search Employee):**
 1. Prompt the user to enter the Employee ID to search.
 2. Search for the employee in the list using the given ID.
 3. If the employee exists:
 - Display the employee's details.
 4. **Else:**
 - Display a "not found" message.
 - **If choice is 5 (Display All Employees):**
 1. If the employee list is empty:
 - Display a message indicating no employees to show.
 2. **Else:**
 - Iterate over the list and display each employee's details.
 - **If the choice is 0 (Exit):**
 - Terminate the program loop.

5.) End Program

Code:

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

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 classwork3 {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        int choice;
        do {
            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: ");
            choice = scanner.nextInt();

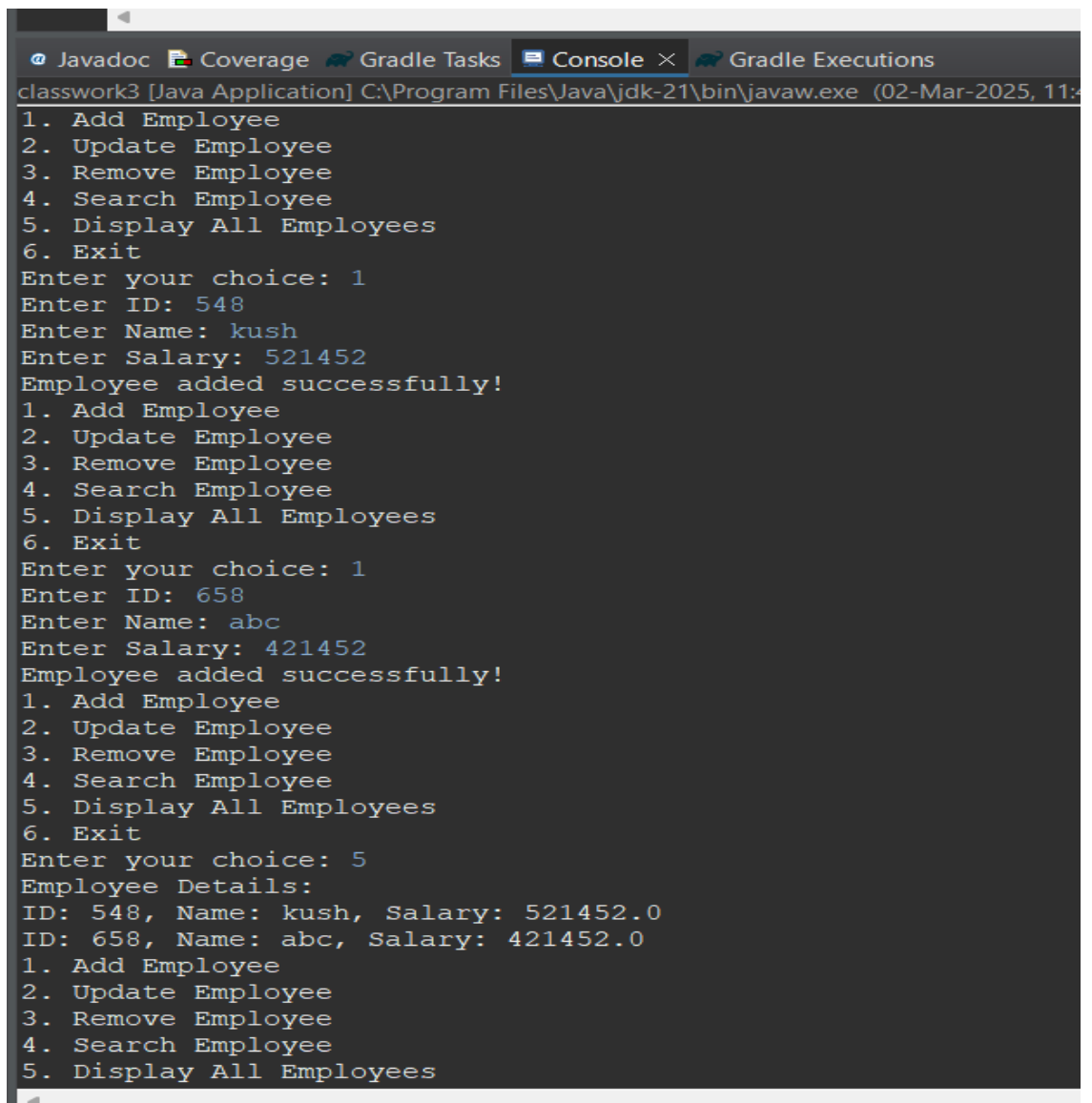
            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = scanner.nextInt();
                    System.out.print("Enter Name: ");
                    String name = scanner.next();
                    System.out.print("Enter Salary: ");
                    double salary = scanner.nextDouble();
                    employees.add(new Employee(id, name, salary));
                    System.out.println("Employee added successfully!");
                    break;
                case 2:
                    System.out.print("Enter Employee ID to Update: ");
                    int updateId = scanner.nextInt();
```

```
boolean found = false;
for (Employee emp : employees) {
    if (emp.id == updateId) {
        System.out.print("Enter New Name: ");
        emp.name = scanner.next();
        System.out.print("Enter New Salary: ");
        emp.salary = scanner.nextDouble();
        System.out.println("Employee updated successfully!");
        found = true;
        break;
    }
}
if (!found) {
    System.out.println("Employee not found!");
}
break;
case 3:
    // Remove Employee
    System.out.print("Enter Employee ID to Remove: ");
    int removeId = scanner.nextInt();
    found = false;
    for (Employee emp : employees) {
        if (emp.id == removeId) {
            employees.remove(emp);
            System.out.println("Employee removed successfully!");
            found = true;
            break;
        }
    }
    if (!found) {
        System.out.println("Employee not found!");
    }
    break;
case 4:
    System.out.print("Enter Employee ID to Search: ");
    int searchId = scanner.nextInt();
    found = false;
    for (Employee emp : employees) {
        if (emp.id == searchId) {
            System.out.println(emp);
            found = true;
            break;
        }
    }
    if (!found) {
        System.out.println("Employee not found!");
    }
    break;
```

```
case 5:
System.out.println("Employee Details:");
for (Employee emp : employees) {
System.out.println(emp);
}
break;
case 6:
System.out.println("Exiting...");
break;
default:
System.out.println("Invalid choice! Please try again.");
}
} while (choice != 6);

scanner.close();
}
}
```

Output:



```
classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (02-Mar-2025, 11:4
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter ID: 548
Enter Name: kush
Enter Salary: 521452
Employee added successfully!
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter ID: 658
Enter Name: abc
Enter Salary: 421452
Employee added successfully!
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 5
Employee Details:
ID: 548, Name: kush, Salary: 521452.0
ID: 658, Name: abc, Salary: 421452.0
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
```

Code 2:

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

Objective: Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

Algorithm:

- **Create a Card Class:**
 - Define attributes: symbol and value.
 - Implement a constructor to initialize these attributes.
 - Override the toString() method to display card details.
- **Create a CardManager Class:**
 - Define a List<Card> to store the cards.
 - Implement methods for adding a card, finding cards by symbol, and displaying all cards.
- **Main Method:**
 - Initialize a Scanner for user input.
 - Display a menu with options to add a card, find cards by symbol, display all cards, and exit.
 - Use a do-while loop to handle user input and call the appropriate methods based on the user's choice.

Code:

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

class Card {
    String symbol;
    String value;

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

    public String toString() {
        return value + " of " + symbol;
    }
}

public class classwork3 {
    public static void main(String[] args) {
        List<Card> cards = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
```

```
int choice;
do {
    System.out.println("1. Add Card");
    System.out.println("2. Find Cards by Symbol");
    System.out.println("3. Display All Cards");
    System.out.println("4. Exit");
    System.out.print("Enter your choice: ");
    choice = scanner.nextInt();

    switch (choice) {
        case 1:
            System.out.print("Enter Symbol (e.g., Hearts, Diamonds): ");
            String symbol = scanner.next();
            System.out.print("Enter Value (e.g., Ace, 2, King): ");
            String value = scanner.next();
            cards.add(new Card(symbol, value));
            System.out.println("Card added successfully!");
            break;
        case 2:
            System.out.print("Enter Symbol to Search: ");
            String searchSymbol = scanner.next();
            System.out.println("Cards with symbol " + searchSymbol + ":");
            for (Card card : cards) {
                if (card.symbol.equalsIgnoreCase(searchSymbol)) {
                    System.out.println(card);
                }
            }
            break;
        case 3:
            System.out.println("All Cards:");
            for (Card card : cards) {
                System.out.println(card);
            }
            break;
        case 4:
            System.out.println("Exiting...");
            break;
        default:
            System.out.println("Invalid choice! Please try again.");
    }
} while (choice != 4);

scanner.close();
}
```

Output:

```
@ Javadoc Coverage Gradle Tasks Console x Gradle Executions
<terminated> classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (0
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (e.g., Hearts, Diamonds): Heart
Enter Value (e.g., Ace, 2, King): 2
Card added successfully!
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (e.g., Hearts, Diamonds): club
Enter Value (e.g., Ace, 2, King): ace
Card added successfully!
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 3
All Cards:
2 of Heart
ace of club
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 4
Exiting...
```


Code 3:

Aim: 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.

Objective: Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

Algorithm:

- **Define Seat Class:**
 - Create a class Seat with attributes seatNumber and isBooked.
 - Implement a constructor to initialize seatNumber and set isBooked to false.
 - Implement methods getSeatNumber(), isBooked(), and bookSeat().
- **Define BookingSystem Class:**
 - Create a class BookingSystem with an array of Seat objects.
 - Implement a constructor to initialize the array with a specified number of seats.
 - Implement a synchronized method bookSeat(int seatNumber):
 - Check if the seat is already booked.
 - If not, mark the seat as booked and return true.
 - If yes, return false.
- **Create BookingThread Class:**
 - Create a class BookingThread that extends Thread.
 - Include attributes for BookingSystem and seatNumber.
 - Implement a constructor to initialize these attributes and set the thread name and priority.
 - Override the run() method to call the bookSeat(int seatNumber) method of BookingSystem.
- **Main Method:**
 - Initialize the BookingSystem with a specified number of seats.
 - Create multiple BookingThread instances for VIP and regular bookings with different priorities.
 - Start the booking threads.

Code: class Seat {

```
private int seatNumber;  
private boolean isBooked;
```

```
public Seat(int seatNumber) {
```

```
this.seatNumber = seatNumber;
this.isBooked = false;
}

public int getSeatNumber() {
    return seatNumber;
}

public boolean isBooked() {
    return isBooked;
}

public void bookSeat() {
    this.isBooked = true;
}
}

class BookingSystem {
    private Seat[] seats;

    public BookingSystem(int numberOfSeats) {
        seats = new Seat[numberOfSeats];
        for (int i = 0; i < numberOfSeats; i++) {
            seats[i] = new Seat(i + 1);
        }
    }

    public synchronized boolean bookSeat(int seatNumber) {
        if (!seats[seatNumber - 1].isBooked()) {
            seats[seatNumber - 1].bookSeat();
            System.out.println("Seat " + seatNumber + " booked successfully by " +
Thread.currentThread().getName());
            return true;
        } else {
            System.out.println("Seat " + seatNumber + " is already booked.");
            return false;
        }
    }
}

class BookingThread extends Thread {
    private BookingSystem bookingSystem;
    private int seatNumber;
```

```
public BookingThread(BookingSystem bookingSystem, int seatNumber, String
name, int priority) {
    super(name);
    this.bookingSystem = bookingSystem;
    this.seatNumber = seatNumber;
    setPriority(priority);
}

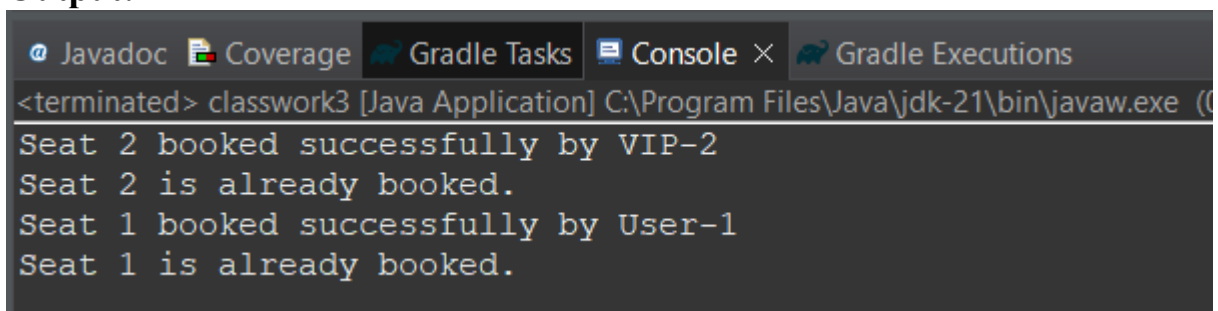
@Override
public void run() {
    bookingSystem.bookSeat(seatNumber);
}
}public class classwork3 {
    public static void main(String[] args) {
        BookingSystem bookingSystem = new BookingSystem(10);

        BookingThread vip1 = new BookingThread(bookingSystem, 1, "VIP-1",
Thread.MAX_PRIORITY);
        BookingThread vip2 = new BookingThread(bookingSystem, 2, "VIP-2",
Thread.MAX_PRIORITY);

        BookingThread user1 = new BookingThread(bookingSystem, 1, "User-1",
Thread.NORM_PRIORITY);
        BookingThread user2 = new BookingThread(bookingSystem, 2, "User-2",
Thread.NORM_PRIORITY);

        vip1.start();
        vip2.start();
        user1.start();
        user2.start();
    }
}
```

Output:



```
<terminated> classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (0
Seat 2 booked successfully by VIP-2
Seat 2 is already booked.
Seat 1 booked successfully by User-1
Seat 1 is already booked.
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Learning Outcomes:

1. **Demonstrate:** Apply key concepts to real-world scenarios to showcase understanding.
2. **Analyze:** Critically evaluate information, identify patterns, and draw meaningful conclusions.
3. **Create:** Develop original work, including presentations, reports, or projects, to exhibit comprehension and skills.