Experiment 2.1

Student Name: Sheetal Kushwaha **UID: 22BCS17181**
Branch: BE CSE **Section/Group: KPIT_901/A**
Semester: 6th **Date of Performance: 21/02/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

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

Easy Level: 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.
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 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.

2. **Objective:** To enhance practical understanding of core Java concepts by developing compact programs that utilize data structures, collections, and multithreading for managing and manipulating data in real-world scenarios.

3. **Implementation/Code + Output:**

Easy Level:

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

class Employee {
   String id;
   String name;
```

```
    double salary;

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

    @Override
    public String toString() {
        return String.format("%-10s %-15s $%,10.2f", id, name, salary);
    }
}

public class Employee_List {
    private static final ArrayList<Employee> employees = new ArrayList<>();
    private static final Scanner scanner = new
Scanner(System.in);

    public static void main(String[] args) {        while (true) {
            System.out.println("\n===== Employee
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 Employees");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();          scanner.nextLine(); // Consume newline

            switch (choice) {              case 1 -> add();
case 2 -> update();              case 3 -> remove();
case 4 -> search();              case 5 -> display();
case 6 -> {                System.out.println("Exiting...
Thank you!");
                return;              }
```

```
            default -> System.out.println("Invalid choice. Please enter a number between
1-6.");
        }
      }
    }

    private static void add() {
      System.out.print("Enter Employee ID: ");
      String id = scanner.nextLine().trim();

      if (id.isEmpty() || find(id) != null) {
        System.out.println("Error: Employee with ID '" + id + "' already exists or ID
cannot be empty.");
        return;
      }
      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 update() {
      System.out.print("Enter Employee ID to update: ");
      String id = scanner.nextLine().trim();

      Employee emp = find(id);       if (emp == null) {
        System.out.println("Error: Employee not found.");          return;
      }
      System.out.print("Enter New Name: ");       emp.name =
scanner.nextLine();       System.out.print("Enter New Salary: ");       emp.salary
= scanner.nextDouble();

      System.out.println("Employee updated successfully.");
```

```
    }

    private static void remove() {
        System.out.print("Enter Employee ID to remove: ");
        String id = scanner.nextLine().trim();

        Employee emp = find(id);        if (emp == null) {
            System.out.println("Error: Employee not found.");           return;
        }

        employees.remove(emp);
        System.out.println("Employee removed successfully.");
    }

    private static void search() {
        System.out.print("Enter Employee ID to search: ");
        String id = scanner.nextLine().trim();

        Employee emp = find(id);        if (emp == null) {
            System.out.println("Error: Employee not found.");           return;
        }
        System.out.println("Employee Found: " + emp);
    }

    private static void display() {        if
(employees.isEmpty()) {
            System.out.println("No employees found.");
            return;
        }
System.out.pri
ntln("\n=====
============
===========
==================");
        System.out.println(String.format("%-10s %-15s
%10s", "ID", "Name", "Salary"));
        System.out.println("============================
```

```
=================");

      for (Employee emp : employees) {
         System.out.println(emp);
      }
      System.out.println("==============================
=================");
   }

   private static Employee find(String id) {       for (Employee emp :
employees) {          if (emp.id.equalsIgnoreCase(id)) {
return emp;
         }       }       return null;
   }
}
```

OUTPUT

```
===== Employee Management System =====
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 1
Enter Employee ID: AC314
Enter Employee Name: mimi
Enter Employee Salary: 54498
Employee added successfully.

===== Employee Management System =====
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 5

================================================
ID          Name                  Salary
================================================
AC314       mimi             $ 54,498.00
================================================

===== Employee Management System =====
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 6
Exiting... Thank you!
```

Medium Level:

```
import java.util.*;

public class Card_Collection {
   private static final Map<Character, String> SUIT_MAP =
Map.of(
         'S', "Spades", 'H', "Hearts", 'D', "Diamonds",
'C', "Clubs"     );
   private static final List<String> CARD_ORDER = List.of("K", "Q", "J", "A", "10", "9",
"8", "7", "6", "5",
"4", "3", "2");

   public static void main(String[] args) {        TreeMap<String, TreeSet<String>> cards
= new TreeMap<>();
      for (String suit : SUIT_MAP.values()) {           cards.put(suit, new
TreeSet<>(Comparator.comparingInt(CARD_ORDER::indexOf)));
      }
      Scanner scanner = new Scanner(System.in);

      while (true) {
         System.out.println("\n===== Card Collection
System =====");
         System.out.println("1. Add a Card");
         System.out.println("2. Search for a Card");
         System.out.println("3. Display All Cards");
         System.out.println("4. Exit");
         System.out.print("Choose an option: ");

         int choice = scanner.nextInt();          scanner.nextLine(); // Consume newline

         switch (choice) {
            case 1 -> addCard(cards, scanner);              case 2 -> searchCard(cards,
scanner);              case 3 -> displayCards(cards);
            case 4 -> {
               System.out.println("Exiting... Thank you!");
               return;
            }
```

```
            default -> System.out.println("Invalid choice. Please enter a number between
1-4.");
        }
      }
    }

    private static void addCard(TreeMap<String,
TreeSet<String>> cards, Scanner scanner) {
      System.out.print("Enter card suit (S for Spades, H for Hearts, D for Diamonds, C
for Clubs): ");        char suitChar =
Character.toUpperCase(scanner.next().charAt(0));

      if (!SUIT_MAP.containsKey(suitChar)) {           System.out.println("Error:
Invalid suit. Choose from S, H, D, C.");           return;
      }
      String suit = SUIT_MAP.get(suitChar);

      System.out.print("Enter card value (K, Q, J, A,
10, 9, ..., 2): ");        String value =
scanner.next().trim().toUpperCase();

      if (!CARD_ORDER.contains(value)) {           System.out.println("Error:
Invalid card value.");           return;
      }

      if (cards.get(suit).contains(value)) {
        System.out.println("Error: Card already exists in the collection.");        } else {
        cards.get(suit).add(value);           System.out.println("Card added
successfully.");
      }
    }

    private static void searchCard(TreeMap<String,
TreeSet<String>> cards, Scanner scanner) {        System.out.print("Enter card suit to
search (S, H, D, C): ");
      char suitChar =
Character.toUpperCase(scanner.next().charAt(0));
```

```
        if (!SUIT_MAP.containsKey(suitChar)) {            System.out.println("Error: Invalid
suit.");            return;
        }
        String suit = SUIT_MAP.get(suitChar);

        System.out.print("Enter card value to search: ");        String value =
scanner.next().trim().toUpperCase();

        if (cards.get(suit).contains(value)) {            System.out.println(value + " of " + suit
+ " is in the collection.");
        } else {
            System.out.println(value + " of " + suit + " is not in the collection.");
        }
    }

    private static void displayCards(TreeMap<String,
TreeSet<String>> cards) {
        System.out.println("\n===== Card Collection
=====");

        for (String suit : SUIT_MAP.values()) {            System.out.print(suit +
": ");            if (cards.get(suit).isEmpty()) {
System.out.println("No cards.");
            } else {
                System.out.println(String.join(", ", cards.get(suit)));
            }
        }
    }
}
```

OUTPUT

```
4. Exit
Choose an option: 1
Enter card suit (S for Spades, H for Hearts, D for Diamonds, C for Clubs): s
Enter card value (K, Q, J, A, 10, 9, ..., 2): k
Card added successfully.

===== Card Collection System =====
1. Add a Card
2. Search for a Card
3. Display All Cards
4. Exit
Choose an option: 1
Enter card suit (S for Spades, H for Hearts, D for Diamonds, C for Clubs): h
Enter card value (K, Q, J, A, 10, 9, ..., 2): 5
Card added successfully.

===== Card Collection System =====
1. Add a Card
2. Search for a Card
3. Display All Cards
4. Exit
Choose an option: 3

===== Card Collection =====
Hearts: 5
Clubs: No cards.
Diamonds: No cards.
Spades: K

===== Card Collection System =====
1. Add a Card
2. Search for a Card
3. Display All Cards
4. Exit
Choose an option: 4
Exiting... Thank you!
```

Hard Level:

```
import java.util.InputMismatchException; import java.util.Scanner;

class TicketBooking extends Thread {     private static int
availableSeats;     private final boolean isVIP;

   public TicketBooking(String name, boolean isVIP) {        super(name);
this.isVIP = isVIP;
   }

   public static void setAvailableSeats(int seats) {
      availableSeats = seats;
   }

   public void run() {
      synchronized (TicketBooking.class) {           if (availableSeats > 0)
{
            System.out.println(getName() + " booked a seat. (" + --availableSeats + "
seats remaining)");
         } else {
            System.out.println(getName() + " could not book a seat. No seats
available.");
         }
      }
   }
}

public class Ticket_Booking_System {     public static void main(String[]
args) {        Scanner scanner = new Scanner(System.in);        int seats = 0,
numPassengers = 0;

      // Get number of available seats
      while (true) {           try {
            System.out.print("\nEnter the number of available seats: ");
            seats = scanner.nextInt();              if (seats < 0) throw new
IllegalArgumentException("Seat count cannot be negative.");
TicketBooking.setAvailableSeats(seats);           break;
```

```
        } catch (InputMismatchException e) {            System.out.println("Invalid
input! Please enter a valid number.");
            scanner.nextLine(); // Clear buffer
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
    // Get number of passengers
    while (true) {          try {
        System.out.print("Enter the number of passengers: ");
        numPassengers = scanner.nextInt();              if (numPassengers <= 0) throw
new IllegalArgumentException("Number of passengers must be at least 1.");
        scanner.nextLine(); // Consume newline
        break;
        } catch (InputMismatchException e) {            System.out.println("Invalid
input! Please enter a valid number.");
        scanner.nextLine(); // Clear buffer
        } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
        }
    }
    TicketBooking[] passengers = new
TicketBooking[numPassengers];

    System.out.println("\n===== Passenger Details
=====");
    for (int i = 0; i < numPassengers; i++) {          String name;
boolean isVIP;

        while (true) {              try {
            System.out.print("Enter passenger name: ");
                    name = scanner.nextLine().trim();
                if (name.isEmpty()) throw new IllegalArgumentException("Name
cannot be empty.");                 break;
            } catch (IllegalArgumentException e) {
System.out.println(e.getMessage());
            }
```

```
        }

        while (true) {              try {
            System.out.print("Is this passenger
VIP? (y/n): ");
            String vipInput = scanner.nextLine().trim().toLowerCase();
if (!vipInput.equals("y") &&
!vipInput.equals("n"))                  throw new
IllegalArgumentException("Please enter 'y' for VIP or 'n' for regular.");
            isVIP = vipInput.equals("y");               break;
          } catch (IllegalArgumentException e) {
System.out.println(e.getMessage());
          }
        }

        passengers[i] = new TicketBooking(name, isVIP);
        if (isVIP) {
          passengers[i].setPriority(Thread.MAX_PRIOR
ITY);
        } else {
          passengers[i].setPriority(Thread.NORM_PRIO
RITY);
        }
      }

      System.out.println("\n===== Ticket Booking Process
=====");
      for (TicketBooking passenger : passengers) {
        passenger.start();
      }
scanner.close()
;
       }
}
```

OUTPUT

```
Enter the number of available seats: 3
Enter the number of passengers: 5

===== Passenger Details =====
Enter passenger name: a
Is this passenger VIP? (y/n): y
Enter passenger name: b
Is this passenger VIP? (y/n): y
Enter passenger name: c
Is this passenger VIP? (y/n): y
Enter passenger name: d
Is this passenger VIP? (y/n): n
Enter passenger name: e
Is this passenger VIP? (y/n): n

===== Ticket Booking Process =====
a booked a seat. (2 seats remaining)
d booked a seat. (1 seats remaining)
e booked a seat. (0 seats remaining)
b could not book a seat. No seats available.
c could not book a seat. No seats available.
```

4. Learning Outcome:

Employee Management (Easy Level):

1. Dynamic Data Handling: Master using an ArrayList for CRUD operations on employee data.
2. Robust Input Validation: Practice error handling with try/catch blocks and manage inputs using the Scanner class.
3. Modular Code Structure: Learn to design modular methods (add(), update(), remove(), search(), display(), find()) for specific operations.

Card Collection (Medium Level):

1. Collections Framework: Understand the use of TreeMap and TreeSet to store and organize data.
2. Custom Sorting: Implement a custom comparator using a defined order (CARD_ORDER) to sort cards.
3. Menu-Driven Interaction: Develop methods (addCard(), searchCard(), displayCards()) to interactively manage the card collection.

Ticket Booking System (Hard Level):

1. Multithreading: Learn to extend Thread and implement the run() method to execute concurrent tasks.
2. Thread Synchronization: Use synchronized blocks to ensure threadsafe operations and prevent double bookings.
3. Priority Management: Apply thread priorities (Thread.MAX_PRIORITY for VIPs, Thread.NORM_PRIORITY for regular passengers) to simulate prioritized processing.