Experiment - 4

Student Name: Garvi Dabas
UID: 23BCS11346
Branch: BE-CSE
Section/Group: KRG-2B
Semester: 5th
Date of Performance: 10/9/25
Subject Name: Project Based Learning in Java
Subject Code: 23CSH-304

Aim: To develop Java programs to manage product details, library systems, and student information using classes, inheritance, and abstraction.

Medium-level Problem-

Aim: Create a program to **collect and store cards**, and assist users in finding all cards of a given symbol using Collection interfaces.

Objective: To understand how to use **HashMap with ArrayList** for storing grouped data.

Procedure:

1. Define a Card class with symbol and number.
2. Use a `HashMap<String, ArrayList<Card>>`, where the key is the symbol.
3. Insert cards into the map, grouping by symbol.
4. Allow users to input a symbol to retrieve all matching cards.

Code -

```
package exp2;
import java.util.*;
class Card {
String symbol;
int number;
Card(String symbol, int number) {
this.symbol = symbol;
this.number = number;
}
public String toString() {
return symbol + " - " + number;
}
}
```

```
public class Medium {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
HashMap<String, ArrayList<Card>> cardMap = new HashMap<>();

cardMap.put("Spade", new ArrayList<>(Arrays.asList(new Card("Spade", 1), new Card("Spade",
3), new Card("Spade", 10))));
cardMap.put("Heart", new ArrayList<>(Arrays.asList(new Card("Heart", 2), new Card("Heart",
5))));

System.out.print("Enter symbol: ");
String symbol = sc.next();

if (cardMap.containsKey(symbol)) {
System.out.println("Cards with symbol '" + symbol + "':");
for (Card c : cardMap.get(symbol)) {
System.out.println(c);
}
} else {
System.out.println("No cards found with symbol " + symbol);
}
}
}
```

Output -

```
Cards with symbol 'Spade':
Spade - 1
Spade - 3
Spade - 10
```

Hard- Level Problem -

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: To understand **multithreading, thread synchronization, and thread priorities** in Java.

Procedure:

1. Create a TicketBooking class with a synchronized bookTicket() method.
2. Create customer threads (Normal & VIP).
3. Assign higher priority to VIP threads using setPriority().
4. Use synchronized methods to prevent double booking.
5. Display results showing correct booking order.

Code :

```
package exp2;
class TicketBooking {
private boolean isBooked = false;
public synchronized void bookTicket(String userType) {
if (!isBooked) {
System.out.println(userType + " booked Seat 1");
isBooked = true;
} else {
System.out.println(userType + " could not book. Seat already booked.");
}
}
}
class Customer extends Thread {
TicketBooking booking;
String userType;
Customer(TicketBooking booking, String userType, int priority) {
this.booking = booking;
this.userType = userType;
```

```
this.setPriority(priority);
}
public void run() {
booking.bookTicket(userType);
}
}
public class Hard {
public static void main(String[] args) {
TicketBooking booking = new TicketBooking();

Customer normal = new Customer(booking, "Normal User", Thread.MIN_PRIORITY);
Customer vip = new Customer(booking, "VIP User", Thread.MAX_PRIORITY);
vip.start();
normal.start();
}
}
```

Output:

```
VIP Thread booked Seat 1
Normal Thread could not book. Seat already booked.
```

Conclusion:

1. **Collections Framework:** Provides unified architecture for storing/manipulating data.
2. **ArrayList** → Dynamic arrays (duplicate elements allowed).
3. **HashMap** → Key-value pairs, unique keys, unsynchronized.
4. **Multithreading:** Allows concurrent execution.
5. **synchronized** → Prevents multiple threads accessing shared resource.
6. **setPriority()** → Controls execution preference of threads.
7. These concepts are widely used in real-world applications like banking, e-ticketing, and server request handling.