

## BASIC LEVEL PROJECT

---

Project Title: Daily Habit Tracker (Console Application)

Description:

- Collect daily habits from user input
- Store habits in a list
- Display completed and pending tasks
- Calculate weekly habit score

## JAVA PROGRAM:

```
package program;

import java.util.ArrayList;

import java.util.Scanner;

class Habit {

    String name;

    boolean completed;

    Habit(String name) {

        this.name = name;

        this.completed = false;

    }

}

public class DailyHabitTracker {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        ArrayList<Habit> habits = new ArrayList<>();

        System.out.println("=== Daily Habit Tracker ===");

        System.out.print("Enter number of habits for today: ");

        int n = sc.nextInt();

        sc.nextLine();

        for (int i = 1; i <= n; i++) {

            System.out.print("Enter habit " + i + ": ");

            String habitName = sc.nextLine();
```

```

        habits.add(new Habit(habitName));
    }

    for (Habit h : habits) {

        System.out.print("Did you complete \"" + h.name + "\" today? (yes/no): ");

        String status = sc.nextLine();

        if (status.equalsIgnoreCase("yes")) {

            h.completed = true;

        }

    }

    System.out.println("\nCompleted Habits:");

    int completedCount = 0;

    for (Habit h : habits) {

        if (h.completed) {

            System.out.println("- " + h.name);

            completedCount++;

        }

    }

    System.out.println("\nPending Habits:");

    for (Habit h : habits) {

        if (!h.completed) {

            System.out.println("- " + h.name);

        }

    }
}

```

```
int totalHabits = habits.size();
```

```
int weeklyScore = (completedCount * 100) / totalHabits;
```

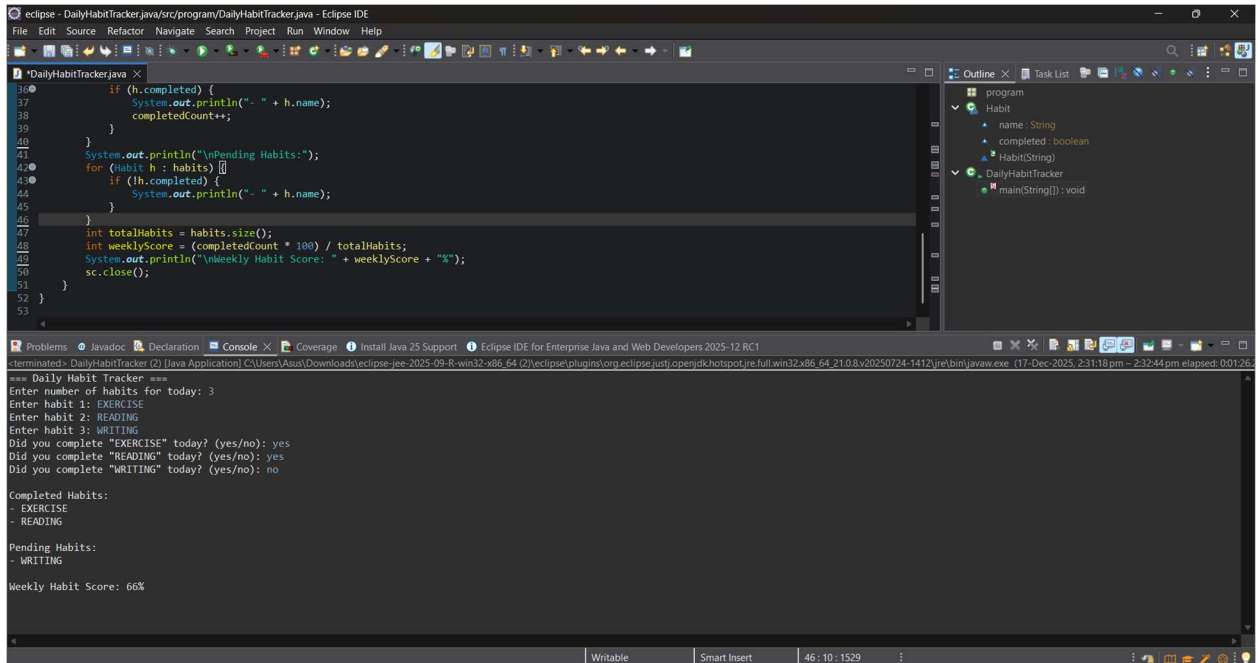
```
System.out.println("\nWeekly Habit Score: " + weeklyScore + "%");
```

```
sc.close();
```

```
}
```

```
}
```

## OUTPUT:



The screenshot shows the Eclipse IDE interface. The main editor displays the `DailyHabitTracker.java` file. The code includes a `main` method that prompts the user to enter the number of habits for today, then lists each habit and asks if it was completed. It then calculates and prints the weekly habit score as a percentage.

```
36     if (h.completed) {
37         System.out.println("- " + h.name);
38         completedCount++;
39     }
40 }
41 System.out.println("\nPending Habits:");
42 for (Habit h : habits) {
43     if (!h.completed) {
44         System.out.println("- " + h.name);
45     }
46 }
47 int totalHabits = habits.size();
48 int weeklyScore = (completedCount * 100) / totalHabits;
49 System.out.println("\nWeekly Habit Score: " + weeklyScore + "%");
50 sc.close();
51 }
52 }
53 }
```

The right-hand side of the IDE shows the Outline view with the following structure:

- program
  - Habit
    - name : String
    - completed : boolean
    - Habit(String)
  - DailyHabitTracker
    - main(String[]) : void

The bottom console window shows the program's execution output:

```
*** Daily Habit Tracker ***
Enter number of habits for today: 3
Enter habit 1: EXERCISE
Enter habit 2: READING
Enter habit 3: WRITING
Did you complete "EXERCISE" today? (yes/no): yes
Did you complete "READING" today? (yes/no): yes
Did you complete "WRITING" today? (yes/no): no

Completed Habits:
- EXERCISE
- READING

Pending Habits:
- WRITING

Weekly Habit Score: 66%
```

## MEDIUM LEVEL PROJECT

---

Project Title: Expense Splitter (Mini Splitwise Console App)

Requirements:

- Add users and expenses
- Auto-split amount
- Show payment summary
- Use OOP + Collections

## JAVA PROGRAM:

```
package project;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

    class User {
        int id;

        String name;

        User(int id, String name) {
            this.id = id;

            this.name = name;
        }
    }

    class Expense {
        User paidBy;

        double amount;

        ArrayList<User> participants;

        Expense(User paidBy, double amount, ArrayList<User> participants) {
            this.paidBy = paidBy;

            this.amount = amount;

            this.participants = participants;
        }
    }

    public class ExpenseSplitter {

        static ArrayList<User> users = new ArrayList<>();

        static ArrayList<Expense> expenses = new ArrayList<>();

        static HashMap<String, Double> balanceSheet = new HashMap<>();
```

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int choice;  
    do {  
        System.out.println("\n===== Expense Splitter Menu =====");  
        System.out.println("1. Add User");  
        System.out.println("2. Add Expense");  
        System.out.println("3. Show Summary");  
        System.out.println("4. Exit");  
        System.out.print("Enter choice: ");  
        choice = sc.nextInt();  
        switch (choice) {  
            case 1:  
                addUser(sc);  
                break;  
            case 2:  
                addExpense(sc);  
                break;  
            case 3:  
                showSummary();  
                break;  
            case 4:  
                System.out.println("Thank you for using Expense Splitter!");  
                break;  
            default:  
                System.out.println("Invalid choice!");  
        }  
    }
```

```

    } while (choice != 4);

    sc.close();
}

static void addUser(Scanner sc) {
    System.out.print("Enter User ID: ");
    int id = sc.nextInt();
    System.out.print("Enter User Name: ");
    String name = sc.next();
    users.add(new User(id, name));
    System.out.println("User added successfully!");
}

static void addExpense(Scanner sc) {
    System.out.println("Select payer ID:");
    for (User u : users) {
        System.out.println(u.id + " - " + u.name);
    }
    int payerId = sc.nextInt();
    User payer = null;
    for (User u : users) {
        if (u.id == payerId) {
            payer = u;
            break;
        }
    }
    System.out.print("Enter total amount: ");
    double amount = sc.nextDouble();

```



```

        System.out.print("Enter number of participants: ");
        int count = sc.nextInt();
        ArrayList<User> participants = new ArrayList<>();
        System.out.println("Enter participant IDs:");
        for (int i = 0; i < count; i++) {
            int pid = sc.nextInt();
            for (User u : users) {
                if (u.id == pid) {
                    participants.add(u);
                }
            }
        }
        Expense expense = new Expense(payer, amount, participants);
        expenses.add(expense);
        splitExpense(expense);
        System.out.println("Expense added successfully!");
    }

    static void splitExpense(Expense expense) {
        double splitAmount = expense.amount / expense.participants.size();
        for (User user : expense.participants) {
            if (!user.name.equals(expense.paidBy.name)) {
                String key = user.name + " owes " + expense.paidBy.name;
                balanceSheet.put(key, balanceSheet.getOrDefault(key, 0.0) +
splitAmount);
            }
        }
    }
}

```

```

static void showSummary() {

    System.out.println("\n===== Payment Summary =====");

    if (balanceSheet.isEmpty()) {

        System.out.println("No balances to show.");

    } else {

        for (String key : balanceSheet.keySet()) {

            System.out.println(key + " : ₹" + balanceSheet.get(key));

        }

    }

}

}

```

OUTPUT:

The screenshot shows the Eclipse IDE with the following components:

- Editor:** Displays the `ExpenseSplitter.java` file with the following code:
 

```

138 // Show Summary
139 static void showSummary() {
140     System.out.println("\n===== Payment Summary =====");
141
142     if (balanceSheet.isEmpty()) {
143         System.out.println("No balances to show.");
144     } else {
145         for (String key : balanceSheet.keySet()) {
146             System.out.println(key + " : ₹" + balanceSheet.get(key));
147         }
148     }
149 }
150

```
- Outline:** Shows the project structure with the following classes:
  - `Expense`
    - `paidBy : User`
    - `amount : double`
    - `participants : ArrayList<User>`
    - `Expense(User, double, ArrayList<User>)`
  - `ExpenseSplitter`
    - `users : ArrayList<User>`
    - `expenses : ArrayList<Expense>`
    - `balanceSheet : HashMap<String, Double>`
    - `main(String[]) : void`
- Console:** Shows the program's execution:
 

```

===== Expense Splitter Menu =====
1. Add User
2. Add Expense
3. Show Summary
4. Exit
Enter choice: 3

===== Payment Summary =====
No balances to show.

===== Expense Splitter Menu =====
1. Add User
2. Add Expense
3. Show Summary
4. Exit
Enter choice: 2
Select payer ID:
1
Enter total amount: 250
Enter number of participants: 3
Enter participant IDs:
2 3 4
Expense added successfully!

===== Expense Splitter Menu =====
1. Add User

```

## ADVANCED LEVEL PROJECT

---

Project Title: Smart Parking Lot System

Requirements:

- Add/Remove vehicles
- Generate parking tickets
- Calculate charges
- File handling + Custom exceptions

## JAVA PROGRAM:

```
package program;

import java.io.*;
import java.time.*;
import java.util.*;

class ParkingFullException extends Exception {

    public ParkingFullException(String message) {

        super(message);

    }

}

class Vehicle implements Serializable {

    private String vehicleNumber;

    private String vehicleType;

    public Vehicle(String vehicleNumber, String vehicleType) {

        this.vehicleNumber = vehicleNumber;

        this.vehicleType = vehicleType;

    }

    public String getVehicleNumber() {

        return vehicleNumber;

    }

    public String getVehicleType() {

        return vehicleType;

    }

}

class ParkingTicket implements Serializable {

    private Vehicle vehicle;

    private LocalDateTime entryTime;
```

```

public ParkingTicket(Vehicle vehicle) {
    this.vehicle = vehicle;
    this.entryTime = LocalDateTime.now();
}

public Vehicle getVehicle() {
    return vehicle;
}

public LocalDateTime getEntryTime() {
    return entryTime;
}
}

class ParkingLot {
    private int capacity;
    private Map<String, ParkingTicket> parkedVehicles;
    private static final String FILE_NAME = "parking_data.dat";
    public ParkingLot(int capacity) {
        this.capacity = capacity;
        parkedVehicles = new HashMap<>();
        loadFromFile();
    }

    public void parkVehicle(Vehicle vehicle) throws ParkingFullException {
        if (parkedVehicles.size() >= capacity) {
            throw new ParkingFullException("Parking lot is full!");
        }

        ParkingTicket ticket = new ParkingTicket(vehicle);
        parkedVehicles.put(vehicle.getVehicleNumber(), ticket);
        saveToFile();
    }
}

```

```

        System.out.println("Vehicle parked successfully. Ticket generated.");
    }

    public void removeVehicle(String vehicleNumber) {
        ParkingTicket ticket = parkedVehicles.remove(vehicleNumber);
        if (ticket == null) {
            System.out.println("Vehicle not found.");
            return;
        }
        double charges = calculateCharges(ticket);
        saveToFile();
        System.out.println("Vehicle removed successfully.");
        System.out.println("Parking Charges: Rs. " + charges);
    }

    private double calculateCharges(ParkingTicket ticket) {
        long hours = Duration.between(ticket.getEntryTime(), LocalDateTime.now()).toHours();
        if (hours == 0) hours = 1;
        if (ticket.getVehicle().getVehicleType().equalsIgnoreCase("CAR")) {
            return hours * 50; // Rs.50 per hour
        } else {
            return hours * 20; // Rs.20 per hour
        }
    }

    private void saveToFile() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(FILE_NAME))) {
            oos.writeObject(parkedVehicles);
        } catch (IOException e) {

```

```

        System.out.println("Error saving data.");
    }
}

@SuppressWarnings("unchecked")
private void loadFromFile() {
    File file = new File(FILE_NAME);
    if (!file.exists()) return;

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME)))
    {
        parkedVehicles = (HashMap<String, ParkingTicket>) ois.readObject();
    } catch (Exception e) {
        System.out.println("Error loading data.");
    }
}

}

}

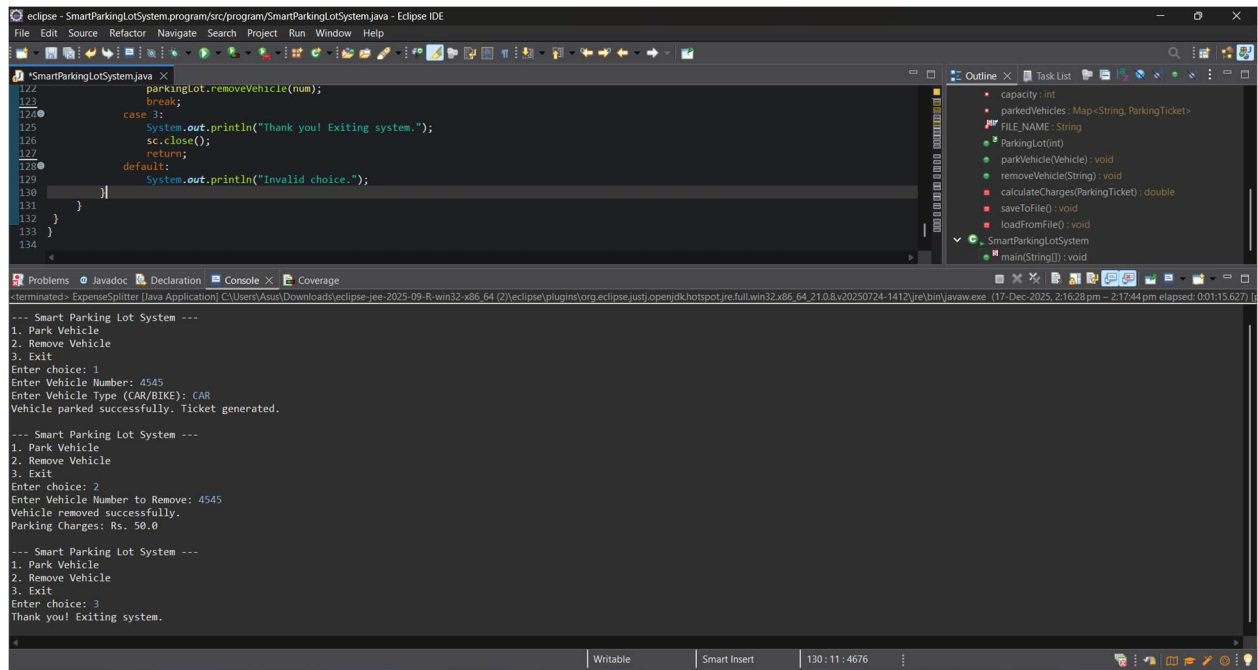
public class SmartParkingLotSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ParkingLot parkingLot = new ParkingLot(5);
        while (true) {
            System.out.println("\n--- Smart Parking Lot System ---");
            System.out.println("1. Park Vehicle");
            System.out.println("2. Remove Vehicle");
            System.out.println("3. Exit");
            System.out.print("Enter choice: ");
            int choice = sc.nextInt();
            sc.nextLine();

```

```
switch (choice) {  
    case 1:  
        try {  
            System.out.print("Enter Vehicle Number: ");  
            String number = sc.nextLine();  
            System.out.print("Enter Vehicle Type (CAR/BIKE): ");  
            String type = sc.nextLine();  
            Vehicle vehicle = new Vehicle(number, type);  
            parkingLot.parkVehicle(vehicle);  
        } catch (ParkingFullException e) {  
            System.out.println(e.getMessage());  
        }  
        break;  
    case 2:  
        System.out.print("Enter Vehicle Number to Remove: ");  
        String num = sc.nextLine();  
        parkingLot.removeVehicle(num);  
        break;  
    case 3:  
        System.out.println("Thank you! Exiting system.");  
        sc.close();  
        return;  
    default:  
        System.out.println("Invalid choice.");  
        }  
    }  
}
```



## OUTPUT:



The screenshot displays the Eclipse IDE interface. The main editor window shows the `SmartParkingLotSystem.java` file. The code includes a `switch` statement for handling user choices. The console window at the bottom shows the program's execution output, which includes prompts for parking and removing vehicles, and the generation of parking tickets.

```
SmartParkingLotSystem.java
122 parkingLot.removeVehicle(num);
123 break;
124 case 3:
125     System.out.println("Thank you! Exiting system.");
126     sc.close();
127     return;
128 default:
129     System.out.println("Invalid choice.");
130 }
131 }
132 }
133 }
134 }

Outline
capacity : int
parkedVehicles : Map<String, ParkingTicket>
FILE_NAME : String
ParkingLot(int)
parkVehicle(Vehicle) : void
removeVehicle(String) : void
calculateCharges(ParkingTicket) : double
saveToFile() : void
loadFromFile() : void
SmartParkingLotSystem
main(String[]) : void

Problems Javadoc Declaration Console Coverage
<terminated> ExpenseSplitter [Java Application] C:\Users\Asus\Downloads\eclipse-jee-2025-09-R-win32-x86_64 (2)\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64-21.0.8.v20250724-1412\jre\bin\javaw.exe (17-Dec-2025, 2:16:28 pm - 2:17:44 pm elapsed: 001:15:627) [

--- Smart Parking Lot System ---
1. Park Vehicle
2. Remove Vehicle
3. Exit
Enter choice: 1
Enter Vehicle Number: 4545
Enter Vehicle Type (CAR/BIKE): CAR
Vehicle parked successfully. Ticket generated.

--- Smart Parking Lot System ---
1. Park Vehicle
2. Remove Vehicle
3. Exit
Enter choice: 2
Enter Vehicle Number to Remove: 4545
Vehicle removed successfully.
Parking Charges: Rs. 50.0

--- Smart Parking Lot System ---
1. Park Vehicle
2. Remove Vehicle
3. Exit
Enter choice: 3
Thank you! Exiting system.
```