Charging Station Management System

Arjun Veeramony(7219119)
Angel Mary (7218965)
Swathi Chandrashekaraiah (7218877)
Farhaad Sheikh Mohammed(7219055)

Charging Station

1. Overview of the Project
2. Objectives and Goals
3. Class Diagram
4. Implementation
5. OutPut
6. Unit testing
7. Test Results
8. Work distribution

Overview of the Project:

The Charging Station Management System is a Java-based project designed to efficiently manage electric vehicle charging stations.

The system facilitates the allocation of charging slots to vehicles, monitors charging durations, and handles waiting lists during peak times.

Objectives and Goals:

Efficient Resource Management: Optimize the utilization of charging stations and slots to ensure a smooth charging experience for electric vehicle owners.

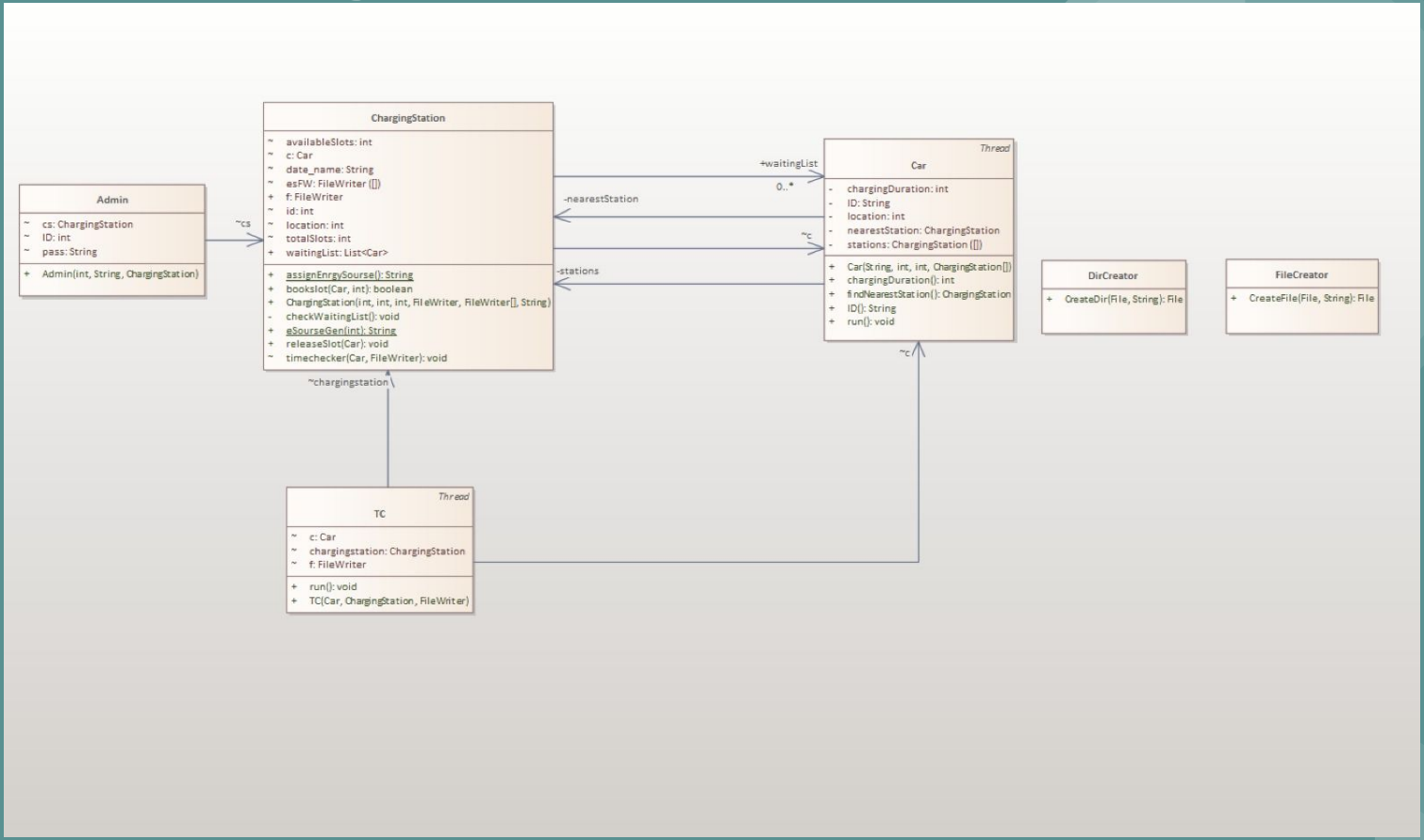
Real-time Monitoring: Provide real-time monitoring of charging activities, including slot availability, charging durations, and energy sources.

User Interaction: Enable administrators to manage the system, users to access charging stations, and view log information.

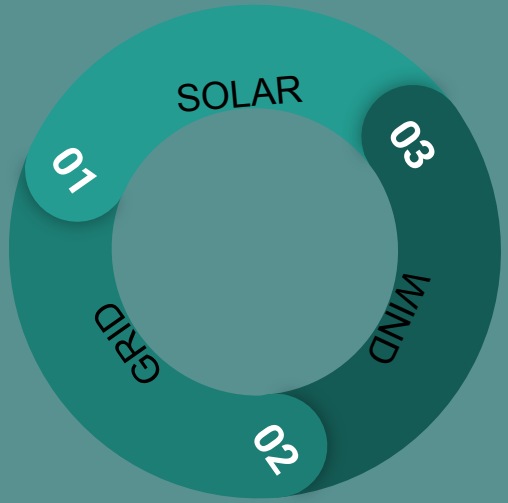
Multi-Threading: Utilize multi-threading to handle concurrent charging sessions and manage waiting lists effectively.

Exception Handling: Implement robust exception handling to manage errors and unexpected events during system operation.

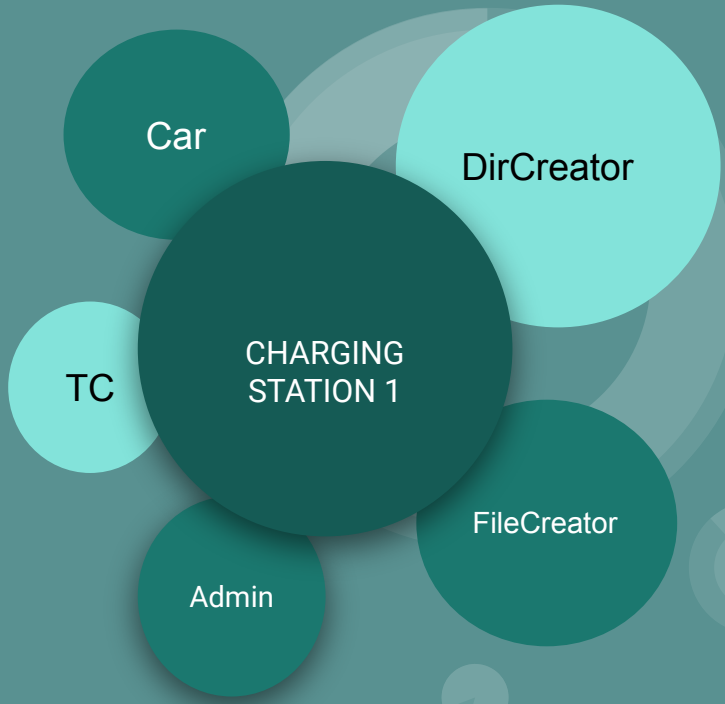
Class Diagram:



Implementation



Energy Sources Used



Classes Used

Charging Station Overview

- ChargingStation Class represents a charging station where cars can charge their batteries.
- Attributes include id, location, totalSlots, availableSlots, and waitingList.
- Manages the booking and release of charging slots for cars.
- The ChargingStation class is instantiated for each charging station in the system.

Car Booking Process

- When a car requests a charging slot, the bookslot() method of the ChargingStation class is called.
- The method checks for available slots and assigns one to the car if available.
- If no slots are available, the car is added to the waiting list.

Time Checker Thread

- The timechecker() method of the ChargingStation class creates and starts a TC thread for each car added to the waiting list.
- It ensures that cars waiting for an particular period are notified about the availability of slots, else removed from the waiting list if it has to wait for very long period of time.

Slot Release Process

- When a car completes charging, the releaseSlot() method of the ChargingStation class is called.
- The method increments the available slots count and notifies cars in the waiting list about slot availability.

Log File Access for Admin

- Log files are organized based on charging station ID, energy sources, and date for easy navigation.
- Admins have privileged access to log files generated by the charging stations.
- Admins can view and manage log files through the system interface.
- Log files contain crucial information such as charging activities, energy sources, and station utilization.

OutPut:

```
PS C:\Users\achua\OneDrive\Desktop\CapStoneProject\Capstone-Project> java Main
C:\Users\achua\OneDrive\Desktop\CapStoneProject\Capstone-Project\StationLogFiles\Station1LogFiles
car5 will be charging for 3 Minutes in Station2 ,with Energy Sourse as: Power Grid
car2 will be charging for 3 Minutes in Station1 ,with Energy Sourse as: Wind
car8 will be charging for 1 Minutes in Station2 ,with Energy Sourse as: Solar
car4 will be charging for 3 Minutes in Station1 ,with Energy Sourse as: Power Grid
car7 is added to the waiting list.
car3 is added to the waiting list.
car6 is added to the waiting list.
car1 is added to the waiting list.
car8 Charging slot released. Available slots: 1
car7 will be charging for 4 Minutes in Station2 ,with Energy Sourse as: Power Grid
car7 moved from the waiting list and got a slot.
Wait called for:car7
car5 Charging slot released. Available slots: 1
car4 Charging slot released. Available slots: 1
car6 will be charging for 2 Minutes in Station2 ,with Energy Sourse as: Solar
car3 will be charging for 1 Minutes in Station1 ,with Energy Sourse as: Power Grid
car6 moved from the waiting list and got a slot.
car3 moved from the waiting list and got a slot.
Wait called for:car6
Wait called for:car3
car2 Charging slot released. Available slots: 1
car1 will be charging for 8 Minutes in Station1 ,with Energy Sourse as: Wind
car1 moved from the waiting list and got a slot.
Wait called for:car1
All cars have finished their charging. Program terminated.
================================================================================
```

```
Would You like to View log files(yes/no)yes
Would you like to view Station log files(yes/no)yes
Station1 or Station2 or ESourse or Date_Wise:Station1
Enter your ID:001
Enter your password:admin1
**********************************************************************
File Name: Station1LogFiles
File Length: 450
File Last Modified: 1707491395747

car2 Charged for 3 Minutes in Station1 ,with Energy Sourse as: Wind
car4 Charged for 3 Minutes in Station1 ,with Energy Sourse as: Power Grid
car3 is added to the waiting list.
car1 is added to the waiting list.
car4 Charging slot released. Available slots: 1
car3 Charged for 1 Minutes in Station1 ,with Energy Sourse as: Power Grid
car2 Charging slot released. Available slots: 1
car1 Charged for 8 Minutes in Station1 ,with Energy Sourse as: Wind
**********************************************************************
Would u like to delete the file(yes/no):
no
Thank u then.----End of Program----
PS C:\Users\achua\OneDrive\Desktop\CapStoneProject\Capstone-Project>
```

Unit Testing(Junit)

Testing Overview:

- In this section, we'll discuss the testing process for our Charging Station Management System.

- We have implemented a series of test cases to ensure the robustness and reliability of our system using Junit testing frame work.

- Our testing covers various aspects of the system functionality, including:
  1. Identifying the nearest charging station for cars.
  2. Assigning energy sources within valid ranges.
  3. Validating thread synchronization for slot booking and release.

Test Case - Find Nearest Station

Test Purpose:

Verifies whether cars can correctly identify the nearest charging station.

Test Description:

- Creates three charging stations at different locations.
- Creates cars at varying distances from the stations.
- Tests if each car correctly identifies the nearest charging station using the findNearestStation() method.

Test Steps:

- Create charging stations at locations 5, 10, and 15.
- Create cars with locations 7, 12, and 18.
- Invoke the findNearestStation() method for each car.
- Validate if the identified stations match the expected nearest stations.

Expected Outcome:

- Car 1 (location 7) should identify station 1 (location 5) as the nearest.
- Car 2 (location 12) should identify station 2 (location 10) as the nearest.
- Car 3 (location 18) should identify station 3 (location 15) as the nearest.

Test Case - Assign Energy Source

Test Purpose:
- Validates the assignment of energy sources within the valid range.

Test Description:
- Creates a charging station with a valid number of available slots.
- Iterates multiple times to assign energy sources.
- Checks if all three types of energy sources (Solar, Wind, Power Grid) are assigned.

Test Steps:
- Create a charging station with valid available slots.
- Iterate to assign energy sources multiple times.
- Check if all three energy sources are assigned within the valid range.

Expected Outcome:
- All three energy sources (Solar, Wind, Power Grid) should be assigned within the valid range.

Test Case - Thread Synchronization

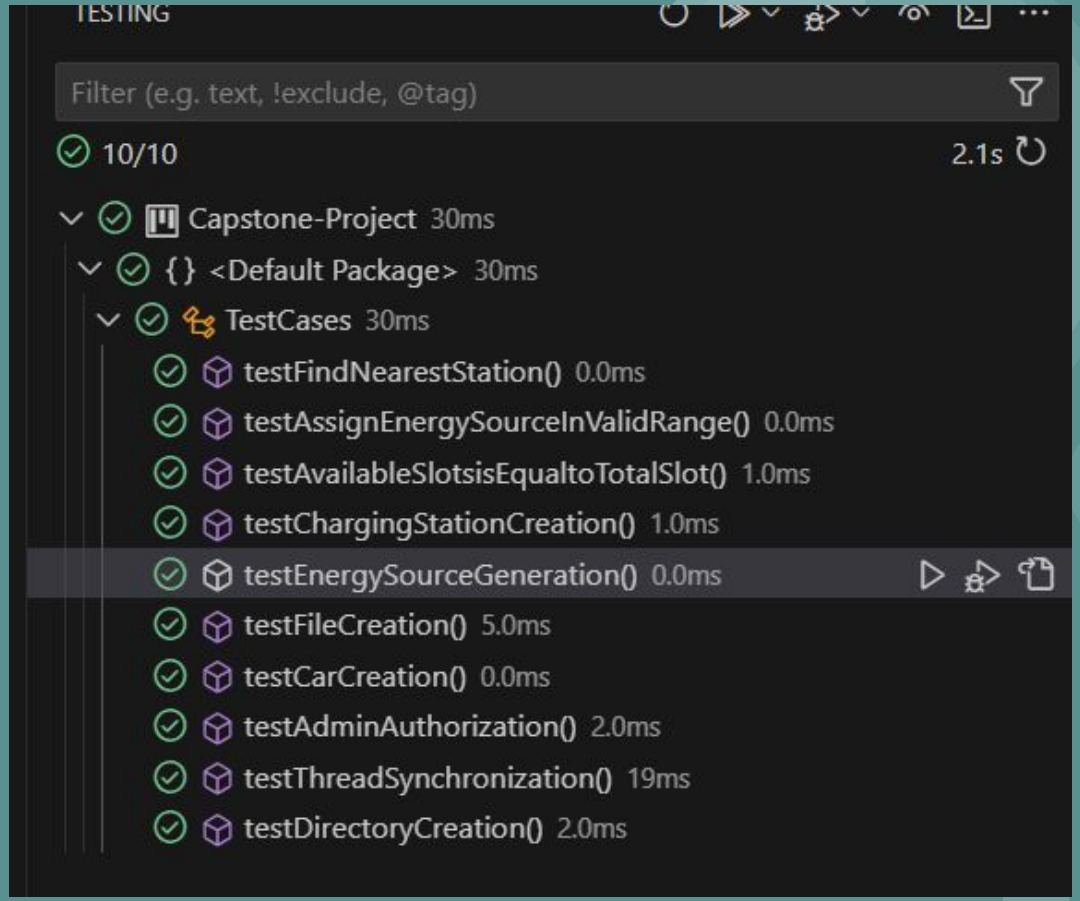
Test Purpose:
- Validates thread synchronization within the charging station.

Test Description:
- Creates a charging station with limited available slots.
- Simulates multiple cars trying to book slots simultaneously.
- Checks if thread synchronization ensures correct slot booking and release.

Test Steps:
- Create a charging station with limited available slots.
- Starts 2 threads representing cars trying to book slots simultaneously.
- Validate if thread is synchronized or not by checking the available slot after join().

Expected Outcome:
- If the Threads are synchronized, the value of available slots will be 2 after checking it after join() function.

Test Results:



Task Distribution

1.**Farhaad**- Car class creation, Implementing Threading concepts inside Car, Class Diagrams, Testing.

2.Arjun- Energy Source Allocation,Slot Booking and waiting list creation, Testing.

3.Angel- Log Files creation, File Writer, Testing

4.Swathi- Admin Class creation, allowing admins to access log files, Testing

Thank You….