Capstone Project

GROUP3

Roles:

Linxin Zhang: Design of the User Interfaces

Yanal Al Halabi: Management System and Simulation

Anjali Bodke: Initial House Configuration

Franklin Viegas: Creation of Unit Testing

Functional Requirements

Application Initialization

- The application can initialize the system by loading configurations (e.g., devices, batteries, and energy sources)

Device Management

- Users can add, remove, and list devices.

Battery Management

- Users can list all batteries.
- Users can start and stop charging batteries.
- Users can start and stop powering devices using battery.

Energy Source Management

- Users can add, remove, and list energy sources.
- Users can toggle the state (active/inactive) of an energy source.

System Monitoring

- The system can monitor total power consumption and battery charge periodically.
- The system can log warnings when power consumption exceeds available battery charge.

Logging

- The system can log events related to devices, batteries, and energy sources (e.g., addition, removal, state changes).
- Logs can be categorized by type (e.g., DEVICE, BATTERY, ENERGY, SYSTEM).
- Users can search logs by name or date.
- Users can delete and archive logs.

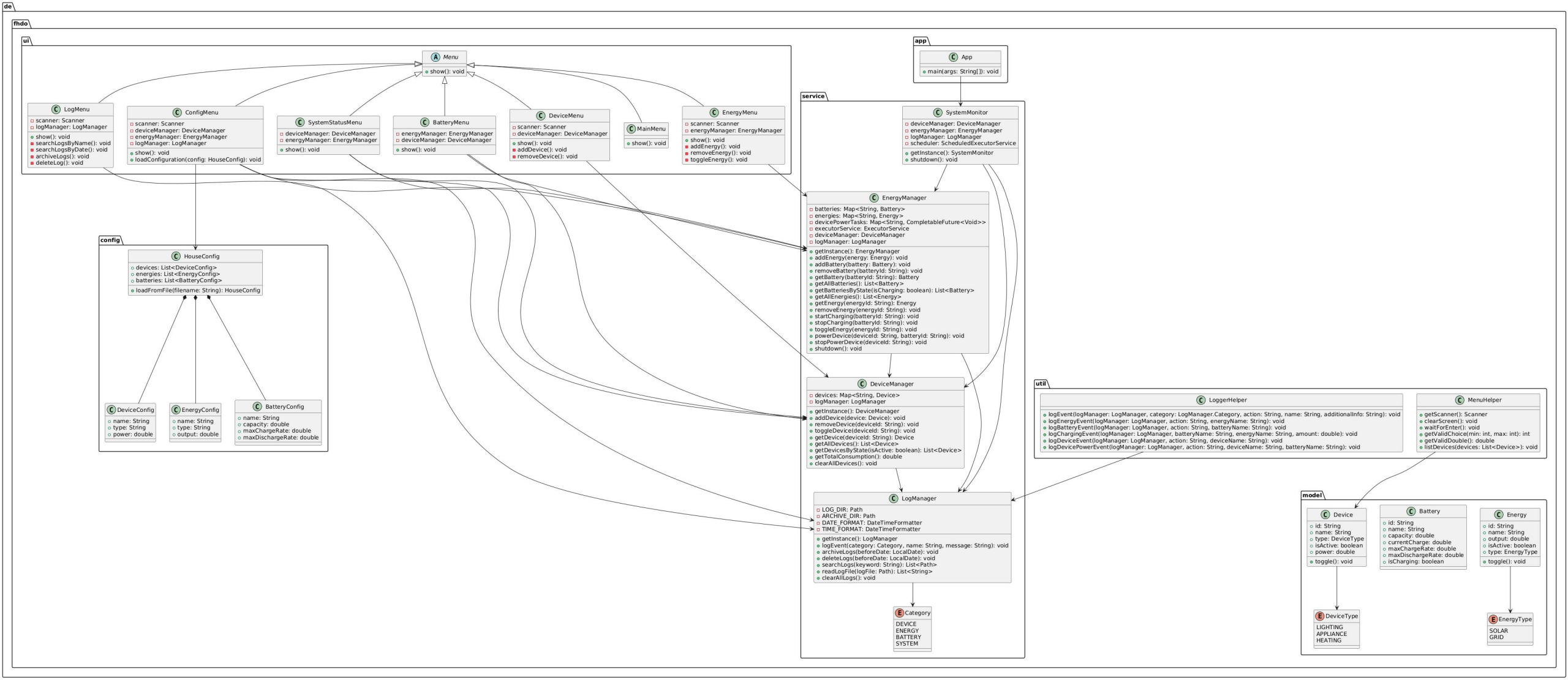
User Interface

- The system can provide interactive menus for managing:
  - Devices, Batteries, Energy sources, Logs, System configuration, Overall system status
- Each menu can validate user inputs and provide feedback.

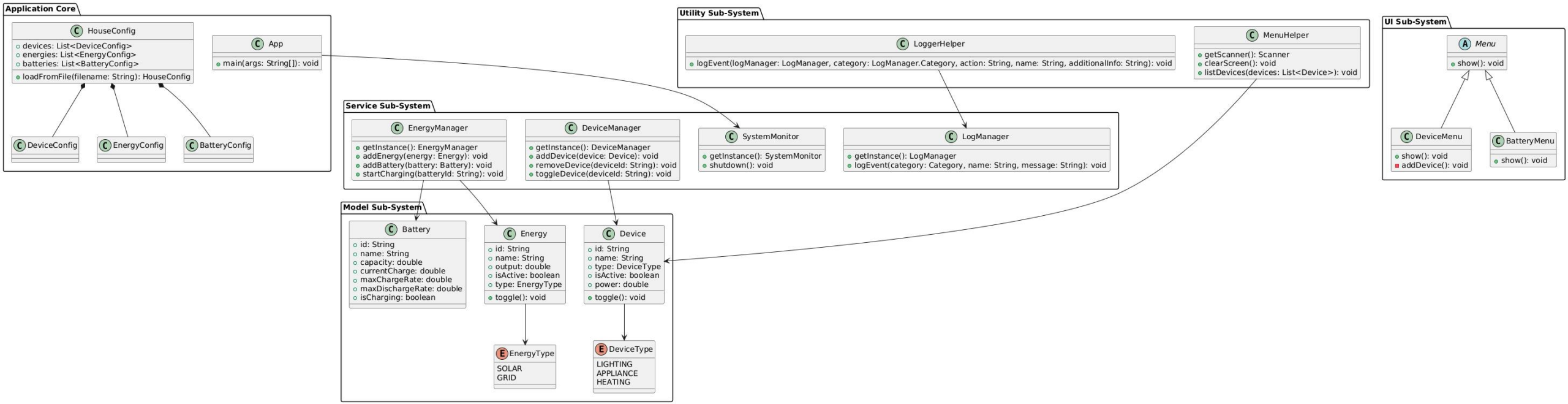
Configuration Loading

- Users can load a new configuration file.
- The configuration file can define devices, batteries, and energy sources with attributes like name, type, and capacity.

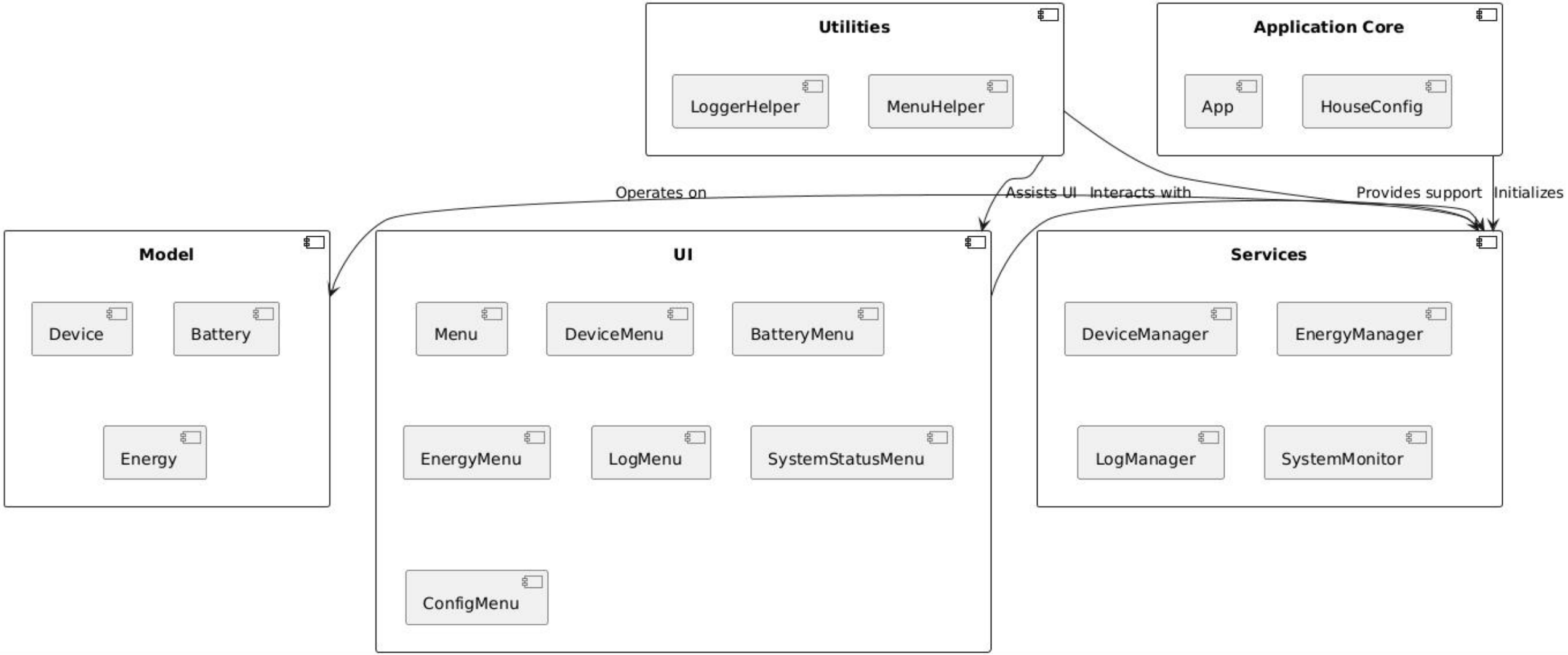
Class Diagram



Sub-systems Diagram



Component Diagram



Management I/O in the System

```java
public class LogManager {
    private static volatile LogManager instance;

    private final Path LOG_DIR = Paths.get("logs");
    private final Path ARCHIVE_DIR = LOG_DIR.resolve("archive");
    public final DateTimeFormatter DATE_FORMAT = DateTimeFormatter.ofPattern("yyyyMMdd");
    private final DateTimeFormatter TIME_FORMAT = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    @Getter
    @AllArgsConstructor
    public enum Category {
        DEVICE("device"), ENERGY("energy"), BATTERY("battery"), SYSTEM("system");

        private final String value;
    }
```

```java
private void initializeDirectories() {
    try {
        Files.createDirectories(LOG_DIR);
        Files.createDirectories(ARCHIVE_DIR);
        for (Category category : Category.values()) {
            Files.createDirectories(LOG_DIR.resolve(category.getValue()));
        }
    } catch (IOException e) {
        Log.error("Failed to initialize log directories", e);
    }
}
```

Events of I/O in the System

```java
public void logEvent(Category category, String name, String message) {
    LocalDateTime now = LocalDateTime.now();
    String date = now.format(DATE_FORMAT);
    Path logFile = LOG_DIR.resolve(category.getValue()).resolve(String.format("%s_%s.log", name, date));
    Path systemLogFile = LOG_DIR.resolve(Category.SYSTEM.getValue()).resolve("system_" + date + ".log");

    writeToLog(logFile, now, message);
    if (!category.equals(Category.SYSTEM)) {
        writeToLog(systemLogFile, now, String.format("%s: %s", category, message));
    }
}
```

```java
public void deleteLogs(LocalDate beforeDate) {
    for (Category category : Category.values()) {
        Path categoryDir = LOG_DIR.resolve(category.getValue());
        if (!Files.exists(categoryDir)) continue;

        try (Stream<Path> paths = Files.list(categoryDir)) {
            paths.filter(path -> isLogFileBeforeDate(path, beforeDate, DATE_FORMAT)).forEach(this::deleteLogFile);
        } catch (IOException e) {
            Log.error("Error deleting logs in category: {}", category, e);
        }
    }
}
```

Managing Files of I/O in the System

```java
public void writeToLog(Path logFile, LocalDateTime timestamp, String message) {
    try {
        Files.createDirectories(logFile.getParent());
        try (BufferedWriter writer = Files.newBufferedWriter(logFile, StandardOpenOption.CREATE, StandardOpenOption.APPEND)) {
            writer.write(String.format("[%s] %s%n", timestamp.format(TIME_FORMAT), message));
        }
    } catch (IOException e) {
        Log.error("Failed to write to log file: {}", logFile, e);
    }
}
```

```java
public void archiveLogFile(Path logFile, ZipOutputStream zos) {
    try {
        ZipEntry entry = new ZipEntry(logFile.getParent().getFileName() + "/" + logFile.getFileName().toString());
        zos.putNextEntry(entry);
        Files.copy(logFile, zos);
        zos.closeEntry();
        Files.delete(logFile);
    } catch (IOException e) {
        Log.error("Failed to archive log file: {}", logFile, e);
    }
}

public void deleteLogFile(Path logFile) {
    try {
        Files.delete(logFile);
        Log.info("Deleted log file: {}", logFile);
    } catch (IOException e) {
        Log.error("Failed to delete log file: {}", logFile, e);
    }
}
```

Users > yanal > git > capstone_project > logs >

Sort View

Name	Date modified	Type	Si
archive	24/11/2024 21:00	File folder	
battery	24/11/2024 21:00	File folder	
device	26/11/2024 12:03	File folder	
energy	26/11/2024 12:04	File folder	
system	26/11/2024 12:03	File folder	

Users > yanal > git > capstone_project > logs > device

Sort View

Name	Date modified	Type	Size
temp1_20241124	24/11/2024 22:18	Text Document	1 KB
temp1_20241126	26/11/2024 12:03	Text Document	1 KB
yanal_20241124	24/11/2024 22:06	Text Document	1 KB

Users > yanal > git > capstone_project > logs > energy

Sort View

Name	Date modified	Type	Size
charger_20241124	24/11/2024 22:04	Text Document	1 KB
energy1_20241126	26/11/2024 12:04	Text Document	1 KB
wind_20241124	24/11/2024 22:02	Text Document	1 KB

Concurrency (EnergyManager)

```
private void manageChargingTasks(Battery battery, List<Energy> activeEnergies) {
    List<CompletableFuture<Void>> tasks = activeEnergies.stream()
            .map(energy -> CompletableFuture.runAsync(() -> chargeFromEnergy(battery, energy), executorService))
            .toList();

    try {
        while (battery.isCharging()) {
            if (tasks.stream().allMatch(CompletableFuture::isDone)) {
                battery.setCharging(false);
                break;
            }
            Thread.sleep(3000);
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
```

Concurrency (EnergyManager)

```java
private void chargeFromEnergy(Battery battery, Energy energy) {
    try {
        while (battery.isCharging()) {
            synchronized (battery) {
                double availablePower = energy.getOutput();
                double batteryDeficit = battery.getCapacity() - battery.getCurrentCharge();
                double deviceConsumption = deviceManager.getTotalConsumption();

                double chargePower = Math.min(battery.getMaxChargeRate(), availablePower);

                if (batteryDeficit <= 0 && deviceConsumption <= 0) {
                    break;
                }

                double netCharge = chargePower - deviceConsumption;
                if (netCharge > 0) {
                    double chargeAmount = Math.min(netCharge, batteryDeficit);
                    battery.setCurrentCharge(battery.getCurrentCharge() + chargeAmount);
                    LoggerHelper.logChargingEvent(logManager, battery.getName(), energy.getName(), chargeAmount);
                } else {
                    battery.setCurrentCharge(Math.max(0, battery.getCurrentCharge() + netCharge));
                    LoggerHelper.logChargingEvent(logManager, battery.getName(), energy.getName(), netCharge);
                }
            }
            Thread.sleep(3000);
        }
```

Concurrency (EnergyManager)

```java
private void manageDevicePowerTask(Device device, Battery battery) {
    try {
        while (device.isActive()) {
            synchronized (battery) {
                double consumption = device.getPower();
                if (battery.getCurrentCharge() >= consumption) {
                    battery.setCurrentCharge(battery.getCurrentCharge() - consumption);
                    LoggerHelper.logDevicePowerEvent(logManager, "Consuming power", device.getName(), battery.getName());
                } else {
                    log.info("Battery {} does not have enough charge to power the device {}", battery.getId(), device.getName());
                    device.setActive(false);
                    LoggerHelper.logDevicePowerEvent(logManager, "Powered off due to low battery", device.getName(), battery.getName());
                    break;
                }
            }
            Thread.sleep(1000);
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```

Design of User Interface

```
=== Smart House Energy Management System ===
1. Manage Devices
2. Manage Energies
3. Manage Batteries
4. View System Status
5. Manage Logs
6. Load Configuration
0. Exit

Please select an option (0-6):
```

```
1
=== Device Management ===
1. Add New Device
2. List All Devices
3. Remove Device
0. Return to Main Menu

Please select an option (0-3):
```

```
2
=== Energy Management ===
1. Add New Energy
2. List All Energies
3. Remove Energy
4. Toggle Energy State
0. Return to Main Menu

Please select an option (0-4):
```

```
3
=== Battery Management ===
1. List All Batteries
2. Start Battery Charging
3. Stop Battery Charging
4. Power Device from Battery
5. Stop Battery Power Supply
0. Return to Main Menu

Please select an option (0-5):
```

Initial Configuration

```
6
=== Load Configuration File ===
Please enter the configuration file path:
(Press Enter to use the default path: src/main/resources/config/house_config.yml)

Loading configuration from: src/main/resources/config/house_config.yml
Continue? (y/n, press Enter to confirm)

Preparing to clear existing data...
Continue? (y/n, press Enter to confirm)

20:05:37.079 [main] INFO de.fhdo.service.DeviceManager -- All devices have been cleared.
20:05:37.083 [main] INFO de.fhdo.service.EnergyManager -- All energies have been cleared
20:05:37.083 [main] INFO de.fhdo.service.EnergyManager -- All batteries have been cleared
20:05:37.090 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\device\temp1_20241124.log
20:05:37.091 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\device\temp1_20241126.log
20:05:37.091 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\device\yanal_20241124.log
20:05:37.092 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\energy\charger_20241124.log
20:05:37.092 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\energy\energy1_20241126.log
20:05:37.092 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\energy\wind_20241124.log
20:05:37.093 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\system\System Monitor_20241124.log
20:05:37.093 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\system\System Monitor_20241126.log
20:05:37.094 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\system\system_20241124.log
20:05:37.094 [main] INFO de.fhdo.service.LogManager -- Deleted log file: logs\system\system_20241126.log

Configuration loaded! Total: 2 devices, 2 energy, 1 batteries

Press Enter to continue...
```

Initial Configuration

```
2
Device #1
ID: b25277f9-e528-48b2-8bb3-491744288aed
Name: Fridge
Type: APPLIANCE
Power: 600.00 W
Status: Inactive

Device #2
ID: 358fc9c1-2637-40e8-918d-9ce8d098cc6f
Name: Living Room Light
Type: LIGHTING
Power: 100.00 W
Status: Inactive
```

```
2
Energy #1
ID: a1722d4e-7be1-422f-999f-380d52c112e0
Name: Grid Power
Type: GRID
Output: 7000.00 W
Status: Inactive

Energy #2
ID: 05322126-bdf8-41a8-a0b9-b6289d838945
Name: Solar Panels
Type: SOLAR
Output: 2000.00 W
Status: Inactive
```

```
1
Battery #1
ID: cf07d4b3-47c1-4dea-8652-1dfd32202827
Name: Main Battery
Capacity: 10000.00
Current Charge: 0.00
Max Charge Rate: 800.00
Max Discharge Rate: 900.00
Status: Not Charging
```

Unit Test

SmartHouseTestSuite.java

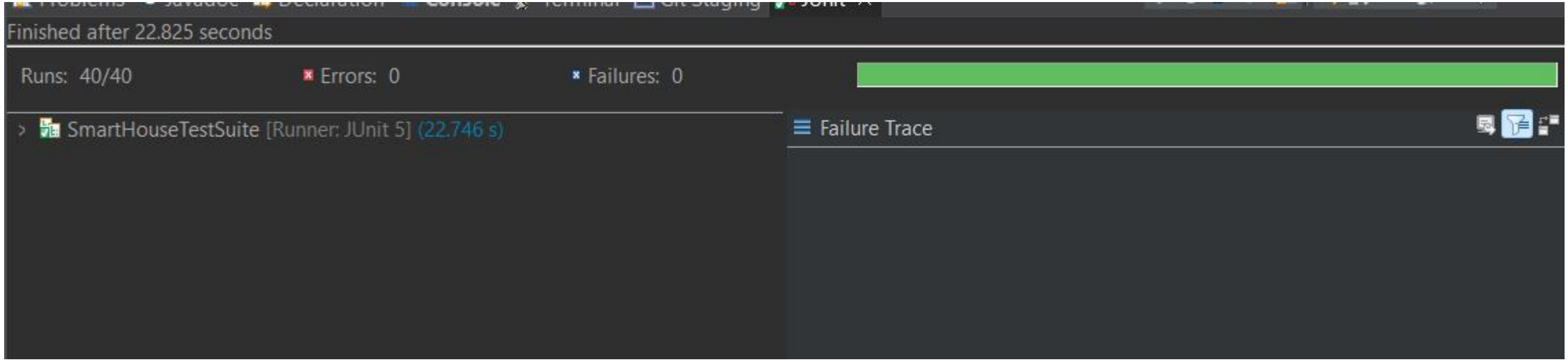
```java
package de.fhdo;


import org.junit.platform.suite.api.SelectClasses;

@Suite
@SelectClasses({
    HouseConfigTest.class,
    LogManagerTest.class,
    DeviceManagerTest.class,
    EnergyManagerTest.class,
    SystemMonitorTest.class
})
public class SmartHouseTestSuite {
}
```

Unit Test of the System

HouseConfigTest.java

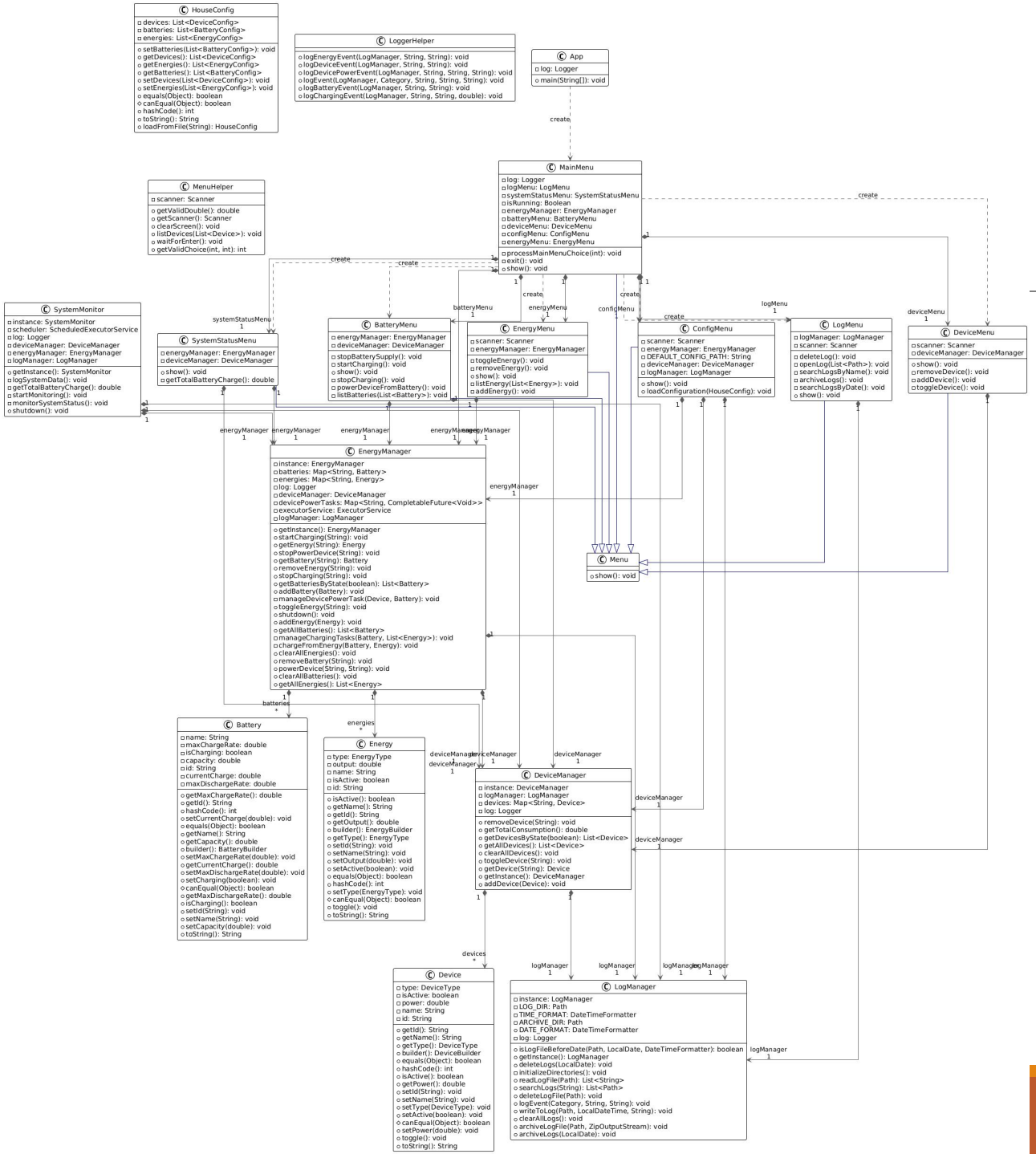
```java
public class HouseConfigTest {

    @Test
    void testLoadFromFile() throws IOException {
        HouseConfig config = HouseConfig.loadFromFile("src/test/resources/house_config.yml");

        assertNotNull(config);
        assertNotNull(config.getDevices());
        assertNotNull(config.getEnergies());
        assertNotNull(config.getBatteries());

        assertEquals(1, config.getDevices().size());
        HouseConfig.DeviceConfig firstDevice = config.getDevices().get(0);
        assertEquals("Living Room Lights", firstDevice.getName());
        assertEquals("LIGHTING", firstDevice.getType());
        assertEquals(100.0, firstDevice.getPower());

        assertEquals(1, config.getEnergies().size());
        HouseConfig.EnergyConfig firstEnergy = config.getEnergies().get(0);
        assertEquals("Solar Panels", firstEnergy.getName());
        assertEquals("SOLAR", firstEnergy.getType());
        assertEquals(5000.0, firstEnergy.getOutput());

        assertEquals(1, config.getBatteries().size());
        HouseConfig.BatteryConfig battery = config.getBatteries().get(0);
        assertEquals("Main Battery", battery.getName());
        assertEquals(10000.0, battery.getCapacity());
        assertEquals(2000.0, battery.getMaxChargeRate());
        assertEquals(2000.0, battery.getMaxDischargeRate());
    }

    @Test
    void testLoadFromNonExistentFile() {
        assertThrows(IOException.class, () ->
            HouseConfig.loadFromFile("non_existent_file.yml")
        );
```

Unit Test Results

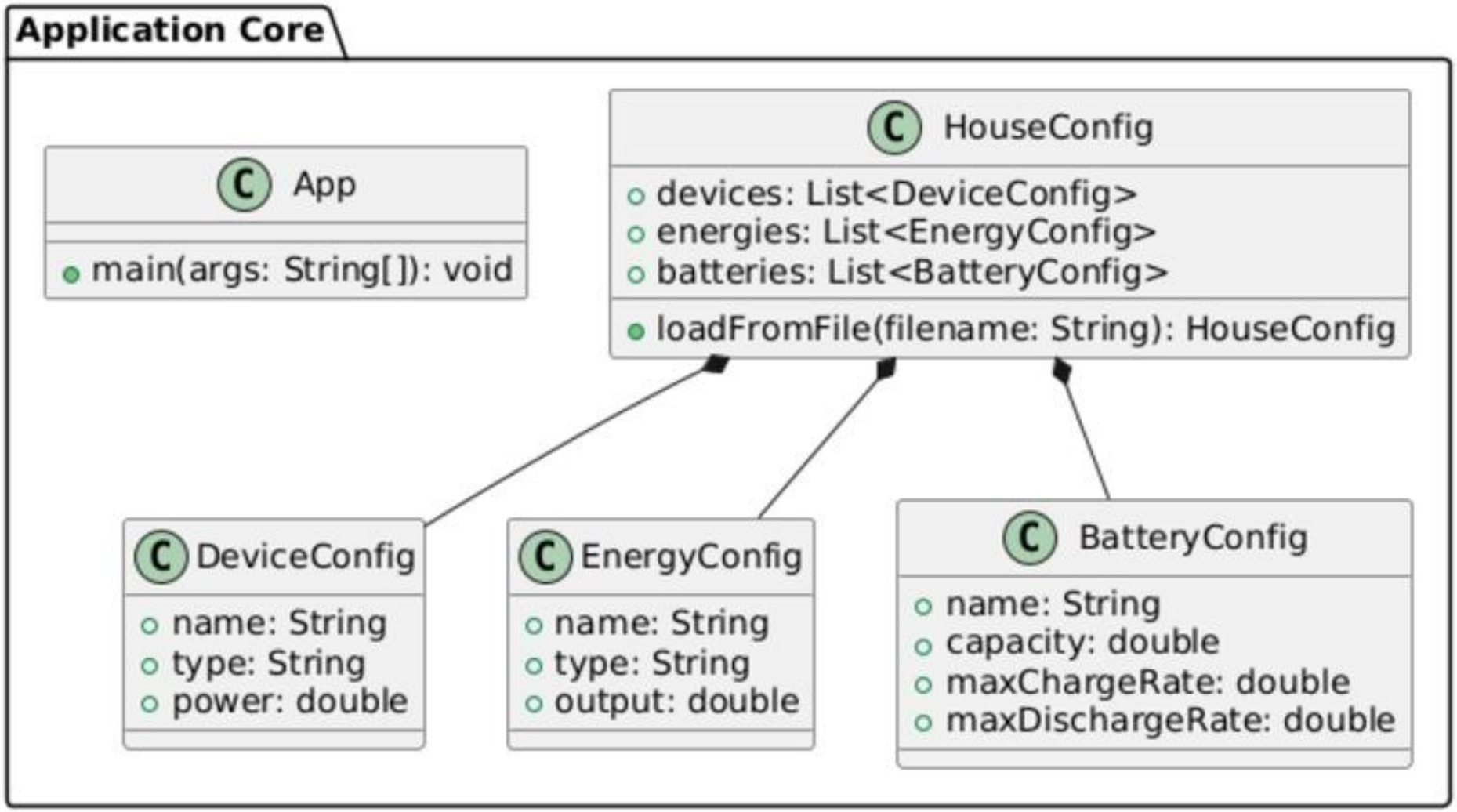


Thank you

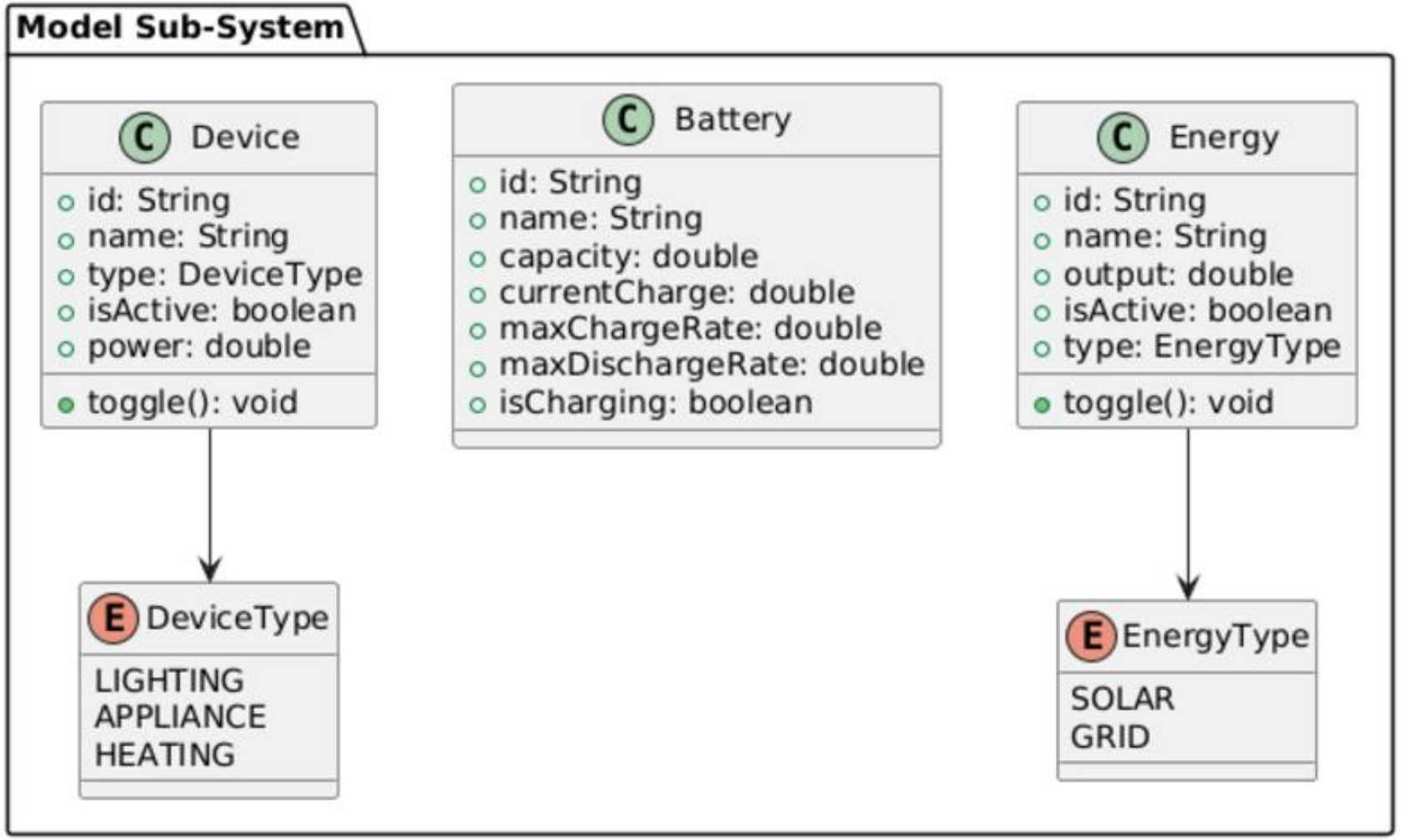
Class Di


HouseConfig
□ devices: List<DeviceConfig>
□ batteries: List<BatteryConfig>
□ energies: List<EnergyConfig>
○ setBatteries(List<BatteryConfig>): void
○ getDevices(): List<DeviceConfig>
○ getEnergies(): List<EnergyConfig>
○ getBatteries(): List<BatteryConfig>
○ setDevices(List<DeviceConfig>): void
○ setEnergies(List<EnergyConfig>): void
○ equals(Object): boolean
◇ canEqual(Object): boolean
○ hashCode(): int
○ toString(): String
○ loadFromFile(String): HouseConfig
LoggerHelper
○ logEnergyEvent(LogManager, String, String): void
○ logDeviceEvent(LogManager, String, String): void
○ logDevicePowerEvent(LogManager, String, String, String): void
○ logEvent(LogManager, Category, String, String, String): void
○ logBatteryEvent(LogManager, String, String): void
○ logChargingEvent(LogManager, String, String, double): void
App
□ log: Logger
○ main(String[]): void
create
MainMenu
□ log: Logger
□ logMenu: LogMenu
□ systemStatusMenu: SystemStatusMenu
□ isRunning: Boolean
□ energyManager: EnergyManager
□ batteryMenu: BatteryMenu
□ deviceMenu: DeviceMenu
□ configMenu: ConfigMenu
□ energyMenu: EnergyMenu
□ processMainMenuChoice(int): void
○ exit(): void
○ show(): void
MenuHelper
□ scanner: Scanner
○ getValidDouble(): double
○ getScanner(): Scanner
○ clearScreen(): void
○ listDevices(List<Device>): void
○ waitForEnter(): void
○ getValidChoice(int, int): int
SystemMonitor
□ instance: SystemMonitor
□ scheduler: ScheduledExecutorService
□ log: Logger
□ deviceManager: DeviceManager
□ energyManager: EnergyManager
□ logManager: LogManager
○ getInstance(): SystemMonitor
○ logSystemData(): void
○ getTotalBatteryCharge(): double
○ startMonitoring(): void
○ monitorSystemStatus(): void
○ shutdown(): void
SystemStatusMenu
□ energyManager: EnergyManager
□ deviceManager: DeviceManager
○ show(): void
○ getTotalBatteryCharge(): double
BatteryMenu
□ energyManager: EnergyManager
□ deviceManager: DeviceManager
○ stopBatterySupply(): void
○ startCharging(): void
○ show(): void
○ stopCharging(): void
○ powerDeviceFromBattery(): void
○ listBatteries(List<Battery>): void
EnergyMenu
□ scanner: Scanner
□ energyManager: EnergyManager
○ toggleEnergy(): void
○ removeEnergy(): void
○ show(): void
○ listEnergy(List<Energy>): void
○ addEnergy(): void
ConfigMenu
□ scanner: Scanner
□ energyManager: EnergyManager
□ DEFAULT_CONFIG_PATH: String
□ deviceManager: DeviceManager
□ logManager: LogManager
○ show(): void
○ loadConfiguration(HouseConfig): void
LogMenu
□ logManager: LogManager
□ scanner: Scanner
○ deleteLog(): void
○ openLog(List<Path>): void
○ searchLogsByName(): void
○ archiveLogs(): void
○ searchLogsByDate(): void
○ show(): void
DeviceMenu
□ scanner: Scanner
□ deviceManager: DeviceManager
○ show(): void
○ removeDevice(): void
○ addDevice(): void
○ toggleDevice(): void
systemStatusMenu
batteryMenu
energyMenu
configMenu
logMenu
deviceMenu
EnergyManager
□ instance: EnergyManager
□ batteries: Map<String, Battery>
□ energies: Map<String, Energy>
□ log: Logger
□ deviceManager: DeviceManager
□ devicePowerTasks: Map<String, CompletableFuture<Void>>
□ executorService: ExecutorService
□ logManager: LogManager
○ getInstance(): EnergyManager
○ startCharging(String): void
○ getEnergy(String): Energy
○ stopPowerDevice(String): void
○ getBattery(String): Battery
○ removeEnergy(String): void
○ stopCharging(String): void
○ getBatteriesByState(boolean): List<Battery>
○ addBattery(Battery): void
□ manageDevicePowerTask(Device, Battery): void
○ toggleEnergy(String): void
○ shutdown(): void
○ addEnergy(Energy): void
○ getAllBatteries(): List<Battery>
□ manageChargingTasks(Battery, List<Energy>): void
□ chargeFromEnergy(Battery, Energy): void
○ clearAllEnergies(): void
○ removeBattery(String): void
○ powerDevice(String, String): void
○ clearAllBatteries(): void
○ getAllEnergies(): List<Energy>
Menu
○ show(): void
batteries
energies
Battery
□ name: String
□ maxChargeRate: double
□ isCharging: boolean
□ capacity: double
□ id: String
□ currentCharge: double
□ maxDischargeRate: double
○ getMaxChargeRate(): double
○ getId(): String
○ hashCode(): int
○ setCurrentCharge(double): void
○ equals(Object): boolean
○ getName(): String
○ getCapacity(): double
○ builder(): BatteryBuilder
○ setMaxChargeRate(double): void
○ getCurrentCharge(): double
○ setMaxDischargeRate(double): void
○ setCharging(boolean): void
◇ canEqual(Object): boolean
○ getMaxDischargeRate(): double
○ isCharging(): boolean
○ setId(String): void
○ setName(String): void
○ setCapacity(double): void
○ toString(): String
Energy
□ type: EnergyType
□ output: double
□ name: String
□ isActive: boolean
□ id: String
○ isActive(): boolean
○ getName(): String
○ getId(): String
○ getOutput(): double
○ builder(): EnergyBuilder
○ getType(): EnergyType
○ setId(String): void
○ setName(String): void
○ setOutput(double): void
○ setActive(boolean): void
○ equals(Object): boolean
○ hashCode(): int
○ setType(EnergyType): void
◇ canEqual(Object): boolean
○ toggle(): void
○ toString(): String
deviceManager
DeviceManager
□ instance: DeviceManager
□ logManager: LogManager
□ devices: Map<String, Device>
□ log: Logger
○ removeDevice(String): void
○ getTotalConsumption(): double
○ getDevicesByState(boolean): List<Device>
○ getAllDevices(): List<Device>
○ clearAllDevices(): void
○ toggleDevice(String): void
○ getDevice(String): Device
○ getInstance(): DeviceManager
○ addDevice(Device): void
devices
Device
□ type: DeviceType
□ isActive: boolean
□ power: double
□ name: String
□ id: String
○ getId(): String
○ getName(): String
○ getType(): DeviceType
○ builder(): DeviceBuilder
○ equals(Object): boolean
○ hashCode(): int
○ isActive(): boolean
○ getPower(): double
○ setId(String): void
○ setName(String): void
○ setType(DeviceType): void
○ setActive(boolean): void
◇ canEqual(Object): boolean
○ setPower(double): void
○ toggle(): void
○ toString(): String
logManager
LogManager
□ instance: LogManager
□ LOG_DIR: Path
□ TIME_FORMAT: DateTimeFormatter
□ ARCHIVE_DIR: Path
□ DATE_FORMAT: DateTimeFormatter
□ log: Logger
○ isLogFileBeforeDate(Path, LocalDate, DateTimeFormatter): boolean
○ getInstance(): LogManager
○ deleteLogs(LocalDate): void
□ initializeDirectories(): void
○ readLogFile(Path): List<String>
○ searchLogs(String): List<Path>
○ deleteLogFile(Path): void
○ logEvent(Category, String, String): void
○ writeToLog(Path, LocalDateTime, String): void
○ clearAllLogs(): void
○ archiveLogFile(Path, ZipOutputStream): void
○ archiveLogs(LocalDate): void

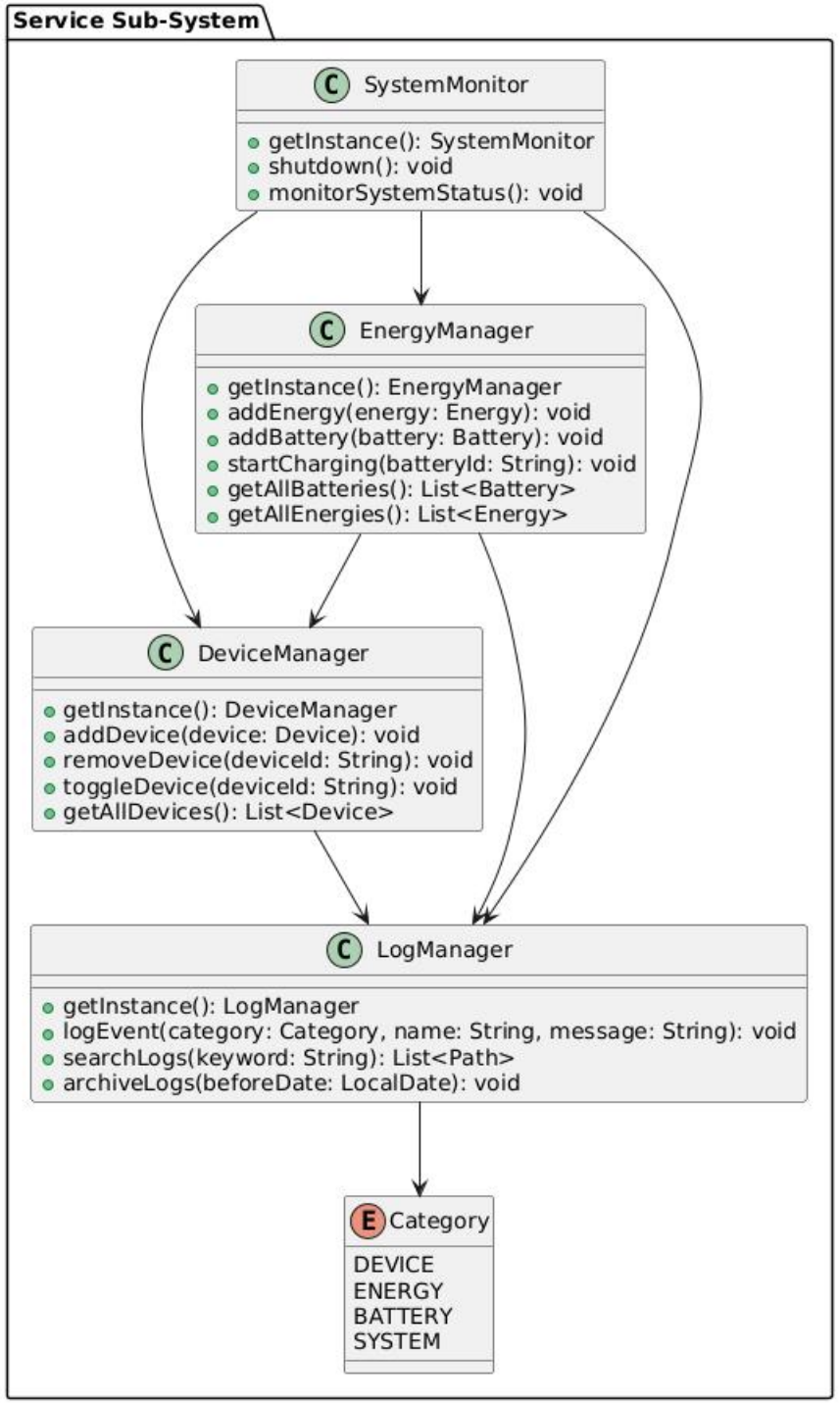
Application Core Sub-System



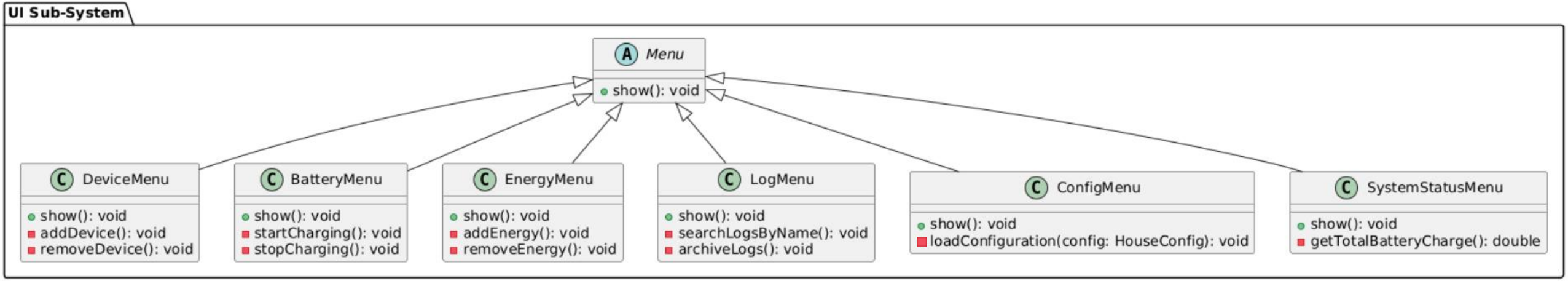
Model Sub-System



Service Sub-System



UI Sub-System



Utility Sub-System