Smart Home Temperature Monitoring System

1. Problem Statement for the Project: Smart Home Temperature Monitoring System Prototype

Develop a **console-based prototype** of a **Smart Home Temperature Monitoring System** using Core Java. This system should simulate temperature readings from multiple "rooms" or "sensors" within a home environment. The central Java application will act as a hub, providing functionalities to view current temperature data, log historical readings, and allow users to set up temperature alerts. The prototype aims to demonstrate the fundamental concepts of data reception, monitoring, and basic alerting in an IoT-like context.

Key Functionality to Include in the Prototype:

- **Simulated Sensor Data Generation:**
  - Generate random temperature values for multiple predefined "rooms" (e.g., Living Room, Bedroom, Kitchen) at regular intervals to simulate real-time sensor data.
  - Allow the simulation to be started and stopped.
- **Data Ingestion and Storage:**
  - The central Java application should "receive" these simulated temperature readings.
  - Store the *current* temperature for each room.
  - Maintain a *simple historical log* of recent temperature readings for each room (e.g., last 10-20 readings or readings from the last hour).
- **Real-time Temperature Display:**
  - Provide a function to display the current temperature of all monitored rooms in a clear, organized format.
- **Temperature Alerting System:**
  - Allow users to define custom *high* and *low* temperature thresholds for individual rooms.
  - Continuously monitor incoming temperature data against these thresholds.
  - Trigger and display a console alert message when a room's temperature goes above its high threshold or below its low threshold.
- **Room/Sensor Management (Basic):**
  - Allow predefined rooms/sensors to be present at startup. For advanced, potentially allow adding/removing rooms during runtime.
- **User Interface:** Design a user-friendly console interface that allows users to navigate menus, view temperatures, set alerts, and manage the simulation.
- **Error Handling:** Implement basic error handling for invalid user inputs (e.g., non-numeric thresholds, non-existent room names).
- **Persistence (In-memory for Prototype):** All current and historical data will be held in memory. (For a next iteration, file persistence could be a goal).

By developing this prototype, users will gain a practical understanding of how an IoT temperature monitoring system could function, from data simulation to real-time display and automated alerting. This will aid in conceptualizing data flow and interaction patterns in basic IoT applications.

Minimum Requirements and System Output:

1. **Sensor Data Simulation (Main Loop):**
   - **Output:** Continuous or interval-based console output showing: Simulating [Room Name] temperature: [XX.X]°C (e.g., Simulating Living Room temperature: 24.5°C).
   - **Interaction:** Option to start/stop the simulation loop from the main menu.
2. **Current Temperature Display:**
   - **Form Creation:** Select option to view current temperatures.
   - **Output:** A formatted list displaying:

```
--- Current Room Temperatures ---
Room Name       | Temperature (°C) | Status
------------------------------------------
Living Room     | 23.8             | Normal
Bedroom 1       | 25.1             | High (Alert!)
Kitchen         | 21.5             | Normal
------------------------------------------
```

3. **Alert Thresholds Setting:**
   - **Form Creation:** Console prompts for: Enter Room Name to set alert:, Enter High Temperature Threshold:, Enter Low Temperature Threshold:.
   - **Output:** Confirmation message: Alert for [Room Name] set: Min [XX.X]°C, Max [YY.Y]°C. or "Error: Room not found."
4. **Alert Triggering:**
   - **Output:** When a threshold is breached, an immediate console alert appears: !!! ALERT !!! [Timestamp] - [Room Name] temperature ([XX.X]°C) is [ABOVE/BELOW] threshold!
5. **Historical Data View (Basic Log):**
   - **Form Creation:** Console prompt for Enter Room Name to view history:.
   - **Output:** A list of recent temperature readings for the selected room:

```
--- Temperature History for [Room Name] ---
[YYYY-MM-DD HH:MM:SS] - 22.1°C
[YYYY-MM-DD HH:MM:SS] - 22.5°C
[YYYY-MM-DD HH:MM:SS] - 23.0°C
... (up to last N readings)
------------------------------------------
```

Weekly Progress Report (Smart Home Temperature Monitoring System)

Name: Manikandan
Domain IoT
Date of submission: July 5,2025
Week Ending: 05

1. Format:

This report is prepared in a clear and organized manner, utilizing headings, subheadings, and bullet points for enhanced readability, mimicking a professional document format.

2. Content: Project Overview and Progress

This week, my work focused on initiating the development of the **Smart Home Temperature**

Monitoring System Prototype. I concentrated on setting up the basic framework for simulating sensor data, ingesting it, and displaying current readings through a console interface. This forms the foundation for future alerting and historical data features.

Tasks Completed:

- **Designed and implemented RoomSensor class:** This class simulates individual temperature sensors, responsible for generating random temperature readings within a defined range.
- **Designed and implemented TemperatureMonitor class:** This central class manages multiple RoomSensor instances, "receives" their simulated data, and stores the current readings.
- **Established the main SmartHomeApp class structure:** Set up the console-based main menu and the primary loop for user interaction.
- **Implemented Current Temperature Display:** Developed the logic to retrieve and present the latest temperature readings for all simulated rooms.
- **Initial setup for Sensor Data Simulation Loop:** Created the basic mechanism to continuously generate and update sensor data in the background (or upon request).

Milestones Achieved:

- **Core data models for rooms/sensors (RoomSensor) are defined.**
- **Central monitoring hub (TemperatureMonitor) is functional for data ingestion.**
- **Console menu navigation is established for core features.**
- **Real-time (simulated) temperature display for all rooms is working.**

Significant Contributions:

- Laid down the fundamental architecture for the IoT prototype, simulating data flow from sensors to a central application.
- Successfully implemented the "real-time" aspect of data monitoring by updating and displaying current temperatures.

Specific Details and Examples of Progress:

- **Sensor Simulation Example:** The RoomSensor object generates a random float temperature each time its readTemperature() method is called, e.g., 23.5, 24.1, 22.9.
- **Current Temperature Display Output:**

```
--- Current Room Temperatures ---
Room Name       | Temperature (°C) | Status
------------------------------------------
Living Room     | 23.8             | Normal
Bedroom         | 25.0             | Normal
Kitchen         | 21.7             | Normal
------------------------------------------
```

- **Data Ingestion:** The TemperatureMonitor stores current temperatures in a Map<String, Double> (e.g., {"Living Room": 23.8, "Bedroom": 25.0}).

3. Challenges and Hurdles:

- **Simulating Continuous Data Flow in a Console App:** The challenge was to make the console application *feel* like it's receiving continuous data without blocking the main user interaction thread.
  - **Approach:** Initially, I considered a simple loop that constantly updates, but this makes the menu unresponsive. I explored using a Thread for the simulation, but for a basic console prototype, a simpler approach of generating new data *each time* the "view current temperatures" option is selected or using a timed update on a

    simple loop was chosen to keep it within "Core Java prototype" bounds without introducing full concurrency complexity just yet.
    - **Solution:** For now, the simulation triggers a batch of new readings when selected, or readings are updated as part of the loop for the "real-time display" feature. (Future: Introduce Timer or ScheduledExecutorService for background updates if required.)
- **Managing Multiple Room Data:** Keeping track of temperatures and potentially settings for multiple distinct rooms efficiently.
    - **Approach:** Utilized Map<String, RoomSensor> and Map<String, Double> within the TemperatureMonitor to map room names to their respective sensor objects and current temperature values.
    - **Solution:** This Map-based approach allowed for easy lookup and management of individual room data, scaling well for a prototype with a few rooms.

4. Lessons Learned:

- **Abstraction in Simulation:** Effectively abstracting the "sensor" as a RoomSensor class, separate from the TemperatureMonitor, simplifies the design and makes it easier to extend if different sensor types were to be simulated later.
- **User Interface vs. Background Processes:** Even in a console application, balancing user interaction with background data updates (simulated in this case) requires careful thought. For simpler prototypes, a "pull" model (user requests update) is often sufficient before moving to "push" (background updates).
- **Data Structure Choice Matters:** Using HashMap for storing room data by name proved very efficient for quick lookups and updates, highlighting the importance of selecting appropriate data structures for the task.
- **Problem-solving technique:** When faced with simulating real-time systems, start with the simplest possible simulation (e.g., random numbers) and gradually introduce complexity (e.g., more realistic ranges, timed updates) as the core framework is stable.
- **Valuable experience for future endeavors:** This project provides practical experience in designing systems that process continuous data streams (even simulated ones), which is a fundamental concept in IoT and data processing applications. It also hones skills in structuring code for clear separation of concerns (simulation vs. monitoring vs. UI).

Smart Home Temperature Monitoring System Prototype

1. Problem Statement for the Project:
Develop a **console-based prototype** of a **Smart Home Temperature Monitoring System** using Core Java. This system should simulate temperature readings from multiple "rooms" or "sensors" within a home environment. The central Java application will act as a hub, providing functionalities to view current temperature data, log historical readings, and allow users to set up temperature alerts. The prototype aims to demonstrate the fundamental concepts of data reception, monitoring, and basic alerting in an IoT-like context.
Key Functionality to Include in the Prototype (Re-stated for context):

- **Simulated Sensor Data Generation:** Generate random temperature values for multiple predefined "rooms" (e.g., Living Room, Bedroom, Kitchen) at regular intervals to simulate real-time sensor data. Allow the simulation to be started and stopped.
- **Data Ingestion and Storage:** The central Java application should "receive" these simulated temperature readings. Store the *current* temperature for each room. Maintain a *simple historical log* of recent temperature readings for each room (e.g., last 10-20 readings or readings from the last hour).
- **Real-time Temperature Display:** Provide a function to display the current temperature of all monitored rooms in a clear, organized format.
- **Temperature Alerting System:** Allow users to define custom *high* and *low* temperature thresholds for individual rooms. Continuously monitor incoming temperature data against these thresholds. Trigger and display a console alert message when a room's temperature goes above its high threshold or below its low threshold.
- **Room/Sensor Management (Basic):** Allow predefined rooms/sensors to be present at startup.
- **User Interface:** Design a user-friendly console interface that allows users to navigate menus, view temperatures, set alerts, and manage the simulation.
- **Error Handling:** Implement basic error handling for invalid user inputs (e.g., non-numeric thresholds, non-existent room names).
- **Persistence (In-memory for Prototype):** All current and historical data will be held in memory.

Java Program Code:
This code implements the RoomSensor to simulate temperature, the TemperatureMonitor to manage rooms and readings, and the SmartHomeApp for the console UI and main loop.

```
import java.util.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

class RoomSensor {
    private String roomName;
    private double currentTemperature;
    private Random random;
    private List<String> history;
    private int historyCapacity = 10;


    private Double highThreshold;
    private Double lowThreshold;

    public RoomSensor(String roomName, double initialTemperature) {
        this.roomName = roomName;
        this.currentTemperature = initialTemperature;
        this.random = new Random();
        this.history = new LinkedList<>();
        addReadingToHistory(initialTemperature);

    public String getRoomName() {
        return roomName;
    }

    public double getCurrentTemperature() {
        return currentTemperature;
    }

    public Double getHighThreshold() {
        return highThreshold;
    }

    public Double getLowThreshold() {
        return lowThreshold;
    }

    public List<String> getHistory() {
        return new ArrayList<>(history);  external modification
    }


    public double readTemperature() {

        double fluctuation = (random.nextDouble() * 2.0) - 1.0;
        currentTemperature += fluctuation;


        if (currentTemperature < 18.0) currentTemperature = 18.0 +
random.nextDouble();
        if (currentTemperature > 30.0) currentTemperature = 30.0 -
random.nextDouble();

        addReadingToHistory(currentTemperature);
        return currentTemperature;
    }

    l
    private void addReadingToHistory(double temperature) {
        DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String formattedReading = String.format("%s - %.1f°C",
dtf.format(LocalDateTime.now()), temperature);
        history.add(formattedReading);
        if (history.size() > historyCapacity) {
            ((LinkedList<String>) history).removeFirst();
        }
    }


    public void setAlertThresholds(double low, double high) {
        if (low >= high) {
            System.err.println("Warning: Low threshold must be less
than high threshold for " + roomName + ". Settings not applied.");
            return;
        }
        this.lowThreshold = low;
        this.highThreshold = high;
        System.out.println("Alert thresholds set for " + roomName + ":
Low=" + String.format("%.1f", low) + "°C, High=" +
String.format("%.1f", high) + "°C");
    }


    public String checkAlert() {
        if (highThreshold != null && currentTemperature >
highThreshold) {
            return "ABOVE";
        }
        if (lowThreshold != null && currentTemperature < lowThreshold)
{
            return "BELOW";
        }
        return "NORMAL";
    }
}


class TemperatureMonitor {
    private Map<String, RoomSensor> roomSensors;
    private ScheduledExecutorService scheduler;
    public TemperatureMonitor() {
        this.roomSensors = new HashMap<>();
        addRoom("Living Room", 22.5);
        addRoom("Bedroom", 20.0);
        addRoom("Kitchen", 24.0);
        addRoom("Study", 21.0);
    }

    public void addRoom(String roomName, double initialTemp) {
        if (!roomSensors.containsKey(roomName)) {
            roomSensors.put(roomName, new RoomSensor(roomName,
initialTemp));
        } else {
            System.out.println("Room '" + roomName + "' already
exists.");
        }
    }

    public RoomSensor getRoomSensor(String roomName) {
        return roomSensors.get(roomName);
    }

    public Collection<RoomSensor> getAllSensors() {
        return roomSensors.values();
    }2q
    public void startSimulation() {
        if (scheduler == null || scheduler.isShutdown()) {
            scheduler = Executors.newSingleThreadScheduledExecutor();
            scheduler.scheduleAtFixedRate(() -> {
                System.out.println("\n--- Simulating new readings
---");
                for (RoomSensor sensor : roomSensors.values()) {
                    double newTemp = sensor.readTemperature();
                    System.out.println("Simulating " +
sensor.getRoomName() + " temperature: " + String.format("%.1f",
newTemp) + "°C");
                    String alertStatus = sensor.checkAlert();
                    if (!"NORMAL".equals(alertStatus)) {
                        System.out.printf("!!! ALERT !!! %s - %s
temperature (%.1f°C) is %s threshold!%n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")),
                                sensor.getRoomName(), newTemp,
alertStatus);
                    }
                }
                System.out.println("------------------------------");
            }, 0, 5, TimeUnit.SECONDS);
            System.out.println("Temperature simulation started. New
readings every 5 seconds.");
        } else {
            System.out.println("Simulation is already running.");
        }
    }

    public void stopSimulation() {
        if (scheduler != null && !scheduler.isShutdown()) {
            scheduler.shutdown();
            try {
                if (!scheduler.awaitTermination(5, TimeUnit.SECONDS))
{
                    scheduler.shutdownNow();
                }
            } catch (InterruptedException e) {
                scheduler.shutdownNow();
            }
            System.out.println("Temperature simulation stopped.");
        } else {
            System.out.println("Simulation is not running.");
        }
    }
}
public class SmartHomeApp {
    private TemperatureMonitor monitor;
    private Scanner scanner;

    public SmartHomeApp() {
        this.monitor = new TemperatureMonitor();
        this.scanner = new Scanner(System.in);
    }

    public static void main(String[] args) {
        SmartHomeApp app = new SmartHomeApp();
        app.run();
    }

    public void run() {
        System.out.println("Welcome to the Smart Home Temperature
Monitoring System Prototype");
        boolean exit = false;

        while (!exit) {
            System.out.println("\n--- Main Menu ---");
            System.out.println("1. Start Temperature Simulation");
            System.out.println("2. Stop Temperature Simulation");
            System.out.println("3. View Current Temperatures");
            System.out.println("4. Set Room Alert Thresholds");
            System.out.println("5. View Room Temperature History");
            System.out.println("6. Exit Application");
            System.out.print("Please choose an option: ");
            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    monitor.startSimulation();
                    break;
                case "2":
                    monitor.stopSimulation();
                    break;
                case "3":
                    viewCurrentTemperatures();
                    break;
                case "4":
                    setRoomAlertThresholds();
                    break;
                case "5":
                    viewRoomTemperatureHistory();
                    break;
                case "6":
                    exit = true;
                    monitor.stopSimulation();
                    System.out.println("Thank you for using the Smart
Home Temperature Monitoring System. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid option. Please try
again.");
            }
        }
        scanner.close();
    }

    private void viewCurrentTemperatures() {
        System.out.println("\n--- Current Room Temperatures ---");
        if (monitor.getAllSensors().isEmpty()) {
            System.out.println("No rooms are currently being
monitored.");
            return;
        }

        System.out.printf("%-15s | %-15s | %-20s%n", "Room Name",
"Temperature (°C)", "Status (Alerts)");

System.out.println("-----------------------------------------------------
");
        for (RoomSensor sensor : monitor.getAllSensors()) {
            String alertStatus = sensor.checkAlert();
            String statusDisplay = "Normal";
            if (!"NORMAL".equals(alertStatus)) {
                statusDisplay = alertStatus + " (Alert!)";
            }

            System.out.printf("%-15s | %-15.1f | %-20s%n",
                    sensor.getRoomName(),
sensor.getCurrentTemperature(), statusDisplay);
        }

System.out.println("-----------------------------------------------------
");
    }

    private void setRoomAlertThresholds() {
        System.out.println("\n--- Set Room Alert Thresholds ---");
        System.out.print("Enter Room Name to set alert for: ");
        String roomName = scanner.nextLine().trim();

        RoomSensor sensor = monitor.getRoomSensor(roomName);
        if (sensor == null) {
            System.out.println("Error: Room '" + roomName + "' not
found. Please check the name.");
            return;
        }

        double lowThreshold = 0;
        double highThreshold = 0;
        boolean validInput = false;

        while (!validInput) {
            System.out.printf("Current thresholds for %s: Low=%.1f,
High=%.1f%n",
                    roomName, sensor.getLowThreshold() != null ?
sensor.getLowThreshold() : 0.0,
                    sensor.getHighThreshold() != null ?
sensor.getHighThreshold() : 0.0);
            System.out.print("Enter Low Temperature Threshold (°C):
");
            try {
                lowThreshold = Double.parseDouble(scanner.nextLine());
                System.out.print("Enter High Temperature Threshold
(°C): ");
                highThreshold =
Double.parseDouble(scanner.nextLine());
```

```
                if (lowThreshold >= highThreshold) {
                    System.out.println("Error: Low threshold must be
less than high threshold. Please re-enter.");
                } else {
                    validInput = true;
                }
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a
numeric value for temperature thresholds.");
            }
        }

        sensor.setAlertThresholds(lowThreshold, highThreshold);
        System.out.println("Alert thresholds for '" + roomName + "'
updated successfully.");
    }

    private void viewRoomTemperatureHistory() {
        System.out.println("\n--- View Room Temperature History ---");
        System.out.print("Enter Room Name to view history for: ");
        String roomName = scanner.nextLine().trim();

        RoomSensor sensor = monitor.getRoomSensor(roomName);
        if (sensor == null) {
            System.out.println("Error: Room '" + roomName + "' not
found. Please check the name.");
            return;
        }

        System.out.println("\n--- Temperature History for " + roomName
+ " ---");
        List<String> history = sensor.getHistory();
        if (history.isEmpty()) {
            System.out.println("No history recorded for " + roomName +
" yet.");
        } else {
            // Display in reverse order (most recent first)
            for (int i = history.size() - 1; i >= 0; i--) {
                System.out.println(history.get(i));
            }
        }

System.out.println("-------------------------------------------");
    }
}
```

Console Output:

```
Welcome to the Smart Home Temperature Monitoring System Prototype

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 1
Temperature simulation started. New readings every 5 seconds.

--- Simulating new readings ---
Simulating Living Room temperature: 22.0°C
Simulating Bedroom temperature: 20.3°C
Simulating Kitchen temperature: 24.9°C
Simulating Study temperature: 20.7°C
----------------------------

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 3

--- Current Room Temperatures ---
Room Name       | Temperature (°C) | Status (Alerts)
--------------------------------------------------
Living Room     | 22.0             | Normal
Bedroom         | 20.3             | Normal
Kitchen         | 24.9             | Normal
Study           | 20.7             | Normal
--------------------------------------------------

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 4
```

```
--- Set Room Alert Thresholds ---
Enter Room Name to set alert for: Bedroom
Current thresholds for Bedroom: Low=0.0, High=0.0
Enter Low Temperature Threshold (°C): 19.0
Enter High Temperature Threshold (°C): 21.0
Alert thresholds set for Bedroom: Low=19.0°C, High=21.0°C
Alert thresholds for 'Bedroom' updated successfully.

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 1
Simulation is already running.

--- Simulating new readings ---
Simulating Living Room temperature: 22.9°C
Simulating Bedroom temperature: 21.6°C
!!! ALERT !!! 10:36:05 - Bedroom temperature (21.6°C) is ABOVE
threshold!
Simulating Kitchen temperature: 24.2°C
Simulating Study temperature: 20.5°C
----------------------------

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 3

--- Current Room Temperatures ---
Room Name         | Temperature (°C) | Status (Alerts)
--------------------------------------------------
Living Room       | 22.9             | Normal
Bedroom           | 21.6             | ABOVE (Alert!)
Kitchen           | 24.2             | Normal
Study             | 20.5             | Normal
--------------------------------------------------

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
```

```
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 5

--- View Room Temperature History ---
Enter Room Name to view history for: Bedroom

--- Temperature History for Bedroom ---
2025-07-02 10:36:05 - 21.6°C
2025-07-02 10:36:00 - 20.3°C
2025-07-02 10:35:55 - 20.0°C
------------------------------------------

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 2
Temperature simulation stopped.

--- Main Menu ---
1. Start Temperature Simulation
2. Stop Temperature Simulation
3. View Current Temperatures
4. Set Room Alert Thresholds
5. View Room Temperature History
6. Exit Application
Please choose an option: 6
Thank you for using the Smart Home Temperature Monitoring System.
Goodbye!
```