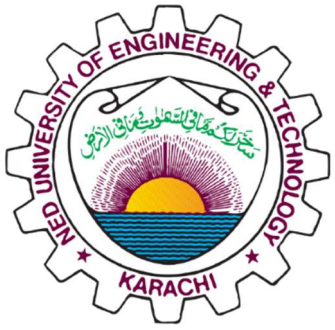
FORMAL METHODS IN SE

(SE-313)

ASSIGNMENT NO: 1



Submitted By:

Sara Hameed (SE-21012)

Syeda Umm E Abiha Rizvi (SE-21014)

Section: A

Batch: 2021

Department: Software Engineering

Submitted To: Dr. Mustafa Latif

Software Engineering Department

NED University of Engineering and Technology, Karachi.

VEND TEMP CONTROLLER

Vending Machine Temperature Controller System for Pharmacy

1. INTRODUCTION:

This document outlines the formal specification of the Pharmacy Vending Machine Temperature Controller using the Vienna Development Method Specification Language (VDM-SL). The primary objective of this system is to maintain the temperature within the critical range of 2 to 8 degrees Celsius, ensuring optimal storage conditions for pharmaceutical items in a vending machine.

2. SCOPE:

The scope of the Pharmacy Vending Machine Temperature Controller encompasses the development, implementation, and operation of a robust temperature control system for vending machines specifically designed for pharmaceutical storage. The primary focus is on maintaining the temperature within the critical range of 2 to 8 degrees Celsius to ensure the integrity and stability of pharmaceutical products. The scope includes the following key elements:

2.1. Temperature Control:

The system will actively control the internal temperature of the pharmacy vending machine, utilizing precision mechanisms to maintain it within the specified range of 2 to 8 degrees Celsius.

2.2. Temperature Monitoring:

Continuous monitoring of the vending machine's internal temperature will be performed in real-time. The system will provide accurate and up-to-date temperature data to operators and relevant personnel.

2.3. Alarm Mechanism:

An alarm system will be implemented to promptly detect temperature deviations beyond the critical range. If the temperature falls below 2 degrees Celsius or exceeds 8 degrees Celsius, the system will trigger an immediate alarm. The alarm will serve as an alert to operators or maintenance personnel, indicating the need for timely intervention to prevent potential damage to pharmaceutical items.

3. SYSTEM ARCHITECTURE:

The Pharmacy Vending Machine Temperature Controller system comprises the following components:

3.1. Temperature Sensor:

Responsible for accurate measurement of the current temperature within the vending machine, providing real-time data for control and monitoring.

3.2. Cooling System:

Controls the refrigeration unit to adjust the temperature as needed, ensuring the pharmaceutical products remain within the specified temperature range.

3.3. Alarm Module:

Activates an immediate alarm response in case of temperature deviations beyond the critical range, facilitating rapid intervention to safeguard the integrity of pharmaceutical items.

4. 4+1 VIEW POINTS:

4.1. LOGICAL VIEW POINT (CLASS DIAGRAM):

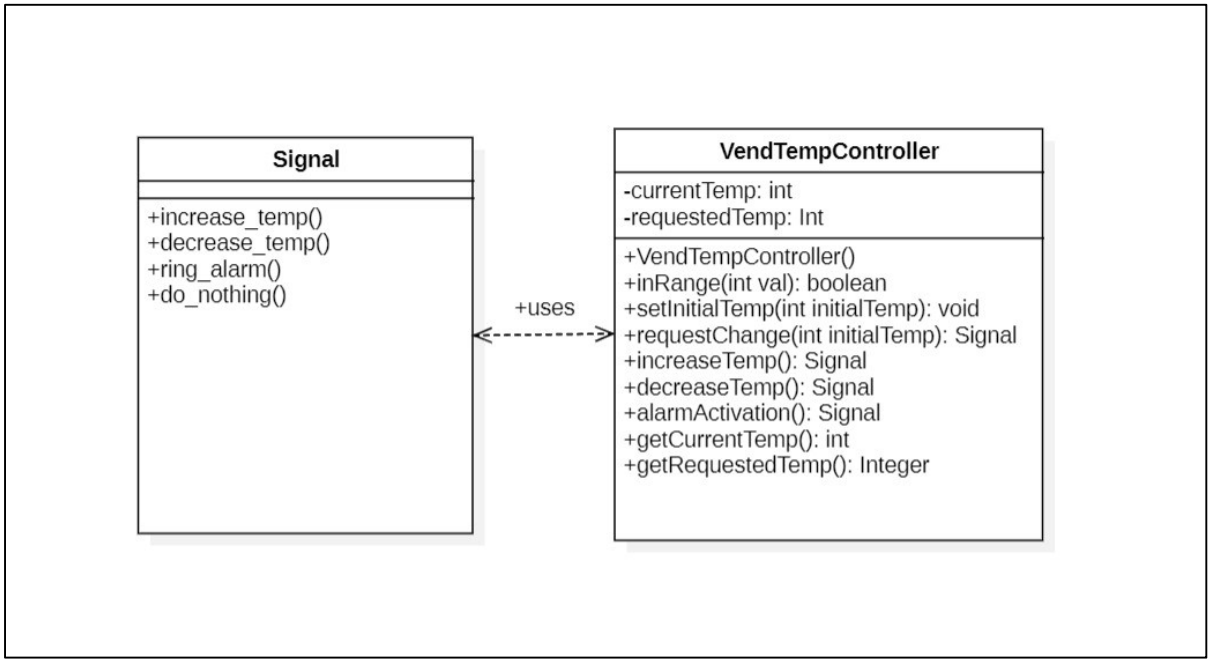


Figure 1 Class Diagram of VendTempController

EXPLANATION:

The VendTempController class encapsulates the logic for managing temperature in a vending machine. It uses the Signal enum to communicate different states or actions. The class provides methods for setting initial temperature, requesting temperature changes, increasing or decreasing temperature, activating alarms, and retrieving current/requested temperature.

1. SIGNAL CLASS:

The Signal class represents different signals that can be generated by the `VendTempController`. It is an enumeration with the following constants:

- **INCREASE_TEMP`**
- **DECREASE_TEMP`**
- **RING_ALARM`**
- **DO_NOTHING**

2. VENDTEMPCONTROLLER CLASS:

The VendTempController class represents a controller for vending machine temperature. It has the following attributes and operation:

- **private int currentTemp:** Represents the current temperature of the vending machine.
- **private Integer requestedTemp:** Represents the requested temperature for the vending machine.
- **VendTempController():** Constructor method to initialize the currentTemp to 2 and requestedTemp to null.
- **inRange(int val): boolean:** Checks if the given value is within the valid temperature range (2 to 8).
- **setInitialTemp(int initialTemp):** void: Sets the initial temperature if it is within the valid range and the current temperature is 2.
- **requestChange(int initialTemp):** Signal: Requests a change in temperature based on the provided initial temperature. Returns a Signal indicating whether to increase, decrease, or do nothing.
- **increaseTemp():** Signal: Increases the temperature by 0.5 units if conditions are met. Returns a Signal indicating whether the increase was successful or if no action is needed.
- **decreaseTemp(): Signal:** Decreases the temperature by 0.5 units if conditions are met. Returns a Signal indicating whether the decrease was successful or if no action is needed.
- **alarmActivation(): Signal:** Checks if an alarm needs to be activated based on temperature conditions. Returns a Signal indicating whether to ring the alarm or do nothing.
- **getCurrentTemp(): int:** Gets the current temperature.
- **getRequestedTemp(): int:** Gets the requested temperature.

4.2. PROCESS VIEW POINT (SEQUENCE DIAGRAM):

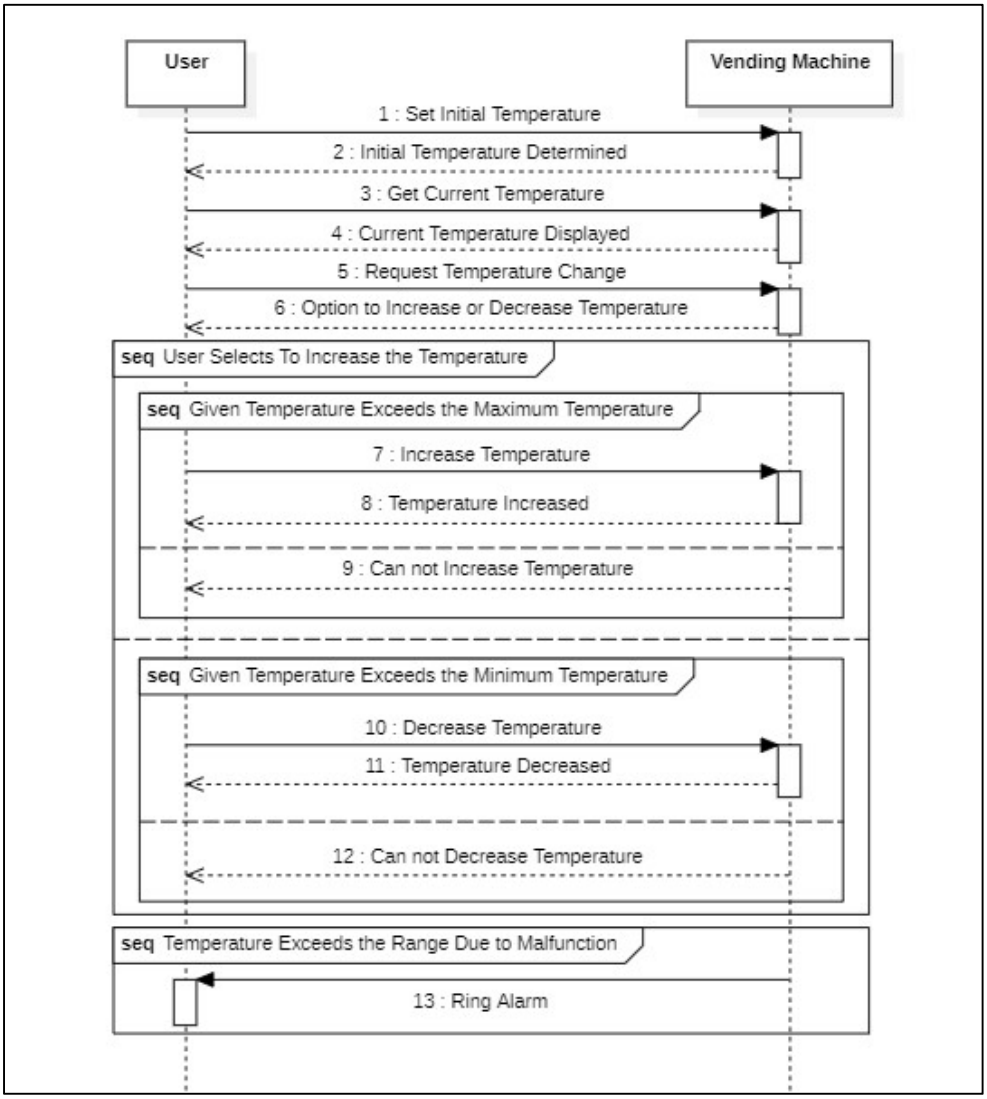


Figure 2 Sequence Diagram of VendTempController

EXPLANATION:

The sequence diagram depicts user interactions with a vending machine's temperature control system. The vending machine's temperature control system features user interactions for setting the initial temperature, obtaining the current temperature, and requesting temperature changes. The system ensures that requested changes, whether increases or decreases, fall within the acceptable temperature range of 2 to 8. If a user attempts to set a temperature outside this range, the system responds with a **DO_NOTHING** signal, preventing any change. Moreover, the system automatically triggers a **RING_ALARM** signal if the current temperature goes beyond the specified range due to a malfunction or any unforeseen circumstances. This proactive alarm mechanism adds an extra layer of safety and ensures that potential issues are promptly addressed, contributing to the overall reliability of the temperature control system.

4.3. PHYSICAL VIEW POINT (DEPLOYMENT DIAGRAM):

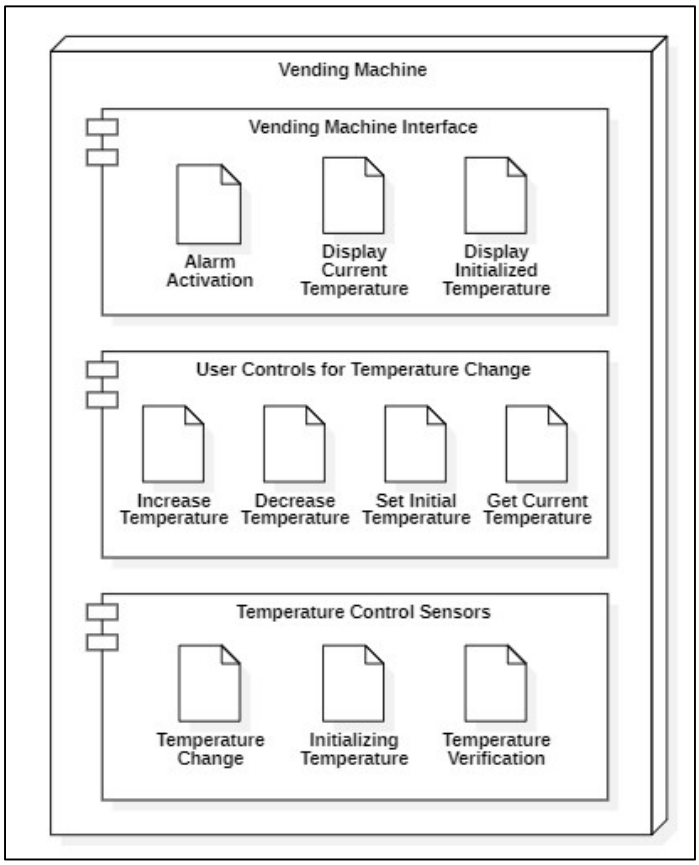


Figure 3 Deployment Diagram of VendTempController

EXPLANATION:

The Deployment Diagram illustrates the physical deployment of components in a vending machine temperature control system. It consists of three main components:

1. Vending Machine Interface:
This component handles the interaction between the user and the vending machine's temperature control system. Functionalities include activating the alarm, displaying the current temperature, and showing the initialized temperature.

2. User Controls for Temperature Change:
This component encompasses artifacts representing user-controlled actions for temperature adjustments. Artifacts include Increase Temp, Decrease Temp, Set Initial Temp, and Get Current Temp, reflecting the user's ability to control and monitor the temperature.

3. Temperature Control Sensors:
This component involves artifacts related to the sensors responsible for monitoring and regulating the temperature. Artifacts include functionalities for temperature change, initializing temperature, and verifying temperature constraints.

4.4. DEVELOPMENT VIEW POINT (COMPONENT DIAGRAM):

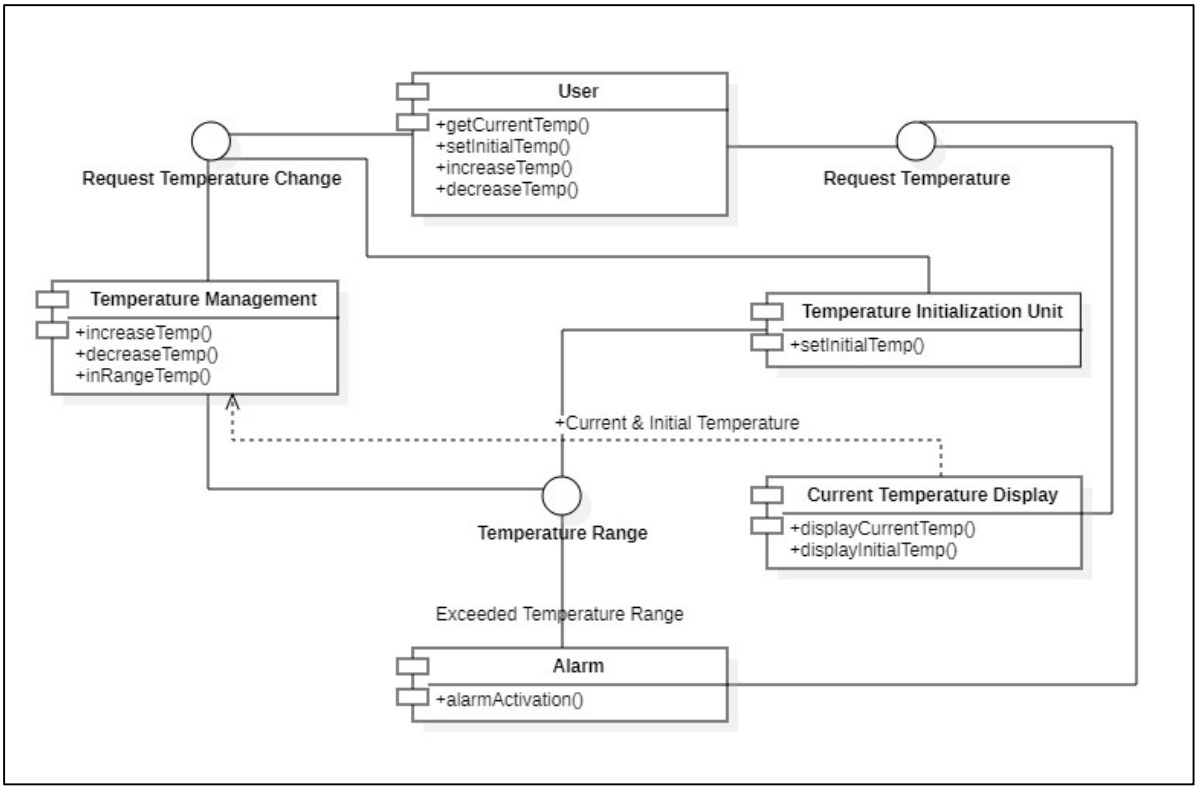


Figure 4 Component Diagram of VendTempController

EXPLANATION:

The component diagram illustrates the system's modular structure, highlighting key components and their interactions. The TempManagement component centralizes temperature control, while the User component provides interfaces for user interactions. The TempInitializationUnit sets the initial temperature, and the CurrentTempDisplay manages temperature displays. The Alarm component handles alarm activation. This modular design enhances system modifiability and maintainability, allowing for easier updates to specific functionalities.

EXPLANATION:

4.5. +1 VIEW POINT OR SCENARIO (USE CASE DIAGRAM):

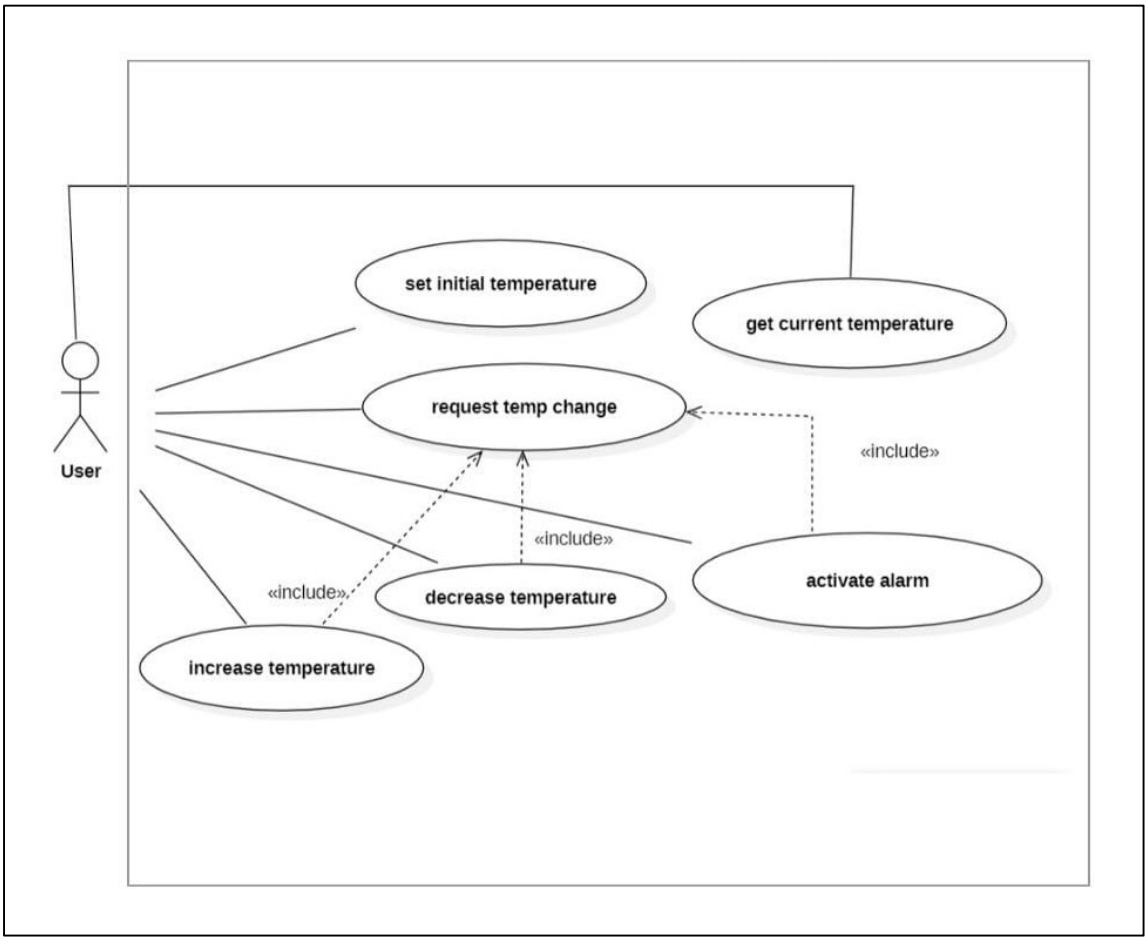


Figure 5 Use Case Diagram of VendTempController

EXPLANATION:

The Use Case Diagram depicts user interactions with the vending machine's temperature control system. Users have three primary actions: setting the initial temperature, obtaining the current temperature, and requesting temperature changes. Within the "Request Temperature Change" use case, users can further choose to increase or decrease the temperature based on their preferences. Additionally, there is an "Alarm Activation" scenario designed to alert users in case of malfunctions or when the temperature surpasses predefined limits.

5. VDM SPECIFICATION:

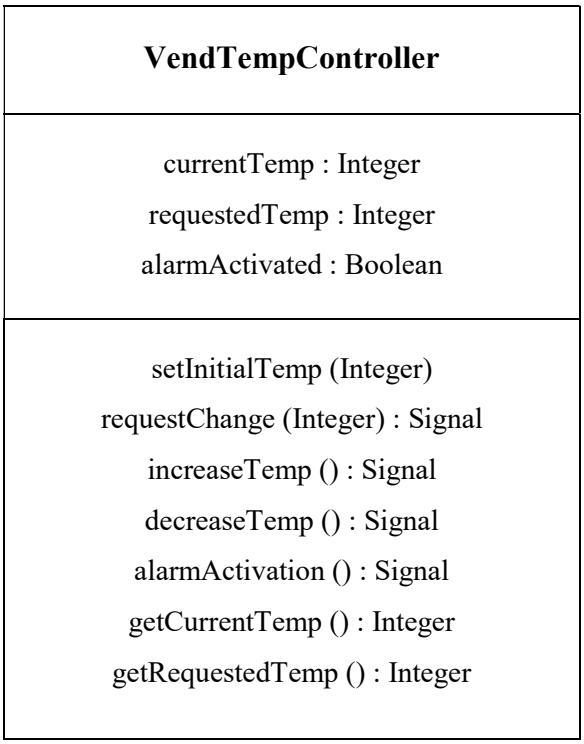


Figure 6 Specification of the VendTempController

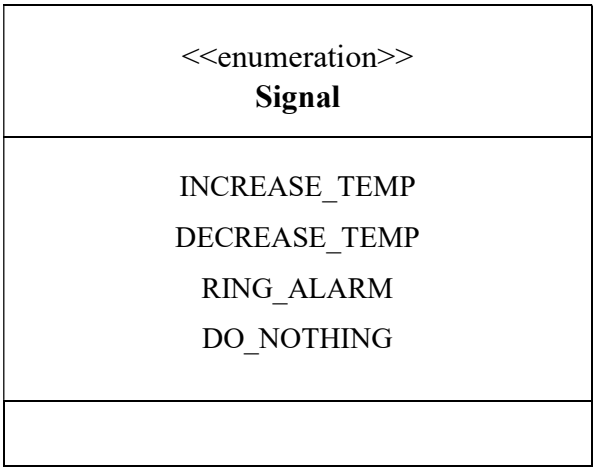


Figure 2 UML specification of the Signal type

types
Signal = <INCREASE_TEMP> | <DECREASE_TEMP> | <RING_ALARM> | <DO_NOTHING>

values
MIN_TEMP: ℤ = 2
MAX_TEMP: ℤ = 8

state VendTempController **of**
currentTemp : [ℤ]
requestedTemp : [ℤ]
alarmActivated : [*B*]

-- both requested and actual temperatures must be in range
inv mk-VendTempController (r, c) ≜ (inRange (r) ∨ r = **nil**) ∧ (inRange (c) ∨ c = **nil**)

-- both requested and current temperatures are undefined, and alarm activation is set to
-- false when the system is initialized
init mk-VendTempController (r, a, alarm) ≜ r = **nil** ∧ c = **nil** ∧ alarm = **FALSE**
end

functions
-- a function that verifies that both requested and current temperatures are in the defined
-- temperature range
inRange(val : ℤ) range: *B*
pre TRUE
post range ⇔ MIN_TEMP ≤ val ≥ MAX_TEMP

operations
-- an operation that records the intitial temperature of the system
setInitialTemp (initialTemp : ℤ)
ext wr currentTemp: [ℤ]
pre inRange(tempIn) ∧ currentTemp = **nil**
post currentTemp = initialTemp

-- an operation that records the requested temperature and signals the hardware to increase or
-- decrease the temperature as appropriate
requestChange (initialTemp : ℤ) signalOut : Signal
ext wr requestedTemp : [ℤ]
rd currentTemp : [ℤ]
pre inRange(initialTemp) ∧ currentTemp ≠ **nil**
post requestedTemp = initialTemp ∧
(initialTemp > currentTemp ∧ signalOut = <INCREASE_TEMP> ∨
initialTemp < currentTemp ∧ signalOut = <DECREASE_TEMP> ∨
initialTemp = currentTemp ∧ signalOut = <DO_NOTHING>)

-- an operation that records 0.5 degree increase and instructs the hardware either to continue increasing the temperature or to stop
increaseTemp ( ) signalOut : Signal
ext wr currentTemp : [ℤ]
rd requestedTemp : [ℤ]
pre currentTemp < requestedTemp ∧ currentTemp ≠ **nil** ∧ requestedTemp ≠ **nil**
post currentTemp = $\overline{\text{currentTemp}}$ + 0.5 ∧
(currentTemp < requestedTemp ∧ signalOut = <INCREASE_TEMP> ∨
currentTemp = requestedTemp ∧ signalOut = <DO_NOTHING>)

-- an operation that records 0.5 degree decrease and instructs the hardware either to continue decreasing the temperature or to stop
decreaseTemp ( ) signalOut : Signal
ext wr currentTemp : [ℤ]
rd requestedTemp : [ℤ]
pre currentTemp > requestedTemp ∧ currentTemp ≠ **nil** ∧ requestedTemp ≠ **nil**
post currentTemp = $\overline{\text{currentTemp}}$ - 0.5 ∧
(currentTemp > requestedTemp ∧ signalOut = <DECREASE_TEMP> ∨
currentTemp = requestedTemp ∧ signalOut = <DO_NOTHING>)

-- an operation that instructs the hardware to ring an alarm when the temperature exceeds the defined range
alarmActivation ( ) signalOut : Signal
ext wr alarmActivated : [*B*]
rd currentTemp : [ℤ]
pre currentTemp > requestedTemp ∧ currentTemp ≠ **nil** ∧ requestedTemp ≠ **nil**
post alarmActivated = TRUE ∧
((currentTemp < MIN_TEMP ∨ currentTemp > MAX_TEMP) ∧ signalOut = <RING_ALARM>
∨ inRange (currentTemp) ∧ signalOut = <DO_NOTHING>)

-- an operation that returns the current temperature of the system
getCurrentTemp ( ) currentActual : [ℤ]
ext rd currentTemp : [ℤ]
pre TRUE
post currentActual = currentTemp

-- an operation that returns the requested temperature of the system
getRequestedTemp ( ) currentRequested : [ℤ]
ext rd requestedTemp : [ℤ]
pre TRUE
post currentRequested = requestedTemp

6. JAVA CODE:

Signal.java:

```
public enum Signal {INCREASE_TEMP, DECREASE_TEMP, RING_ALARM, DO_NOTHING}
```

VendTempController.java:

```
public class VendTempController {
    private int currentTemp;
    private Integer requestedTemp;

    public VendTempController() {
       currentTemp = 2;
       requestedTemp = null;}

    private boolean inRange(int val) {return val >= 2 && val <= 8;}

    public void setInitialTemp(int initialTemp) {
       if (inRange(initialTemp) && currentTemp == 2) {
          currentTemp = initialTemp;}}

    public Signal requestChange(int initialTemp) {
       if (inRange(initialTemp) && currentTemp != 0) {
          requestedTemp = initialTemp;
          if (initialTemp > currentTemp) {return Signal.INCREASE_TEMP;}
          else if (initialTemp < currentTemp) {return Signal.DECREASE_TEMP;}
          else {return Signal.DO_NOTHING;}}
       return null;}

    public Signal increaseTemp() {
       if (currentTemp < requestedTemp && currentTemp != 0 && requestedTemp != null) {
          currentTemp += 0.5;
          if (currentTemp < requestedTemp) {return Signal.INCREASE_TEMP;}
          else {return Signal.DO_NOTHING;}}
       return null;}

    public Signal decreaseTemp() {
       if (currentTemp > requestedTemp && currentTemp != 0 && requestedTemp != null) {
          currentTemp -= 0.5;
          if (currentTemp > requestedTemp) {return Signal.DECREASE_TEMP;}
          else {return Signal.DO_NOTHING;}}
       else if (currentTemp == 2 || currentTemp == 8) {return Signal.DO_NOTHING;}
       return null;}

    public Signal alarmActivation() {
       if (currentTemp > requestedTemp && currentTemp != 0 && requestedTemp != null) {
```

```
        if (currentTemp < 2 || currentTemp > 8) {return Signal.RING_ALARM;}
        else {return Signal.DO_NOTHING;}}
      return null;}

    public int getCurrentTemp() {return currentTemp;}
    public Integer getRequestedTemp() {return requestedTemp;}

    public void PrintSignals() {
      Signal increaseSignal = increaseTemp();
      Signal decreaseSignal = decreaseTemp();
      Signal alarmSignal = alarmActivation();
      System.out.println("Increase Signal: " + increaseSignal);
      System.out.println("Decrease Signal: " + decreaseSignal);
      System.out.println("Alarm Signal: " + alarmSignal);}

  public static void main(String[] args) {
    VendTempController controller = new VendTempController();
    controller.setInitialTemp(5);
    // Case 1: Increase Temperature
    Signal increaseSignal = controller.requestChange(7);
    controller.PrintSignals();
    System.out.println("Current Temperature: " + controller.getCurrentTemp());
    System.out.println("Requested Temperature: " + controller.getRequestedTemp());
    System.out.println();
    // Case 2: Decrease Temperature
    Signal decreaseSignal = controller.requestChange(3);
    controller.PrintSignals();
    System.out.println("Current Temperature: " + controller.getCurrentTemp());
    System.out.println("Requested Temperature: " + controller.getRequestedTemp());}}
```

OUTPUT WITHOUT PRINTING THE FUNCTIONALITY RESULTS:

```
Increase Signal: INCREASE_TEMP
Current Temperature: 5
Requested Temperature: 7
Decrease Signal: DECREASE_TEMP
Decrease Signal: null
Current Temperature: 5
Requested Temperature: 3
```

OUTPUT WITH STEP BY STEP EXECUTION ACCORDING TO VDM SPECIFICATIONS

```
Increase Signal: INCREASE_TEMP
Decrease Signal: null
Alarm Signal: null
Current Temperature: 5
Requested Temperature: 7

Increase Signal: null
Decrease Signal: DECREASE_TEMP
Alarm Signal: DO_NOTHING
Current Temperature: 4
Requested Temperature: 3
```

EXPLANATION OF CODE ACCORDING TO VDM STEPS:

SCENARIO 1: INCREASE IN TEMPERATURE FROM 5 TO 7

- **Signal: INCREASE_TEMP:** This line indicates the signal received after the first request to change the temperature from 5 to 7.The signal received is **INCREASE_TEMP**, suggesting that the requested temperature is higher than the current temperature.
- **Increase Signal: INCREASE_TEMP:** After attempting to increase the temperature, the method returned an **INCREASE_TEMP** signal. This indicates that the system successfully increased the temperature by 0.5 units.
- **Decrease Signal: null:** The attempt to decrease the temperature did not result in any change. The method returned **null**, indicating that the decrease did not occur, because the conditions for decreasing the temperature were not met.
- **Alarm Signal: null:** The alarm signal was not triggered. The method returned **null**, indicating that the alarm conditions were not met, and hence, no alarm was activated.
- **Current Temperature: 5:** This displays the current temperature, which remains at 5 after the operations.
- **Requested Temperature: 7:** This shows the requested temperature, which is set to 7 initially.

In summary, there was an initial increase in temperature from 5 to 7. The attempts to decrease the temperature or trigger an alarm did not result in any change or alarm activation.

SCENARIO 2: DECREASE IN TEMPERATURE FROM 4 TO 3

- **Signal: DECREASE_TEMP:** This line indicates the signal received after the second request to change the temperature from 4 to 3. The signal received is **DECREASE_TEMP**, suggesting that the requested temperature is lower than the current temperature.
- **Increase Signal: null:** The attempt to increase the temperature did not result in any change. The method returned **null**, indicating that the increase did not occur because the conditions for increasing the temperature were not met.

- **Decrease Signal: DECREASE_TEMP:** After attempting to decrease the temperature, the method returned a **DECREASE_TEMP** signal. This indicates that the system successfully decreased the temperature by 0.5 units.
- **Alarm Signal: null:** The alarm signal was not triggered. The method returned null, indicating that the alarm conditions were not met, and hence, no alarm was activated.
- **Current Temperature: 4:** This displays the current temperature, which remains at 4 after the operations.
- **Requested Temperature: 3:** This shows the requested temperature, which is set to 3 initially.

In this scenario, there was a decrease in temperature from 4 to 3. The attempts to increase the temperature or trigger an alarm did not result in any change or alarm activation.

7. TESTING CLASS:

VendTempControllerTest.java:

```
public class VendTempControllerTest {
   public static void main(String[] args) {
      testSetInitialTemp();
      testRequestChange();
      testIncreaseTemp();
      testDecreaseTemp();
      testAlarmActivation();
      testGetCurrentTemp();
      testGetRequestedTemp();}

   private static void testSetInitialTemp() {
      VendTempController controller = new VendTempController();
      System.out.println("Testing setInitialTemp:");
      try {
         // Valid initial temperature
         controller.setInitialTemp(5);
                        System.out.println("Current Temperature after setting initial temp to 5: " +
controller.getCurrentTemp());
         // Invalid initial temperature (out of range)
         controller.setInitialTemp(10);
                       System.out.println("Current Temperature after setting invalid initial temp: " +
controller.getCurrentTemp());
      } catch (Exception e) {
         System.out.println("An error occurred: " + e.getMessage());}}

   private static void testRequestChange() {
      VendTempController controller = new VendTempController();
      System.out.println("\nTesting requestChange:");
```

```
    try {
        // Valid change request (increase temperature)
        controller.setInitialTemp(5);
        Signal signal = controller.requestChange(7);
        System.out.println("Signal for increasing temperature: " + signal);
        // Valid change request (decrease temperature)
        signal = controller.requestChange(3);
        System.out.println("Signal for decreasing temperature: " + signal);
        // No change request (same temperature)
        signal = controller.requestChange(5);
        System.out.println("Signal for no change in temperature: " + signal);
    } catch (Exception e) {
        System.out.println("An error occurred: " + e.getMessage());}}

private static void testIncreaseTemp() {
    VendTempController controller = new VendTempController();
    System.out.println("\nTesting increaseTemp:");
    try {
        // Increase temperature
        controller.setInitialTemp(5);
        controller.requestChange(7);
        Signal signal = controller.increaseTemp();
        System.out.println("Signal for increasing temperature: " + signal);
        // No increase needed (already at requested temperature)
        signal = controller.increaseTemp();
        System.out.println("Signal for no increase in temperature: " + signal);
    } catch (Exception e) {
        System.out.println("An error occurred: " + e.getMessage());}}

private static void testDecreaseTemp() {
    VendTempController controller = new VendTempController();
    System.out.println("\nTesting decreaseTemp:");
    try {
        // Decrease temperature
        controller.setInitialTemp(5);
        controller.requestChange(3);
        Signal signal = controller.decreaseTemp();
        System.out.println("Signal for decreasing temperature: " + signal);
        // No decrease needed (already at requested temperature)
        signal = controller.decreaseTemp();
        System.out.println("Signal for no decrease in temperature: " + signal);
        // No decrease needed (at minimum temperature)
        controller.setInitialTemp(2);
        signal = controller.decreaseTemp();
```

```
            System.out.println("Signal for no decrease at minimum temperature: " + signal);
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());}}

    private static void testAlarmActivation() {
        VendTempController controller = new VendTempController();
        System.out.println("\nTesting alarmActivation:");
        try {
            // Activate alarm (temperature out of range)
            controller.setInitialTemp(10);
            controller.requestChange(8);
            Signal signal = controller.alarmActivation();
            System.out.println("Signal for activating alarm: " + signal);
            // No alarm activation (within range)
            controller.setInitialTemp(5);
            controller.requestChange(7);
            signal = controller.alarmActivation();
            System.out.println("Signal for no alarm activation: " + signal);
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());}}

    private static void testGetCurrentTemp() {
        VendTempController controller = new VendTempController();
        System.out.println("\nTesting getCurrentTemp:");
        try {
            // Get current temperature
            int currentTemp = controller.getCurrentTemp();
            System.out.println("Current Temperature: " + currentTemp);
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());}}

    private static void testGetRequestedTemp() {
        VendTempController controller = new VendTempController();
        System.out.println("\nTesting getRequestedTemp:");
        try {
            // Get requested temperature
            Integer requestedTemp = controller.getRequestedTemp();
            System.out.println("Requested Temperature: " + requestedTemp);
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());}}}
```

OUTPUT:

```
Testing setInitialTemp:
Current Temperature after setting initial temp to 5: 5
Current Temperature after setting invalid initial temp: 5

Testing requestChange:
Signal for increasing temperature: INCREASE_TEMP
Signal for decreasing temperature: DECREASE_TEMP
Signal for no change in temperature: DO_NOTHING

Testing increaseTemp:
Signal for increasing temperature: INCREASE_TEMP
Signal for no increase in temperature: INCREASE_TEMP

Testing decreaseTemp:
Signal for decreasing temperature: DECREASE_TEMP
Signal for no decrease in temperature: DO_NOTHING
Signal for no decrease at minimum temperature: null

Testing alarmActivation:
Signal for activating alarm: null
Signal for no alarm activation: null

Testing getCurrentTemp:
Current Temperature: 2

Testing getRequestedTemp:
Requested Temperature: null
```

EXPLANATION OF TEST CASES:

1. **testSetInitialTemp:**
   - **Purpose:** To verify that the setInitialTemp method properly initializes the current temperature.
   - **Scenario:**
     1. Set a valid initial temperature (5).
     2. Display the current temperature after the valid initialization.
     3. Attempt to set an invalid initial temperature (10).
     4. Display the current temperature.
   - **Verification:** Ensure that the current temperature is correctly initialized and updated according to the valid and invalid input.

2. **testRequestChange:**
   - **Purpose:** To verify the correctness of the requestChange method in generating appropriate signals.
   - **Scenario:**
     1. Set an initial temperature (5).
     2. Request an increase in temperature (7) and display the corresponding signal.
     3. Request a decrease in temperature (3) and display the corresponding signal.
     4. Request no change in temperature (5) and display the corresponding signal.

- **Verification:** Confirm that the signals generated match the expected behavior for different temperature change requests.

3. **testIncreaseTemp:**
   - **Purpose:** To verify the behavior of the increaseTemp method in adjusting the temperature.
   - **Scenario:**
     1. Set an initial temperature (5).
     2. Request an increase in temperature (7).
     3. Display the signal for increasing temperature.
     4. Attempt to increase temperature again when it's already at the requested temperature.
     5. Display the signal for no increase in temperature.
   - **Verification:** Ensure that the temperature is adjusted correctly, and the signals align with the expected outcomes.

4. **testDecreaseTemp:**
   - **Purpose:** To verify the behavior of the decreaseTemp method in adjusting the temperature.
   - **Scenario:**
     1. Set an initial temperature (5).
     2. Request a decrease in temperature (3).
     3. Display the signal for decreasing temperature.
     4. Attempt to decrease temperature again when it's already at the requested temperature.
     5. Display the signal for no decrease in temperature.
     6. Set the initial temperature to the minimum (2).
     7. Attempt to decrease further.
     8. Display the signal for no decrease at the minimum temperature.
   - **Verification:** Confirm that the temperature is adjusted correctly, and the signals align with the expected outcomes, considering the minimum temperature constraint.

5. **testAlarmActivation:**
   - **Purpose:** To verify that the alarmActivation method correctly triggers alarms based on temperature conditions.
   - **Scenario:**
     1. Set an initial temperature outside the valid range (10).
     2. Request a temperature change within the valid range (8).
     3. Display the signal for activating the alarm due to the temperature being out of range.
     4. Set an initial temperature within the valid range (5).
     5. Request a temperature change within the valid range (7).
     6. Display the signal for no alarm activation.
   - **Verification:** Ensure that alarms are triggered appropriately based on temperature conditions.

6. **testGetCurrentTemp:**
   - **Purpose:** To verify that the getCurrentTemp method returns the correct current temperature.
   - **Scenario:**
     1. Get the current temperature.

- **Verification:** Confirm that the returned current temperature matches the expected value.

7. **testGetRequestedTemp:**
   - **Purpose:** To verify that the getRequestedTemp method returns the correct requested temperature.
   - **Scenario:**
     1. Get the requested temperature.
   - **Verification:** Confirm that the returned requested temperature matches the expected value.

OVERALL WORKING:

- **Initialization:** The VendTempControllerTester initializes a VendTempController object.
- **Test Execution:** The tester method sequentially calls methods of VendTempController to test various functionalities such as setting initial temperature, requesting temperature change, increasing/decreasing temperature, and checking alarm activation.
- **Error Handling:** The try-catch block captures exceptions that may occur during the method calls. These exceptions are due to deliberate violations of VDM constraints in specific test cases (like trying to increase the temperature when not allowed or activating the alarm in invalid conditions).
- **Output Display:** Outputs relevant information about the signal or current temperature based on the actions performed.
- **Error Reporting:** If any exception occurs during the test cases, the catch block displays an error message with details of the exception that was caught.

This overall process systematically checks the behavior of different methods of VendTempController against both expected outcomes and VDM-defined constraints, ensuring the correctness and compliance of the controller's functionalities.

8. CONCLUSION:

This VDM-SL specification defines the formal requirements and behavior of the Vending Machine Temperature Controller. It ensures that the system consistently maintains the temperature within the specified range and activates an alarm when necessary. The provided operations can be used as a basis for implementing the temperature control system using VDM-SL.