NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Formal Methods in Software Engineering

Formal Specification Document for

"Nuclear Fission Temperature Controller"

Group

- Umme Hani (SE-21006)
- Syeda Ramsha (SE-21011)

Table of Contents

Nuclear Fission Temperature Controller

1. Problem Statement:

Nuclear Fission Overview:

Nuclear fission is a process in which the nucleus of an atom splits into two or more smaller nuclei, releasing a significant amount of energy. This process is at the heart of nuclear power generation. When a heavy nucleus, such as uranium-235 or plutonium-239, undergoes fission, it releases neutrons and a substantial amount of heat.

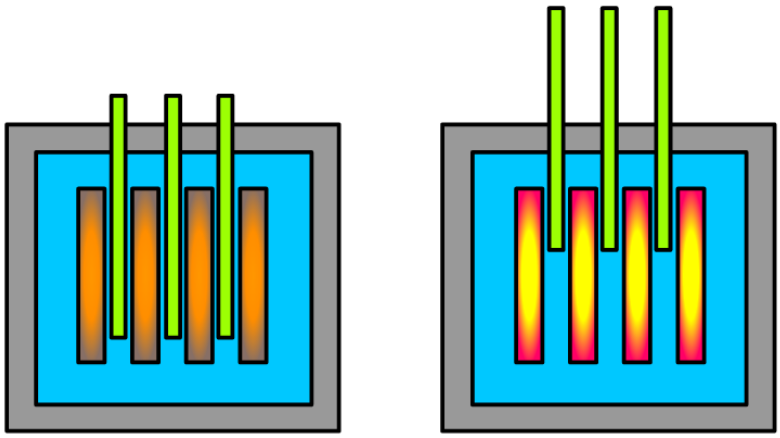
Importance of Temperature Control in Nuclear Fission:

The nuclear fission temperature controller system is crucial for preventing catastrophic consequences such as reactor core meltdown and radioactive releases. Analyzing real-time sensor data, it sends signals to control mechanisms, including control rods and emergency systems, to regulate fission processes and maintain optimal operating temperatures, ensuring overall nuclear power generation safety.

Sensors: Temperature sensors continuously provide real-time temperature data to the temperature controller system, ensuring it has accurate and up-to-date information about the reactor's thermal conditions.

Control Rods (CRDs):

- Control rods, composed of materials that absorb neutrons, are used to regulate the rate of fission reactions within the reactor core.
- The temperature controller system collaborates with the Control rod drive system, sending signals based on temperature readings. If the temperature exceeds 70°C, it inserts control rods to absorb more neutrons, slowing fission and reducing heat.
- Conversely, if the temperature drops below 50°C, it withdraws control rods, allowing more neutrons for fission, adjusting the rate to maintain safe temperatures.



Coolant System: In emergencies signaled by the temperature controller system, the coolant system adjusts coolant flow, dissipating excess heat and preventing unsafe temperature levels.

Alarm Systems: Linked with the temperature controller, alarm systems trigger alerts if temperatures surpass 150°C. Immediate operator notifications enable timely corrective actions, enhancing reactor safety.

Functionalities:

- Get Temperature
  Input: Recorded temperature value sent by sensor that must be integer
  Precondition: Recorded temperature should not be undefined
  Post condition: The recorded temperature is equal to the received temperature.

- Check Temperature
  Precondition: Received temperature should not be undefined.
  Post condition: If the temperature is greater than 70°C but less than 150°C, the insert rod operation is called. If the temperature is less than 50°C, the withdraw rod function is called.

- Insert Rods
  Precondition: A signal indicating the need for fewer rods.
  Post condition: Signals are sent to insert more rods into the system.

- Withdraw Rods
  Precondition: A signal indicating the need for more rods.
  Post condition: Signals are sent to insert fewer rods into the system.

- Emergency Control
  The received temperature should not be undefined, and it is greater than the critical temperature.
  Post condition: Calls the function to activate the coolant system and rings the alarm until the temperature returns to normal.

- Ring Alarm
  Precondition: The alarm is off.
  Post condition: The alarm is activated.

- Stop Alarm
  Precondition: The alarm is on.
  Post condition: The alarm is turned off.

- Activate Coolant
  Precondition: The coolant system is off.
  Post condition: The coolant system is activated.

- Deactivate Coolant
  Precondition: The coolant system is on.
  Post condition: The coolant system is deactivated.

2. Vienna Development Model

types

AlarmSignal = <OFF>| <ON>
CoolantSignal = <OFF>| <ON>
RodSignal= <MORE> | <LESS>| <NOCHANGE>

values
MIN_TEMP: Z= 50.0;
MAX_TEMP: Z= 70.0;
CRITICAL_TEMP: Z > 150.0;

state *FissionTemperatureMonitor* **of**
receivedTemp : [Z]

-- Received Temperature can be nil and also at some conditions cannot be nil
inv mk- *FissionTemperatureMonitor* (r) $\triangleq$ (validTemp (r) $\vee$ r = **nil)**

-- Received temperature, rod status, alarm status and coolant status are undefined when the system is initialized
init mk- *FissionTemperatureMonitor* (r, a, c,t) $\triangleq$ r = **nil** $\wedge$ a= **nil** $\wedge$ c = **nil** $\wedge$ t = **nil**
end

functions
validTemp:(val: Z) result: B
pre True
post result $\Leftrightarrow$ (val $\neq$ nil);

operations
--an operation that records the initial temperature of the system
--received temp is undefined initially
--recordedTemp is input received by through signal ,which should not be nil

getTemperature: (recordedTemp: Z)
ext wr receivedTemp:[Z]
pre validTemp(**recordedTemp**)
post receivedTemperature = recordedTemp

--an operation that checks the temperature of the system and signals the control --rod drive system to insert or withdraw rods to set the temperature as appropriate

checkTemperature( )
ext rd receivedTemp:[Z]
pre validTemp(**receivedTemp**)
post (CRITICAL_temp> receivedTemp ≥ MAX_TEMP∧ insertRod()) ∨ (temp ≤ MIN_TEMP ∧ withdrawRod()) ∨ (MIN_TEMP <receivedTemp< MAX_TEMP∧ optimalRods())

insertRod() rodStatus: RodSignal
pre rodStatus = <LESS>
post rodStatus = <MORE>

withdrawRod() rodStatus: RodSignal
pre rodStatus = <MORE>
post rodStatus = <LESS>

optimalRods() rodStatus: RodSignal
pre rodStatus = TRUE
post rodStatus = <NOCHANGE>

--an operation that records the critical temperature of the system and signals the --coolant system to ring alarm until it gets less than the critical tempearture

emergencyControl() coolantStatus : CoolantSignal
ext rd recievedTemp: [Z]
pre validTemp(**receivedTemp**) ∧ receivedTemp > CRITICAL_TEMP
post (receivedTemp > = CRITICAL_TEMP ∧ ringAlarm() ∧ activateCoolant()) ∨ (receivedTemp < CRITICAL_TEMP ∧ stopAlarm() ∧ deactiavteCoolant())

ringAlarm() alarmStatus : AlarmSignal
pre alarmStatus = <OFF>
post alarmStatus = <ON>

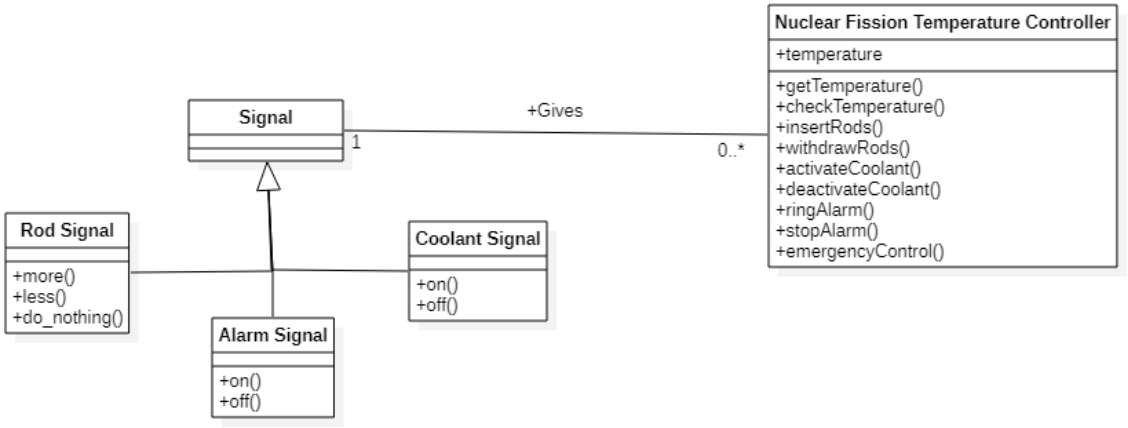
activateCoolant() coolantStatus : CoolantSignal
pre coolantStatus = <OFF>
post coolantStatus = <ON>

stopAlarm() alarmStatus : AlarmSignal
pre alarmStatus = <ONN>
post alarmStatus = <OFF>

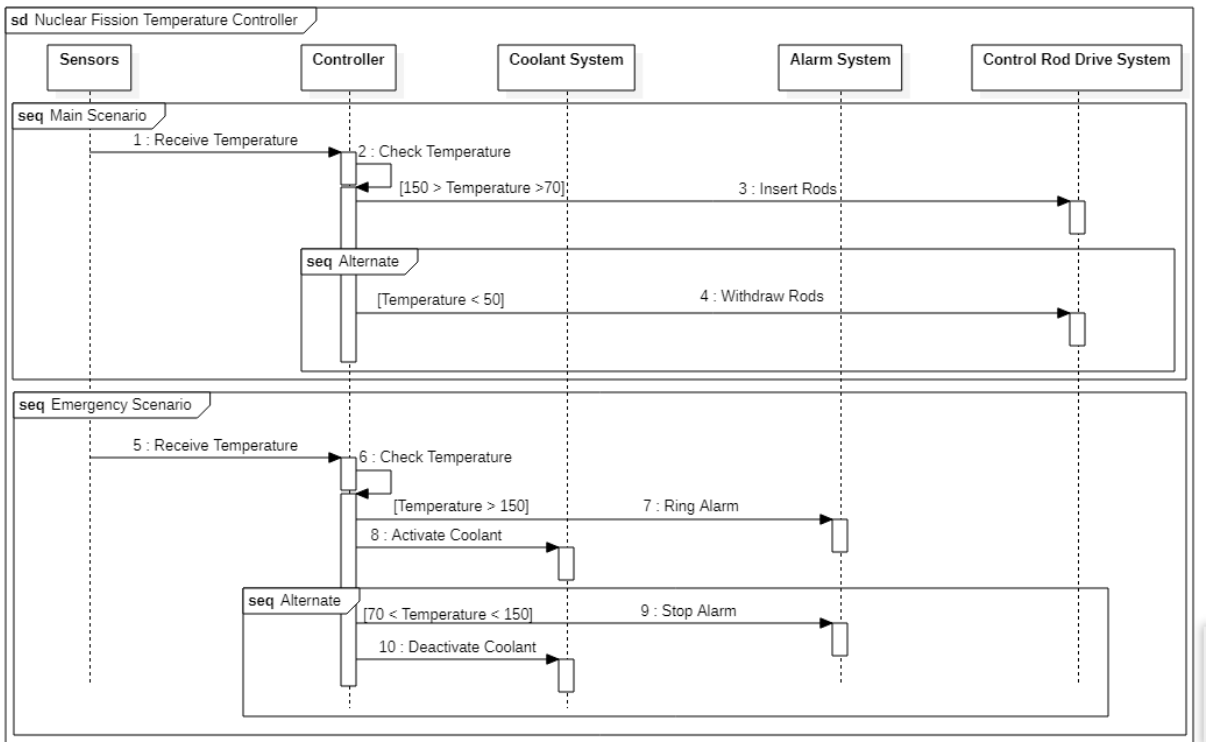
deactivateCoolant() coolantStatus : CoolantSignal
pre coolantStatus = <ON>
post coolantStatus = <OFF>

3. 4+1 Architectural View

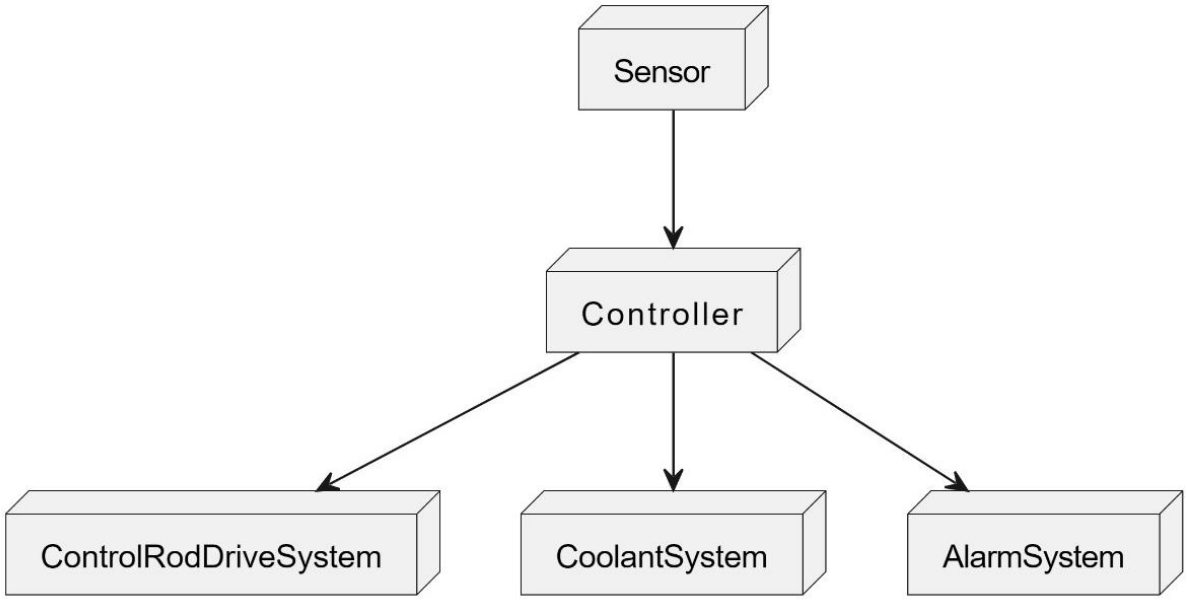
3.1. Logical View



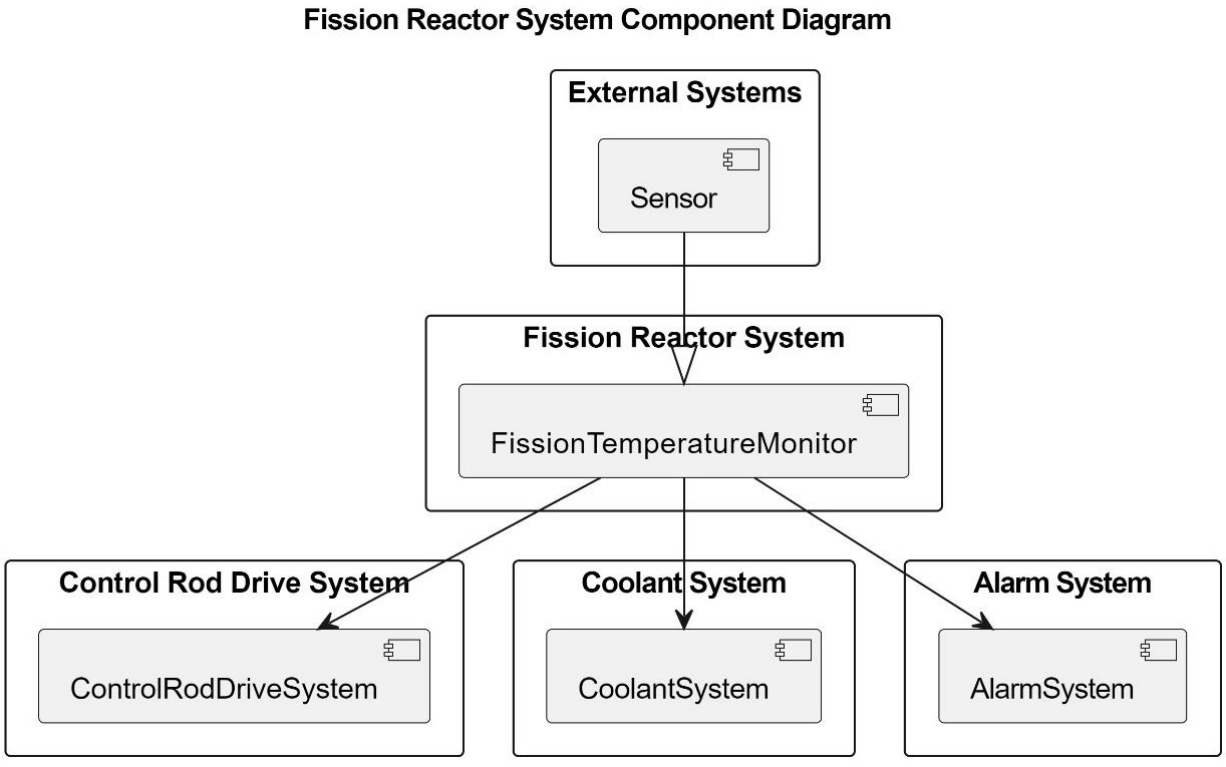
3.2. Process View



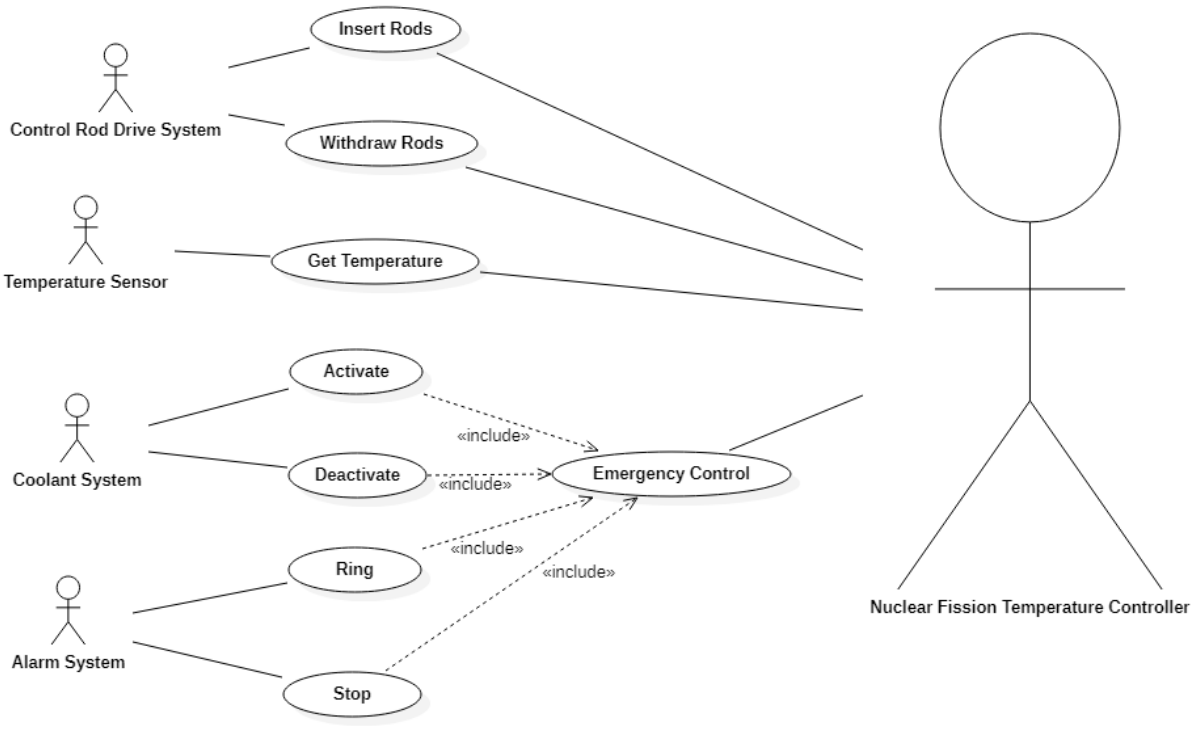
3.3. Physical View



3.4. Development View



3.5. +1 Scenario



4. Java Implementation

```
// Enum for AlarmSignal
enum AlarmSignal {
    OFF,
    ON
}

// Enum for CoolantSignal
enum CoolantSignal {
    OFF,
    ON
}

// Enum for RodSignal
enum RodSignal {
    MORE,
    LESS,
    NOCHANGE
}

// Temperature class
class Temperature {
    private Double value;

    public Temperature(Double value) {
        this.value = value;
```

```
    }

    public Double getValue() {
        return value;
    }
}

// Class representing the state of FissionTemperatureMonitor
class FissionTemperatureMonitor {
    private Temperature receivedTemp;
    private AlarmSignal alarmStatus;
    private CoolantSignal coolantStatus;
    private RodSignal rodStatus;

    // Constructor
    public FissionTemperatureMonitor() {
        this.receivedTemp = null;
        this.alarmStatus = AlarmSignal.OFF;
        this.coolantStatus = CoolantSignal.OFF;
        this.rodStatus = RodSignal.NOCHANGE;
    }

    // Invariant
    public boolean validTemp(Temperature temp) {
        return temp != null && temp.getValue() != null;
    }
```

```
    // Function to record the initial temperature of the system
    public void getTemperature(Temperature recordedTemp) {
        assert validTemp(recordedTemp) : "Recorded temperature should not be
null";
        this.receivedTemp = recordedTemp;
    }

    // Function to check the temperature of the system
    public void checkTemperature() {
        assert validTemp(receivedTemp) : "Received temperature should not be
null";

        if (receivedTemp.getValue() < 50.0){
            withdrawRod();
        }
        else if (receivedTemp.getValue() >= 50.0&& receivedTemp.getValue() <=
70.0) {
            optimalRods();
        }

        else if (150.0 > receivedTemp.getValue() && receivedTemp.getValue() >
70.0) {
            insertRod();
        }

        else if (receivedTemp.getValue() >= 150.0) {
            optimalRods();
```

```
    }

}

// Function to insert rods
private void insertRod() {
    if (rodStatus == RodSignal.LESS || rodStatus == RodSignal.NOCHANGE) {
        rodStatus = RodSignal.MORE;
    }
}

// Function to withdraw rods
private void withdrawRod() {
    if (rodStatus == RodSignal.MORE || rodStatus == RodSignal.NOCHANGE) {
        rodStatus = RodSignal.LESS;
    }
}

// Function for optimal rod status
private void optimalRods() {
    if (rodStatus == RodSignal.NOCHANGE) {
        // Your logic for optimal rod status here
        // For now, it remains NOCHANGE
    }
}
```

```
    // Function for emergency control
    public void emergencyControl() {
        assert validTemp(receivedTemp) : "Received temperature should not be
null";

        if (receivedTemp.getValue() >= 150.0) {
            ringAlarm();
            activateCoolant();
        } else {
            stopAlarm();
            deactivateCoolant();
        }
    }

    // Function to ring the alarm
    private void ringAlarm() {
        if (alarmStatus == AlarmSignal.OFF) {
            alarmStatus = AlarmSignal.ON;
        }
    }

    // Function to activate the coolant
    private void activateCoolant() {
        if (coolantStatus == CoolantSignal.OFF) {
            coolantStatus = CoolantSignal.ON;
        }
```

```
}

// Function to stop the alarm
private void stopAlarm() {
    if (alarmStatus == AlarmSignal.ON) {
        alarmStatus = AlarmSignal.OFF;
    }
}

// Function to deactivate the coolant
private void deactivateCoolant() {
    if (coolantStatus == CoolantSignal.ON) {
        coolantStatus = CoolantSignal.OFF;
    }
}

// Getter for alarm status
public AlarmSignal getAlarmStatus() {
    return alarmStatus;
}

// Getter for coolant status
public CoolantSignal getCoolantStatus() {
    return coolantStatus;
}
```

```
    // Getter for rod status
    public RodSignal getRodStatus() {
        return rodStatus;
    }
}
public class Main {
    public static void main(String[] args) {
        FissionTemperatureMonitor monitor = new FissionTemperatureMonitor();

        // Example: Valid recorded temperature, should print signals
        Temperature recordedTemp = new Temperature(90.0);
        monitor.getTemperature(recordedTemp);
        monitor.checkTemperature();
        monitor.emergencyControl();
        printStatus(monitor);
    }

    private static void printStatus(FissionTemperatureMonitor monitor) {
        System.out.println("==== Current System Status ====");
        System.out.println("Alarm Status: " + monitor.getAlarmStatus());
        System.out.println("Coolant Status: " + monitor.getCoolantStatus());
        System.out.println("Rod Status: " + monitor.getRodStatus());
        System.out.println("==============================");
    }
};
```

Output

```
==== Current System Status ====
Alarm Status: OFF
Coolant Status: OFF
Rod Status: MORE
================================
```

5. Testing Class

```
import java.util.Scanner;
public class FissionTemperatureMonitorMain {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        FissionTemperatureMonitor monitor = new FissionTemperatureMonitor();

        int choice;
        do {
            System.out.println("==== Temperature Monitor Menu ====");
            System.out.println("1. Record Temperature");
            System.out.println("2. Print Status");
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");

            try {
                choice = scanner.nextInt();
                scanner.nextLine(); // Consume the newline character
            } catch (java.util.InputMismatchException e) {
                System.out.println("Invalid input. Please enter a number.");
                scanner.nextLine(); // Consume the invalid input
                choice = -1;
            }

            switch (choice) {
```

```
                case 1:
                    recordAndCheckTemperature(monitor, scanner);
                    break;
                case 2:
                    printStatus(monitor);
                    break;
                case 0:
                    System.out.println("Exiting Temperature Monitor.
Goodbye!");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid
option.");
                    break;
            }
        } while (choice != 0);


        scanner.close();
    }


    private static void recordAndCheckTemperature(FissionTemperatureMonitor
monitor, Scanner scanner) {
        System.out.print("Enter recorded temperature: ");
        try {
            double recordedTemp = scanner.nextDouble();
            monitor.getTemperature(new Temperature(recordedTemp));
            System.out.println("Temperature recorded.");
```

```
            // Check temperature and perform emergency control
            monitor.checkTemperature();
            monitor.emergencyControl();
            System.out.println("Temperature checked. ");
        } catch (java.util.InputMismatchException e) {
            System.out.println("Invalid input for temperature. Please enter a
valid number.");
            scanner.nextLine(); // Consume the invalid input
        } catch (AssertionError e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void printStatus(FissionTemperatureMonitor monitor) {
        System.out.println("==== Current System Status ====");
        System.out.println("Alarm Status: " + monitor.getAlarmStatus());
        System.out.println("Coolant Status: " + monitor.getCoolantStatus());
        System.out.println("Rod Status: " + monitor.getRodStatus());
        System.out.println("==============================");
    }
}
```

Output

```
==============================
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 50
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: OFF
Coolant Status: OFF
Rod Status: NOCHANGE
==============================
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 70
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: OFF
Coolant Status: OFF
Rod Status: NOCHANGE
==============================
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 150
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: ON
Coolant Status: ON
Rod Status: NOCHANGE
==============================
```

```
================================
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 65
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: OFF
Coolant Status: OFF
Rod Status: NOCHANGE
================================
```

```
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 185
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: ON
Coolant Status: ON
Rod Status: NOCHANGE
================================
```

```
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 34
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: OFF
Coolant Status: OFF
Rod Status: LESS
================================
```

```
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 1
Enter recorded temperature: 90
Temperature recorded.
Temperature checked.
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 2
==== Current System Status ====
Alarm Status: OFF
Coolant Status: OFF
Rod Status: MORE
==============================
==== Temperature Monitor Menu ====
1. Record Temperature
2. Print Status
0. Exit
Enter your choice: 0
Exiting Temperature Monitor. Goodbye!
```