# Smart Home Automation System - Implementation Rubric

## OOP Features Implementation Table

| Requirement | Minimum Required | Actual Implementation | Locations in Code |
|---|---|---|---|
| **(I)** Overloaded methods | 2 | 3 | • `Admin.login(Scanner)` and `RegularUser.login(Scanner)`<br>• `Security.validateUsernames(String, String)` and `Security.validateUsernames(String, String...)`<br>• `ControllableDevice.getStatus()` and `ControllableDevice.getDeviceStatus()` |
| **(II)** Overloaded constructors | 2 | 4 | • `ControllableDevice()`, `ControllableDevice(double)`, and `ControllableDevice(String, double)`<br>• `Admin(String, String)` and `Admin(String, String, ControllableDevice...)` |
| **(III)** Vararg overloading | 2 | 2 | • `Security.validateUsernames(String, String...)`<br>• `Admin(String, String, ControllableDevice...)` |
| **(IV)** Nested classes | 1 | 3 | • `Security.PasswordHandler` (static nested class)<br>• `ControllableDevice.DeviceStatus` (static nested class)<br>• `SmartHomeException.DeviceControlException` (nested class) |
| **(V)** Abstract class | 1 | 2 | • `User` (abstract class with abstract method `getRole()`)<br>• `ControllableDevice` (abstract class with abstract methods) |
| **(VI)** Interface | 1 | 1 | • `Automation` interface in `smarthome.automation` package |
| **(VII)** Hierarchical Inheritance | 1 | 2 | • `User -> Admin, RegularUser`<br>• `ControllableDevice -> Light, Fan, AC` |
| **(VIII)** Multiple Inheritance | 1 | 1 | • Classes implementing multiple interfaces and extending classes simultaneously (e.g., `RegularUser extends User implements Serializable`) |
| **(IX)** Wrappers | Required | 4 | • `Boolean` wrapper in `Security.PasswordHandler.isStrongPassword()`<br>• `Character` wrapper in `Security.PasswordHandler.toCharArray()`<br>• `Integer.parseInt()` for input conversions<br>• `Double` wrapper for energy calculations |

| Requirement | Minimum Required | Actual Implementation | Locations in Code |
|---|---|---|---|
| **(X)** Package | Required | 6 | • `smarthome.auth`<br>• `smarthome.devices`<br>• `smarthome.automation`<br>• `smarthome.io`<br>• `smarthome.exceptions`<br>• `smarthome.main` |
| **(XI)** Exception handling | 2 cases | 10+ | • Custom exception classes through `SmartHomeException`<br>• File I/O exception handling in `FileHandler`<br>• Input validation try-catch in UI components<br>• Thread interruption handling<br>• Device control exception handling<br>• User authentication error handling<br>• Data validation exceptions |
| **(XII)** I/O: File Handling | At least one | 3 | • Object serialization for user data in `FileHandler.saveUsers()`<br>• Text file logging in `FileHandler.logEvent()`<br>• User data loading in `FileHandler.loadUsers()` |
| **(XIII)** Multithreading | Required | 1 | • `AutomationThread` implementing `Runnable` interface |

## Detailed Implementation Notes

### I. Overloaded Methods

The system implements multiple overloaded methods throughout the codebase:

1. Authentication methods: Different implementations for Admin and RegularUser
2. Security validation methods: Multiple ways to validate usernames
3. Status methods: Different ways to retrieve device statuses

### II. Overloaded Constructors

Multiple constructors are provided for flexibility:

1. `ControllableDevice` constructors with different parameter sets
2. `Admin` constructors allowing creation with or without initial devices

### III. Vararg Overloading

The system uses variable arguments in:

1. Security validation to allow checking multiple usernames
2. Admin constructor to accept a variable number of initial devices

### IV. Nested Classes

Several nested classes improve code organization:

1. `PasswordHandler` provides security functionality
2. `DeviceStatus` encapsulates device state information
3. `DeviceControlException` specializes exception handling

### V. Abstract Classes

Two key abstract classes form the foundation of the system:

1. `User` defines common user properties with abstract role designation
2. `ControllableDevice` provides common device functionality with abstract device-specific methods

### VI. Interfaces

The `Automation` interface provides a contract for automation functionality across different components.

### VII. Hierarchical Inheritance

Two main inheritance hierarchies organize system entities:

1. User hierarchy for authentication and permissions
2. Device hierarchy for different device types and behaviors

### VIII. Multiple Inheritance

Several classes implement multiple inheritance through interfaces while extending base classes.

### IX. Wrappers

Wrapper classes are used throughout the system:

1. Boolean wrappers for validation results
2. Character wrappers for secure password handling
3. Numeric wrappers for parsing and calculations

### X. Packages

The system is organized into six logical packages:

1. `auth`: User authentication and management
2. `devices`: Smart device implementations
3. `automation`: Scheduling and automation logic
4. `io`: File operations and persistence

5. `exceptions`: Custom exception handling
6. `main`: Application entry points and UI

## XI. Exception Handling

Comprehensive exception handling throughout the system:

1. Custom exceptions for domain-specific errors
2. Try-catch blocks for input validation
3. I/O exception handling for file operations
4. Thread interruption handling in automation
5. Graceful error recovery throughout the UI

## XII. I/O File Handling

The system uses file operations for:

1. User data persistence through serialization
2. System event logging
3. Data loading on startup

## XIII. Multithreading

Automation is handled through a dedicated thread:

1. `AutomationThread` implements Runnable
2. Thread management in main application class
3. Safe thread termination handling