

Assignment 1

Smart Home Automation Problem

Scenario:

You are a software developer working on a **smart home automation system**. The system needs to control multiple devices in the house, including **lights**, a **thermostat**, and a **security system**. Your task is to create a command-line interface (CLI) application using Dart to manage the smart home's functionality. The system will handle user input to toggle devices, adjust temperature, and monitor the system's status.

Requirements:

1. Main Method:

The program should start with a `main()` function that initializes the smart home system and provides the user with options to control devices.

2. Device Status:

- The system should include three main devices:
 - **Lights** (can be on or off)
 - **Thermostat** (adjustable between 15°C and 30°C)
 - **Security System** (can be armed or disarmed)
- Initialize each device with default values (`LightsOff`, `20°C` for thermostat, `disarmed` for the security system).

3. User Interaction:

Use a loop to allow the user to continuously interact with the system until they choose to exit. The user can:

- **Turn the lights on/off**
- **Set the thermostat temperature** between 15°C and 30°C
- **Arm/disarm** the security system
- **View system status**
- **Exit the program**

4. Input Validation:

Ensure valid input by rejecting values outside the temperature range and prompting the user for correct values when necessary.

5. Control Flow:

- Use `if/else` statements to handle user choices.
- Use a `switch` statement to handle device commands (lights, thermostat, security system).
- Implement a **FizzBuzz-like exercise**: if the temperature is a multiple of 3, print "Cool Day" along with the temperature when displaying status; if it's a multiple of 5, print "Warm Day"; if it's a multiple of both, print "Perfect Day."

6. String Interpolation and Operations:

- When displaying device status, use **string interpolation** to include the values of the lights, thermostat, and security system in user-friendly messages.
- Implement string operations to convert device status (e.g., "armed" to "ARMED") when showing system status.

7. **Final/Const Variables:**

- Use `const` for any constant values, such as maximum and minimum temperatures.

8. **Logical & Relational Operators:**

- Use these to validate user input when adjusting the thermostat and when turning the security system on/off.
- **Example Output:**

```
Welcome to the Smart Home System!
```

```
Please choose an action:
```

1. `Toggle` lights
2. `Set` thermostat temperature
3. Arm/disarm security system
4. View system status
5. `Exit`

```
Enter your choice: 1
```

```
Lights are now ON.
```

```
Enter your choice: 2
```

```
Set thermostat temperature (15°C – 30°C): 25
```

```
Thermostat set to 25°C.
```

```
Enter your choice: 4
```

```
System Status:
```

- Lights: ON
- Thermostat: 25°C – Warm Day
- Security System: DISARMED

```
Enter your choice: 5
```

```
Exiting the system. Goodbye!
```

Budget Management Application

Scenario:

You have been hired by a small business to create a simple **budget management application** using Dart. The application will help the business manage its monthly income and expenses. It will allow the user to input various income sources (like sales or investments) and expenses (like rent, salaries, and utilities), then calculate and display the **net profit** or **loss**.

Requirements:

1. Main Method:

The application should begin by welcoming the user and displaying a menu of options they can perform on the budget (e.g., add income, add expense, view summary, or exit).

2. Income and Expenses Management:

- The user can:
 - **Add income:** The program should prompt the user to enter the income description and amount.
 - **Add expense:** The program should prompt the user to enter the expense description and amount.
 - **View net profit or loss:** The program should calculate the total income, total expenses, and display the **net profit** (income - expenses) or **net loss** (if expenses exceed income).
 - **Exit:** The user should be able to exit the program.

3. Data Storage:

- Use **lists** to store multiple income and expense entries. Each entry should be stored as a **String** (description) and a **double** (amount).
- Example: `List<Map<String, double>> incomeList = []` and `List<Map<String, double>> expenseList = []`.

4. Input Validation:

- Ensure that the user enters valid numeric values for income and expense amounts (i.e., no negative amounts).
- Use **logical operators** to validate inputs.

5. Control Flow & Loops:

- Use **if/else statements** to handle menu options.
- Use a loop to keep asking the user for actions until they decide to exit the program.

6. String Interpolation & Operations:

- When displaying the summary of income, expenses, and net profit or loss, use **string interpolation** to clearly show amounts and messages like `"Your net profit is $5000"` or `"Your net loss is $200"`.

7. Arithmetic Operations:

- Perform calculations for total income, total expenses, and net profit or loss.

8. Exercise: Tax Calculation (Optional):

- Introduce a tax system where, at the end of the month, the business needs to pay **15%** of its net profit as tax.
- Use the ternary operator to ensure the tax is only calculated if the business is profitable. If there's a loss, no tax is calculated.
- After displaying net profit or loss, if the business made a profit, display the tax amount to be paid.

Example Output:

```
Welcome to the Budget Management Application!
Please choose an option:
1. Add income
2. Add expense
3. View summary
4. Exit

Enter your choice: 1
Enter income description: Sales
Enter income amount: 10000
Income added!

Enter your choice: 2
Enter expense description: Rent
Enter expense amount: 3000
Expense added!

Enter your choice: 3
Summary:
Total income: $10000
Total expenses: $3000
Net profit: $7000
Tax to be paid: $1050

Enter your choice: 4
Exiting the application. Goodbye!
```