Exercise 2 - Nospresso

Nospresso: Multi-Machine Management

***

Exercise 2 builds upon the program developed in Exercise 1. It introduces support for multiple Nospresso machines, each with its own stock levels and PIN for secure access. You must refactor your code from Exercise 1 to handle these new requirements. You must use knowledge from Module 1 to Module 6 **only**.

***

Goals

This exercise aims to:

- Introduce **arrays** to manage stock levels and PINs for multiple machines.
- Refactor the program into **methods** for modularity and reusability.
- Implement secure PIN management with a limited number of attempts.

Key Requirements

Machine Management

- The program must handle multiple Nospresso machines.
- Use a constant `nbMachines` to define the total number of machines (`val nbMachines = 5`). Each machine has a unique identifier (ID) ranging from 0 to $nbMachines - 1$.
- Each machine has its own stock levels for coffee (in grams), sugar (in grams), and milk (in millilitres), stored in arrays.
- Machines also have unique PINs stored in an array with the same default pin, initialise with 434343.

PIN Management

- Access to Admin mode on a machine requires entering its PIN.
- Users have a maximum of 3 attempts to enter the correct PIN. After 3 failed attempts, the program exits.
- Admins can update the PIN for any machine. The new PIN must be exactly 6 digits.

Stock Management

- Each machine has independent stock levels for coffee powder, sugar, and milk.
- Stock checks and updates must be specific to the selected machine.
- Admins can restock machines by adding specific quantities of ingredients.

Client and Admin Modes

- Client mode: Users can order drinks from a selected machine, after inputing the machine ID.
- Admin mode: Admins can restock ingredients and update machine PINs after selecting the machine ID and validating their credentials.

Methods

Refactor major operations into methods. The following method signatures must be implemented:

```
def validatePin(machineId: Int, machinePins: Array[String]): Boolean
def updatePin(machineId: Int, machinePins: Array[String]): Unit
def serveClient(machineId: Int, coffeeStocks: Array[Int],
                sugarStocks: Array[Int], milkStocks: Array[Int]): Boolean
def restockMachine(machineId: Int, coffeeStocks: Array[Int],
                   sugarStocks: Array[Int], milkStocks: Array[Int]): Unit
```

`validatePin`

Purpose: Validates the PIN for the selected machine.

Inputs:

- `machineId` - The ID of the machine to validate the PIN for (index in the arrays).
- `machinePins` - An array containing the PINs for all machines.

Output: `Boolean` - Returns `true` if the entered PIN matches the machine's stored PIN; `false` otherwise.

Description: This method prompts the user to enter the PIN for the given machine and validates it against the stored PIN in `machinePins(machineId)`. If the correct PIN is entered, the method returns `true`, else the method returns `false`.

`updatePin`

Purpose: Updates the PIN for the selected machine.

Inputs:

- `machineId` - The ID of the machine whose PIN needs to be updated (index in the arrays).
- `machinePins` - An array containing the PINs for all machines.

Output: `Unit` - The method modifies the `machinePins` array directly.

Description: This method allows an admin to update the PIN for a specific machine. The new PIN must be exactly 6 digits long. It prompts the user to enter the new PIN, validates the format, and updates the value in `machinePins(machineId)` if the input is valid. If the input is invalid, the user is re-prompted until a valid PIN is provided.

`serveClient`

Purpose: Processes a client transaction for the selected machine.
Inputs:

- `machineId` - The ID of the machine being used (index in the arrays).
- `coffeeStocks` - An array containing the coffee stock levels for all machines.
- `sugarStocks` - An array containing the sugar stock levels for all machines.
- `milkStocks` - An array containing the milk stock levels for all machines.

Output: `Boolean` - The method directly updates the stock arrays and simulates an interaction with the customer. It returns `false` if the transaction fails, or `true` if it succeeds.
Description: This method handles the client mode for a selected machine. It allows the client to:

- Choose a beverage.
- Customize the drink (e.g., sugar level, extra milk).
- Pay for the drink via Twint (simulated).

The method validates stock levels and deducts the required amounts from `coffeeStocks(machineId)`, `sugarStocks(machineId)`, and `milkStocks(machineId)`. If the stocks are insufficient, it displays an error message, does not process the transaction, and the program then prompts the user to select another machine.

`restockMachine`

Purpose: Restocks ingredients for the selected machine. **Inputs:**

- `machineId` - The ID of the machine to restock (index in the arrays).
- `coffeeStocks` - An array containing the coffee stock levels for all machines.
- `sugarStocks` - An array containing the sugar stock levels for all machines.
- `milkStocks` - An array containing the milk stock levels for all machines.

Output: `Unit` - The method modifies the stock arrays directly.
Description: This method allows an admin to restock the ingredients for a specific machine. It prompts the admin to enter the quantities of coffee, sugar, and milk to add. The inputs are validated to ensure they are positive values, and the respective stock levels in `coffeeStocks(machineId)`, `sugarStocks(machineId)`, and `milkStocks(machineId)` are updated accordingly.

Input Validation

- Validate inputs, including machine selection, drink selection and PIN.
- Re-prompt users until valid inputs are provided.

Scenarios

Scenario 1: Admin Mode - Restocking Ingredients

- Selected Machine: Machine 3
- Entered PIN: Correct
- Restocked: 100g coffee, 200g sugar, 0.5L milk

```
Selected machine (1-5) > 3
Enter PIN:
> 434343
Access granted to Machine 3.

Current stock levels:
    Coffee powder: 30g
    Sugar: 15g
    Milk: 0.2L

Enter quantities to add:
Coffee powder > 100
Sugar > 200
Milk > 0.5
Stocks updated successfully.
Returning to main menu...
```

Scenario 2: Failed PIN Attempts

- Selected Machine: Machine 1
- Entered PIN: Incorrect (3 times)

```
Selected machine (1-5) > 1
Enter PIN:
> 123456
Incorrect PIN. 2 attempts remaining.
> 654321
Incorrect PIN. 1 attempt remaining.
> 111111
Incorrect PIN. 0 attempts remaining.

Too many failed attempts. Exiting program.
```

Scenario 3: Admin Mode - Updating PIN

- Selected Machine: Machine 5
- Entered PIN: Correct
- Updated PIN: 123456

```
Selected machine (1-5) > 5
Enter PIN:
>  434343
```

```
Access granted.

Updating PIN for Machine 5.
Enter new 6-digit PIN > 123456
PIN updated successfully.
Returning to main menu...
```

Scenario 4: Failed PIN

- Selected Machine: Machine 1
- Entered PIN: Incorrect (3 times)

```
Selected machine (1-5) > 1
Enter PIN:
> 1234
Incorrect PIN. 2 attempts remaining.
> 654
Incorrect PIN. 1 attempt remaining.
> 1
Incorrect PIN. 0 attempts remaining.

Too many failed attempts. Exiting program.
```

Scenario 5: Admin Mode - Updating PIN Error

- Selected Machine: Machine 5
- Entered PIN: Correct
- Updated PIN: 1234

```
Selected machine (1-5) > 5
Enter PIN:
>  434343
Access granted.

Updating PIN for Machine 5.
Enter new 6-digit PIN > 1234
Enter new 6-digit PIN > 12
Enter new 6-digit PIN > 1
Enter new 6-digit PIN > 123567
PIN updated successfully.
Returning to main menu...
```