**1. Classes and Objects (Basic Practice)**

To build a cable operator system, we need to use classes to represent **Customers** and their related data (like plan, payment status). Here's how you can practice working with classes:

**Practice Example: Customer Class**

* **Task**: Create a Customer class with properties like Name, Plan, and PaymentStatus. Then, instantiate a Customer object and display its details.

**Steps**:

* + Create a class called Customer.
  + Add properties like Name, Plan, and PaymentStatus (boolean).
  + Create a method to display customer information.
  + Create an instance of Customer with real data (not hardcoded).

**Questions to Practice**:

* + Can you create a class for a Customer with properties for the subscription plan (Basic, Premium, VIP)?
  + How do you instantiate a Customer with a unique name and a plan?

**2. Constructors (Setting Values Dynamically)**

When creating an object, constructors allow you to set initial values for the properties. This is key to dynamically assigning values.

**Practice Example: Constructor for the Customer Class**

* **Task**: Practice using a constructor in the Customer class to initialize a new customer object with a name, plan, and payment status.

**Steps**:

* + In your Customer class, create a constructor that takes parameters for Name, Plan, and PaymentStatus.
  + Use the constructor to create a customer object dynamically.

**Questions to Practice**:

* + How can you initialize customer data (name, plan, and payment status) using a constructor?
  + How would you set a customer’s Plan to "VIP" and PaymentStatus to true when creating the object?

**3. Lists (Storing and Managing Multiple Customers)**

You’ll need to store multiple customers and work with them as a group. **Lists** are the perfect tool for this.

**Practice Example: Managing a List of Customer Objects**

* **Task**: Create a list of customers and add a few customer objects to it. Practice looping through the list to display each customer's data.

**Steps**:

* + Create a List<Customer> to hold all customers.
  + Add multiple customers to this list.
  + Loop through the list using foreach and display each customer's information.

**Questions to Practice**:

* + How do you add new customers to the list dynamically?
  + How can you iterate through a list to show all customers and their data?

**4. Modifying Objects (Change Customer’s Plan or Status)**

One core functionality is to **modify** customer data (like changing their plan or updating their payment status).

**Practice Example: Changing Customer Plan**

* **Task**: Write a method to change a customer’s subscription plan after finding the customer in the list by name.

**Steps**:

* + Create a method that accepts a customer name and a new plan as parameters.
  + Find the customer from the list.
  + Update the customer's Plan to the new one.

**Questions to Practice**:

* + How do you search for a customer by name in the list?
  + Once found, how can you change the customer's plan from "Basic" to "Premium"?

**5. Searching and Finding Specific Items**

In your system, users will need to search for a customer by their **unique name** or **ID**. You need to practice searching within a list or collection of customers.

**Practice Example: Search for a Customer by Name**

* **Task**: Create a method that searches for a customer by name and returns their details.

**Steps**:

* + Use List.Find() or a foreach loop to search for a customer by name.
  + Display the customer's details if found.

**Questions to Practice**:

* + How do you find a customer by their name using the List collection?
  + If the customer isn't found, what will your program do?

**6. Payment Status (Updating Payment)**

You’ll need to track whether a customer has paid, which can be done by **modifying the PaymentStatus**.

**Practice Example: Toggle Payment Status**

* **Task**: Write a method that **updates** the payment status of a customer (changing from unpaid to paid or vice versa).

**Steps**:

* + Create a method that takes the customer's name and the new payment status (true for paid, false for unpaid).
  + Update the customer’s PaymentStatus.

**Questions to Practice**:

* + How would you toggle the payment status of a customer from "not paid" to "paid" based on user input?
  + What kind of validation would you do to make sure that payment status is correctly updated?

**7. Menu-Driven Program (Interactivity)**

Your program will allow users to choose between multiple options, such as adding a customer, viewing all customers, and updating customer information. This is an example of a **menu-driven program**.

**Practice Example: Build a Basic Menu**

* **Task**: Create a simple menu system to interact with customer data, like adding customers, viewing customers, or quitting.

**Steps**:

* + Use a loop to continuously show the menu.
  + Based on user input, trigger the appropriate method (e.g., AddCustomer(), ViewCustomers()).

**Questions to Practice**:

* + How do you build a basic interactive menu using Console.ReadLine() for input?
  + How would you handle invalid choices by the user (e.g., if they type something other than the available options)?

**8. Removing a Customer (Optional)**

You might need to **remove customers** from your system. Practicing this will help you with **data removal** logic.

**Practice Example: Remove a Customer by Name**

* **Task**: Write a method that allows you to remove a customer by their name.

**Steps**:

* + Find the customer by name using List.Find() or a loop.
  + Remove the customer from the list using List.Remove().

**Questions to Practice**:

* + How would you safely remove a customer from the list after confirming they exist?
  + If the customer is not found, how would you handle the situation?

**9. Validating User Input**

You need to handle **user input validation** to ensure that the input is correct (e.g., the plan is either Basic, Premium, or VIP).

**Practice Example: Validate Plan Input**

**Task**: Create a method that asks the user to input a plan and validates that the input is either "Basic", "Premium", or "VIP".

**Steps**:

* + Use a while loop to repeatedly ask for input until the user provides a valid plan.
  + If the input is invalid, print an error message and prompt again.

**Questions to Practice**:

* + How can you use a while loop to ask for input until the user enters a valid plan?
  + How would you ensure the input is case-insensitive (e.g., "basic" should work)?

**10. Additional Challenge: Updating Multiple Customer Information**

Now, imagine you need to **update the plan and payment status for multiple customers** at once. You can use a combination of search and modification functions.

**Practice Example: Update Multiple Customers**

* **Task**: Allow the user to select multiple customers and change their plans or payment statuses.

**Steps**:

* + Allow the user to input multiple names.
  + Loop through those names and update the customer’s details accordingly.

**Questions to Practice**:

* + How can you efficiently update multiple customers in one go?
  + How would you ensure that you don’t accidentally change the wrong customer’s data?