**1. Understanding the Problem**

The goal is to create a system where a user can generate an invoice. The invoice contains customer details (name, email, address) and item details (number, name, price, quantity). The system must also handle the following:

* **Calculating total costs**: Based on items added to the invoice.
* **VAT**: Automatically calculating VAT (20% of the total cost).
* **Discounts**: Optionally applying a discount rate provided by the user or calculating a discount if the total cost exceeds 5000.
* **Due Date Management**: Automatically setting a due date for the invoice (7 days after the invoice date).

**2. Class Design**

To manage this functionality, the system is split into separate classes:

* **Customer**: Stores customer information (name, email, address). This helps encapsulate all customer-related data and can be extended if more customer-related features are needed in the future.
* **Item**: Represents individual items in the invoice. Each item has a price, quantity, and a calculated total price. This encapsulates the item’s properties and behavior (like calculating the total price).
* **Invoice**: Holds all details related to the invoice, such as the customer, items, VAT, and discount. The invoice is the central class because it manages the calculations (total cost, VAT, discount) and stores all relevant data for an invoice.
* **App**: The main driver class, responsible for interacting with the user. It collects input from the user, creates the Customer and Invoice objects, and prints the final output.

**3. Thought Process for Each Class**

**Customer Class**

* **Attributes**: Simple attributes for storing the customer’s information.
* **Constructor**: The constructor is designed to accept values for name, email, and address. A default constructor initializes these values to "Default". This is a common practice to ensure that objects can be created even if no specific input is provided (e.g., for testing purposes).
* **Methods**: Getter methods provide access to customer information, encapsulating the attributes.

**Item Class**

* **Attributes**: Each item has an ID, name, price, and quantity sold.
* **Constructor**: The constructor is designed to accept these values and initialize the item. A default constructor is provided for creating an item with placeholder values, useful when creating empty items in a collection.
* **Method (getTotalPrice)**: The method calculates the total price for the item (price \* quantitySold). This is an encapsulation of the logic needed to calculate the item’s total cost.

**Invoice Class**

* **Attributes**: The Invoice class holds more complex information. It manages:
  + **Customer**: A customer object is embedded within the invoice.
  + **Item List**: An array to hold the items included in the invoice. This array grows dynamically as more items are added.
  + **Financial Information**: VAT, discount, and total cost are calculated based on the items added.
  + **Due Date**: The due date is always 7 days after the invoice creation date, which is automatically handled.
* **Constructor**: The constructor accepts the customer object, number of items, and discount rate. It initializes the invoice date, due date, and then iteratively asks for item details (number, name, price, quantity).
  + **VAT and Discount Calculation**: VAT is calculated directly as 20% of the total cost, and the discount is calculated based on the total cost or discount rate provided by the user.
  + **Adding Items**: As items are added, the total cost is updated.
* **Setter Methods**: These methods allow modification of individual fields after the invoice has been created, providing flexibility for updates.
* **addItems()**: This method allows the user to add items to the invoice dynamically, with a check to ensure the number of items doesn't exceed 10. This enforces a simple business rule (max of 10 items per invoice).
* **calculateDiscount()**: This method checks if the total cost exceeds 5000 and applies a 10% discount if it does.
* **toString()**: This method builds a formatted string representing the invoice. It’s used to print all relevant information in a structured manner, making the output user-friendly. It iterates over the items and prints them, then shows the calculated total cost, VAT, and discount.

**App Class**

* **Input Handling**: The App class is responsible for collecting input from the user. It asks for the customer’s information (name, email, address), number of items, and discount rate.
* **Creating Objects**: The user input is used to create a Customer object and an Invoice object. The invoice constructor asks for details of each item (number, name, price, quantity), and then calculations are performed automatically.
* **Output**: After creating the invoice, the toString() method of the Invoice class is called to print all the invoice details, including the customer’s info, item details, and financial data.