The code consists of two classes, Bill and Driver, written in Java. It simulates a simple billing system for a food ordering scenario. Here's a detailed explanation:

**1. Class: Bill**

This class represents the billing functionality for ordered food items. It has the following key components:

**Attributes**

1. **orderedFood**:
   * A list to store the names of the food items ordered.
   * Type: ArrayList<String>.
2. **orderedQuantity**:
   * A list to store the quantity of each food item ordered.
   * Type: ArrayList<Integer>.
3. **totalCost**:
   * A variable to store the total cost of the ordered items.
   * Type: double.

**Constructor**

* **Bill()**:
  + Initializes the orderedFood and orderedQuantity lists.

**Methods**

1. **getTotal()**:
   * Returns the total cost of the ordered items.
   * Return Type: double.
2. **addOrder(String meal, int quantity, String[] dish, double[] cost)**:
   * Adds a new order to the bill.
   * **Parameters**:
     + meal: Name of the dish being ordered (e.g., "Idli").
     + quantity: Number of servings ordered.
     + dish: Array of available dishes.
     + cost: Array of costs corresponding to the dishes.
   * **Logic**:
     + The method adds the meal and its quantity to the respective lists.
     + It iterates through the dish array to find the matching dish name. When a match is found, it calculates the cost by multiplying the quantity with the corresponding cost in the cost array and adds it to totalCost.
3. **getOrder()**:
   * Displays all the items ordered and their respective quantities.
   * **Logic**:
     + Iterates through the orderedFood and orderedQuantity lists and prints each item along with its quantity.

**2. Class: Driver**

This is the main class that drives the program. It contains the main() method where the execution begins.

**Steps in main():**

1. **Define available dishes and their costs**:
   * An array dish contains the names of available dishes (e.g., "Idli", "Dosa").
   * An array cost contains the respective prices for these dishes (e.g., 20.5 for "Idli").
2. **Create a Bill object**:
   * An instance of the Bill class is created using the default constructor.
3. **Add orders to the bill**:
   * addOrder() is called to add items like "Idli", "Dosa", and "Tea" with respective quantities.
4. **Display the ordered items and total cost**:
   * getOrder() is called to print the ordered items and their quantities.
   * getTotal() is called to print the total cost of the order.

**Output of the Code**

When executed, the program produces the following output:

Copy code

Idli 3

Dosa 2

Tea 2

154.7

**Explanation of the Output:**

1. **Order Details**:
   * "Idli" ordered: 3 servings (Cost = 3 × 20.5 = 61.5).
   * "Dosa" ordered: 2 servings (Cost = 2 × 30.6 = 61.2).
   * "Tea" ordered: 2 servings (Cost = 2 × 15.7 = 31.4).
2. **Total Cost**:
   * Sum of costs = 61.5 + 61.2 + 31.4 = 154.7.

**Key Concepts Illustrated in the Code**

1. **Encapsulation**:
   * The Bill class encapsulates the billing logic, ensuring data (like orderedFood and totalCost) is accessed and modified only through methods.
2. **Dynamic Arrays (ArrayList)**:
   * ArrayList is used for dynamic storage of ordered food items and quantities.
3. **Control Structures**:
   * A for loop is used to match the ordered dish with its corresponding cost.
4. **Object-Oriented Programming (OOP)**:
   * Objects of Bill class manage individual bills.
   * Reusable methods like addOrder() and getOrder() make the code modular.

This program is a basic representation of how billing systems can be implemented programmatically.