Assignment 1

Software Construction & Development

Ali Iqbal

2377-2022

10/19/2024

Online Food Delivery System

**Part 1:**

**Object-Oriented Design and Key Construction Decisions**

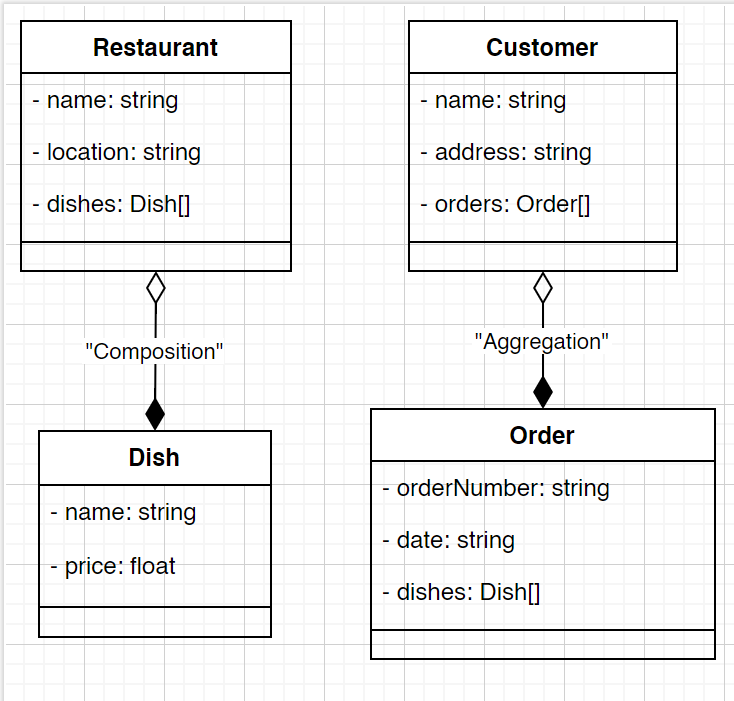
#### **Problem Statement:**

The Online Food Delivery System manages restaurant menus and customer orders. Restaurants can add dishes to their menu, customers can place orders, and the system ensures the delivery process is managed efficiently.

#### **Classes:**

1. **Restaurant**
   * Attributes: restaurantID, name, menu (list of Dish objects)
   * Methods: addDish(), removeDish(), updateDish()
2. **Dish**
   * Attributes: dishID, name, price, isAvailable
   * Methods: changeAvailability()
3. **Customer**
   * Attributes: customerID, name, orderHistory (list of Order objects)
   * Methods: placeOrder(), viewOrderStatus(), cancelOrder()
4. **Order**
   * Attributes: orderID, customer, restaurant, dishes (list of Dish), status
   * Methods: updateStatus(), addDish(), removeDish()

#### **Classes Digram:**



#### **Key Construction Decisions:**

* **Encapsulation**: Private attributes are used for orderStatus and menu, ensuring these can only be modified through methods, maintaining system integrity.
* **Polymorphism**: In the future, we could implement polymorphism with different customer types, like a PremiumCustomer who gets faster deliveries.
* **Abstraction**: By abstracting the User class, we allow for easy extension to add more user types, like Admin.

### **Part 2:**

### **Algorithm Design and Complexity Analysis**

#### **Feature: Assigning the Nearest Restaurant**

We need to find the closest restaurant to the customer’s location. The following algorithm is used to calculate the nearest restaurant based on Euclidean distance.

#### **Algorithm:**

def findNearestRestaurant(customerLocation, restaurantList):

nearestRestaurant = None

minDistance = float('inf')

for restaurant in restaurantList:

distance = ((restaurant.x - customerLocation.x) \*\* 2 + (restaurant.y - customerLocation.y) \*\* 2) \*\* 0.5

if distance < minDistance:

minDistance = distance

nearestRestaurant = restaurant

return nearestRestaurant

#### **Time Complexity:**

* **O(n)** where n is the number of restaurants. Each restaurant must be checked once, resulting in linear time complexity.

#### **Space Complexity:**

* **O(1)** because we only use a few extra variables regardless of the input size.

#### 

#### 

#### **Trade-offs and Optimizations:**

* While the algorithm is simple and efficient for a small number of restaurants, a spatial partitioning algorithm like a k-d tree would be a better choice for larger datasets, reducing search time to **O(log n)**.

### **Part 3:**

## **Part 3: Better Use of Programming Language and Design**

### **Programming Language: Python**

Python was chosen for the implementation of the Online Food Delivery System due to its dynamic nature, ease of use, and rich set of libraries that accelerate development. In this section, we’ll discuss specific features of Python that influenced the development process and suggest how the programming language could be used more effectively in future iterations.

### **Language Features**

1. **Dynamic Typing**:  
   One of Python’s core features is dynamic typing, which allows developers to define variables without specifying their types. This flexibility makes Python well-suited for rapid prototyping and iterative development. For instance, in the Online Food Delivery System, it was easy to quickly implement classes like Restaurant, Customer, and Order without having to define strict data types for every variable. This leads to faster coding, especially in the initial stages of development, as developers can focus on functionality rather than enforcing type constraints.  
   However, while dynamic typing improves development speed, it can also introduce errors that are caught only at runtime, such as type mismatches or unexpected variable behaviors. This can be mitigated using type hints, a feature introduced in Python 3, which allows developers to specify expected data types without sacrificing Python’s flexibility.
2. **Built-in Libraries**:  
   Python's extensive standard library is another reason it is favored for such projects. For example, in this system, the math library was used to calculate the Euclidean distance between a customer and nearby restaurants, which significantly reduced the amount of custom code needed. Similarly, Python’s collections library offers built-in data structures like defaultdict and OrderedDict, making it easier to manage orders, menus, and customer histories.  
   The availability of such libraries not only accelerates development but also ensures that code is efficient and reliable since these libraries are thoroughly tested. This feature reduces the need for building custom utilities from scratch, saving time and effort, and improving overall code quality.
3. **Memory Management**:  
   Python’s automatic memory management, handled by its built-in garbage collector, simplifies the development process. When creating instances of classes like Restaurant or Order, developers do not need to manually allocate or deallocate memory. This is particularly useful in applications like the Online Food Delivery System, where numerous objects are created and discarded dynamically (e.g., new orders, restaurants being added or removed).  
   By handling memory deallocation automatically, Python prevents memory leaks and reduces the complexity involved in memory management, especially for junior developers or in projects where memory efficiency is not the primary concern. This leads to cleaner and more maintainable code, especially in projects with frequent object creation and deletion.

### **Impact of Language Design on Development**

1. **Performance vs. Readability**:  
   One of Python’s most appreciated qualities is its clear, readable syntax, which is closer to natural language compared to other programming languages. This results in faster development cycles, as code is easier to write, read, and debug. For the Online Food Delivery System, Python’s clean syntax allowed rapid implementation of features like order tracking, menu management, and distance calculation without unnecessary complexity.  
   However, Python’s emphasis on readability comes at a cost: performance. Python is an interpreted language, which makes it slower than compiled languages like C++ or Java. For small to medium-sized projects, this is negligible, but as the system scales (e.g., with hundreds of restaurants or thousands of orders), performance bottlenecks may arise. For example, the algorithm used to find the nearest restaurant could be significantly faster if implemented in a lower-level language like C or by using Python’s NumPy for vectorized operations.
2. **Concurrency**:  
   Python’s Global Interpreter Lock (GIL) can be a limiting factor when it comes to multi-threaded applications. While Python allows for multi-threading, the GIL ensures that only one thread executes Python bytecode at a time, which can prevent full CPU utilization in multi-core systems. This becomes an issue if the system grows and requires simultaneous processing of multiple orders and deliveries.  
   That said, Python’s **asyncio** module provides an efficient way to handle concurrency in I/O-bound applications like the Online Food Delivery System. Using **asyncio**, tasks like handling multiple order requests or fetching restaurant data concurrently can be processed without blocking the main thread, improving responsiveness without the overhead of traditional multi-threading.

### **Suggestions for Better Use**

1. **Asynchronous Programming**:  
   The current version of the Online Food Delivery System is a simple implementation that does not handle concurrent orders. However, as the system scales, handling multiple customer requests at once becomes critical. For instance, many customers could be browsing menus or placing orders simultaneously. To handle this efficiently, Python’s asyncio module can be leveraged. By making functions asynchronous (e.g., placing an order, updating order status), the system can handle multiple requests in a non-blocking manner, significantly improving performance and responsiveness.  
   For example, instead of waiting for each step in the order process (e.g., confirming availability, notifying the restaurant) to complete before handling the next task, the system could handle other tasks in the meantime. This would lead to a more responsive user experience, especially under heavy load.
2. **Using External Libraries for Performance Improvements**:  
   As mentioned earlier, while Python is great for development speed and flexibility, it lacks the performance of lower-level languages when it comes to intensive computation tasks. For certain parts of the system, such as finding the nearest restaurant or processing large amounts of order data, Python's performance can be improved by using external libraries like **NumPy** or **C extensions**. NumPy allows for fast array-based calculations, while C extensions can be used to write performance-critical sections of the code in C, which can then be called from Python.  
   By using these optimizations, the system could handle larger datasets (e.g., thousands of restaurants or customer orders) more efficiently. This would be especially important for geographic features like distance calculation, where performance improvements could greatly reduce response times for finding nearby restaurants or managing large menus.