Python fundamentals day 50

Today's Agenda

- Design principals of object orientation

Design principals of object orientation

The four major principles of object orientation are:

- Encapsulation
- Polymorphism
- Inheritance
- Abstraction

An object oriented program is based on classes and there exists a collection of interacting objects, as opposed to the conventional model, in which a program consists of functions and routines. In OOP, each object can receive messages, process data, and send messages to other objects.

Let us consider the same example of food delivery that we had considered earlier and develop it

Customer

State:

- Name
- Email
- Phone_num
- Address

Behaviour:

- Sel_restaurant()
- Sel_item()
- Sel_address()
- Place_order()
- Cancel_order()
- Make_payment()
- Avail_discount()

State:

- Name
- Address
- Ratings
- Item

Behaviour:

- Confirm_order()
- Add_item()
- Del_item()

Delivery Agent

State:

- Name
- Ratings
- Phone_num

Behaviour:

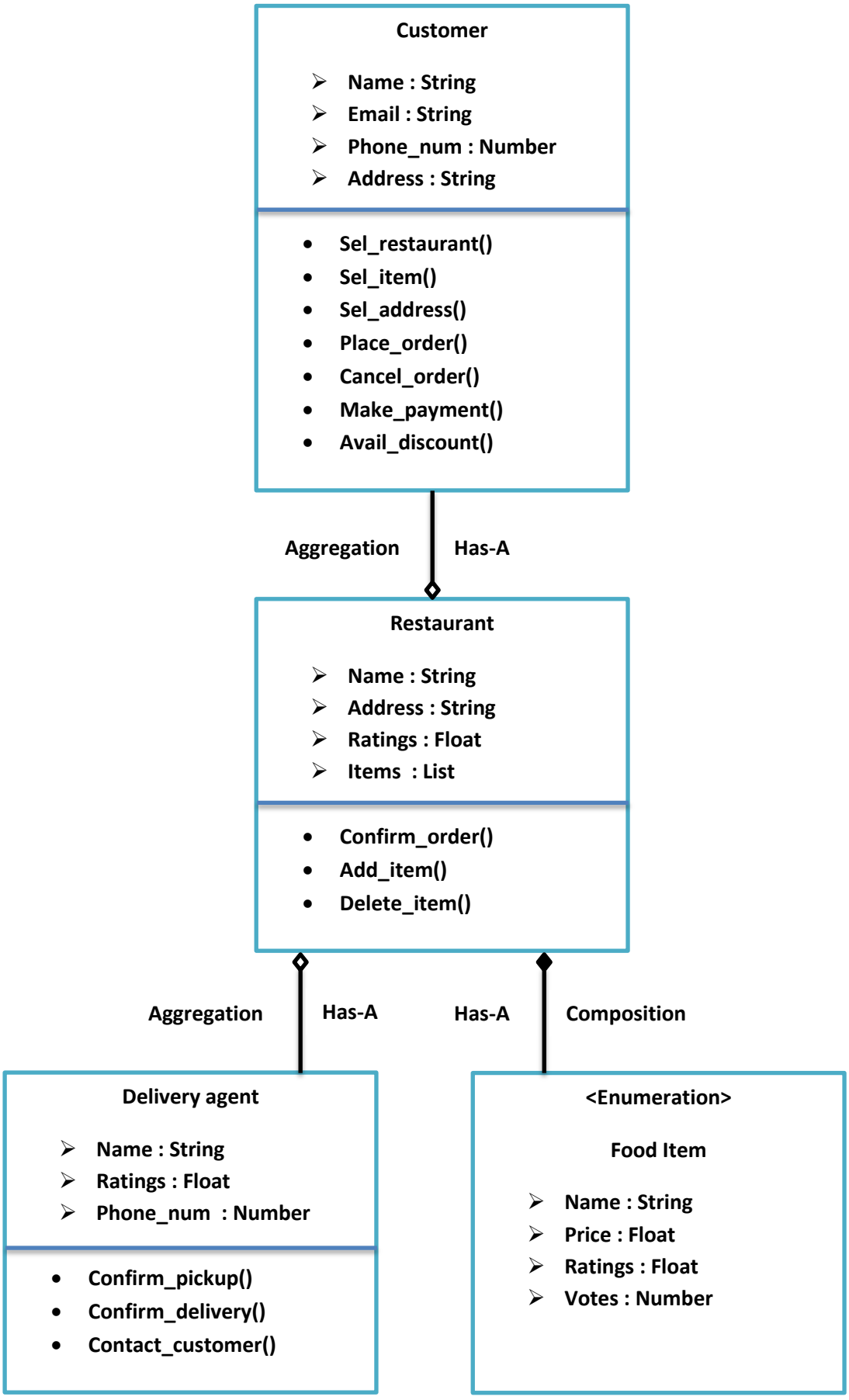
- Confirm_pickup()
- Confirm_delivery()
- Contact_customer()

State:

- Name
- Price
- Ratings
- Votes

Food Item

Now all these objects should be having a connection between them. Let us see


Customer
Name : String
Email : String
Phone_num : Number
Address : String
Sel_restaurant()
Sel_item()
Sel_address()
Place_order()
Cancel_order()
Make_payment()
Avail_discount()
Aggregation
Has-A
Restaurant
Name : String
Address : String
Ratings : Float
Items : List
Confirm_order()
Add_item()
Delete_item()
Aggregation
Has-A
Has-A
Composition
Delivery agent
Name : String
Ratings : Float
Phone_num : Number
Confirm_pickup()
Confirm_delivery()
Contact_customer()
<Enumeration>
Food Item
Name : String
Price : Float
Ratings : Float
Votes : Number

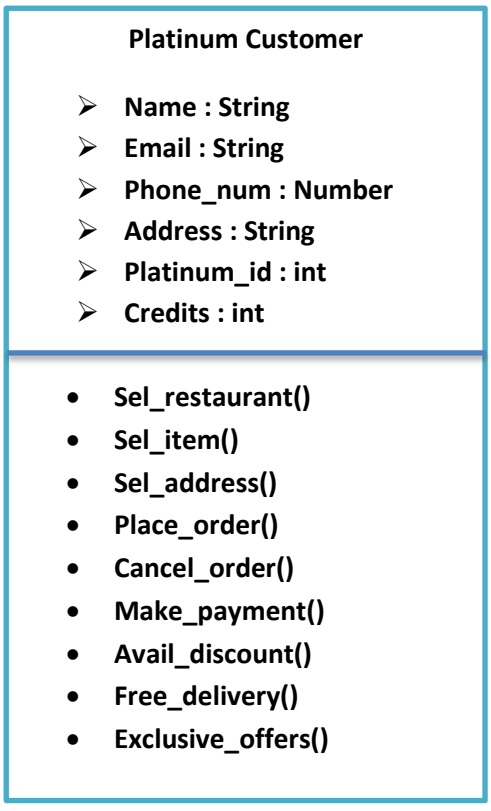
From the above UML we have derived the relation between all the objects. Now before going further let us see what is aggregation and composition

Composition and aggregation are specialised form of Association. Whereas Association is a relationship between two classes without any rules.

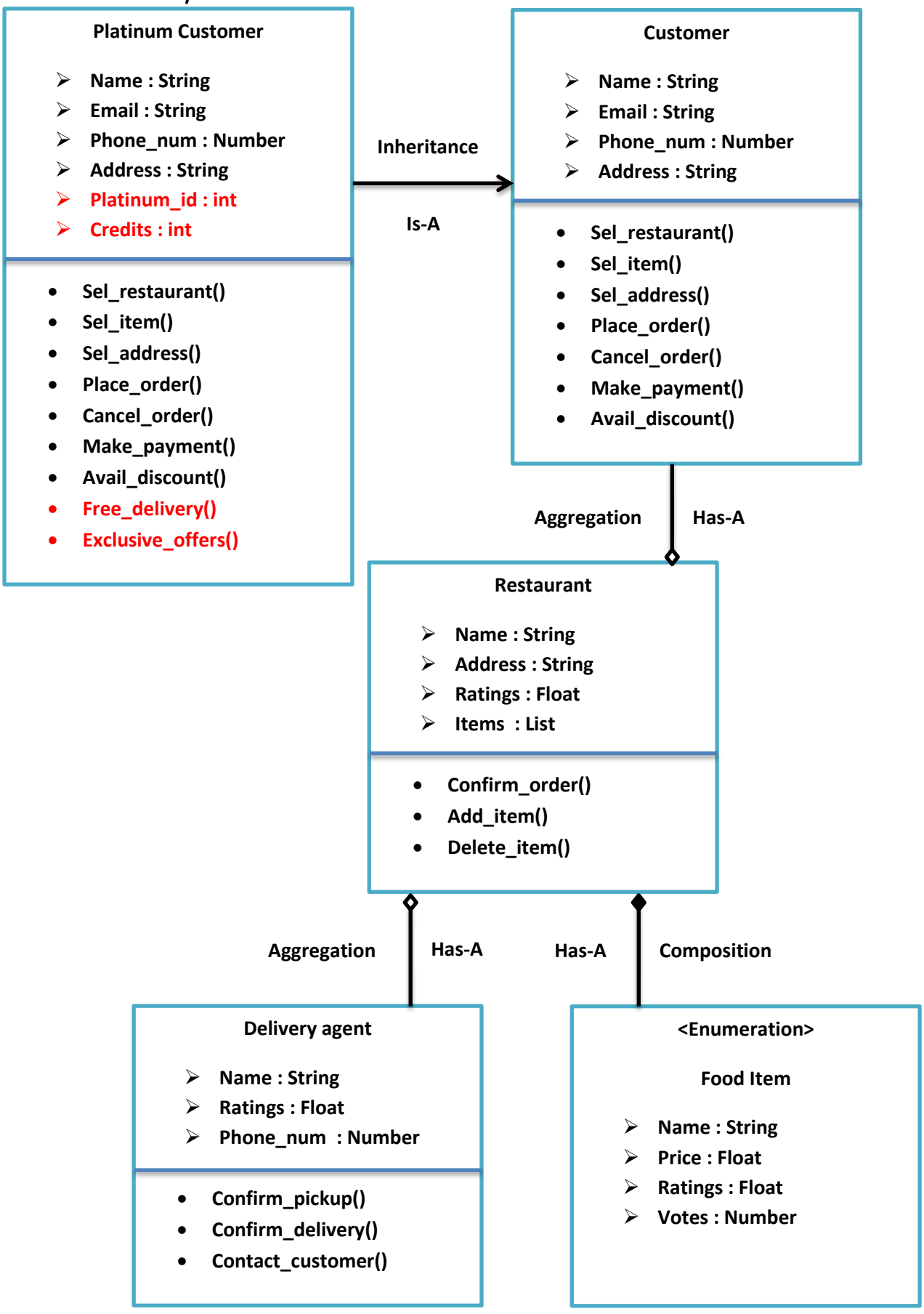
Composition: One class is container and other class is content and if you delete the container object then all of its contents objects are also deleted.

Aggregation: Aggregation is a weak form of composition. If you delete the container object contents objects can live without container object.

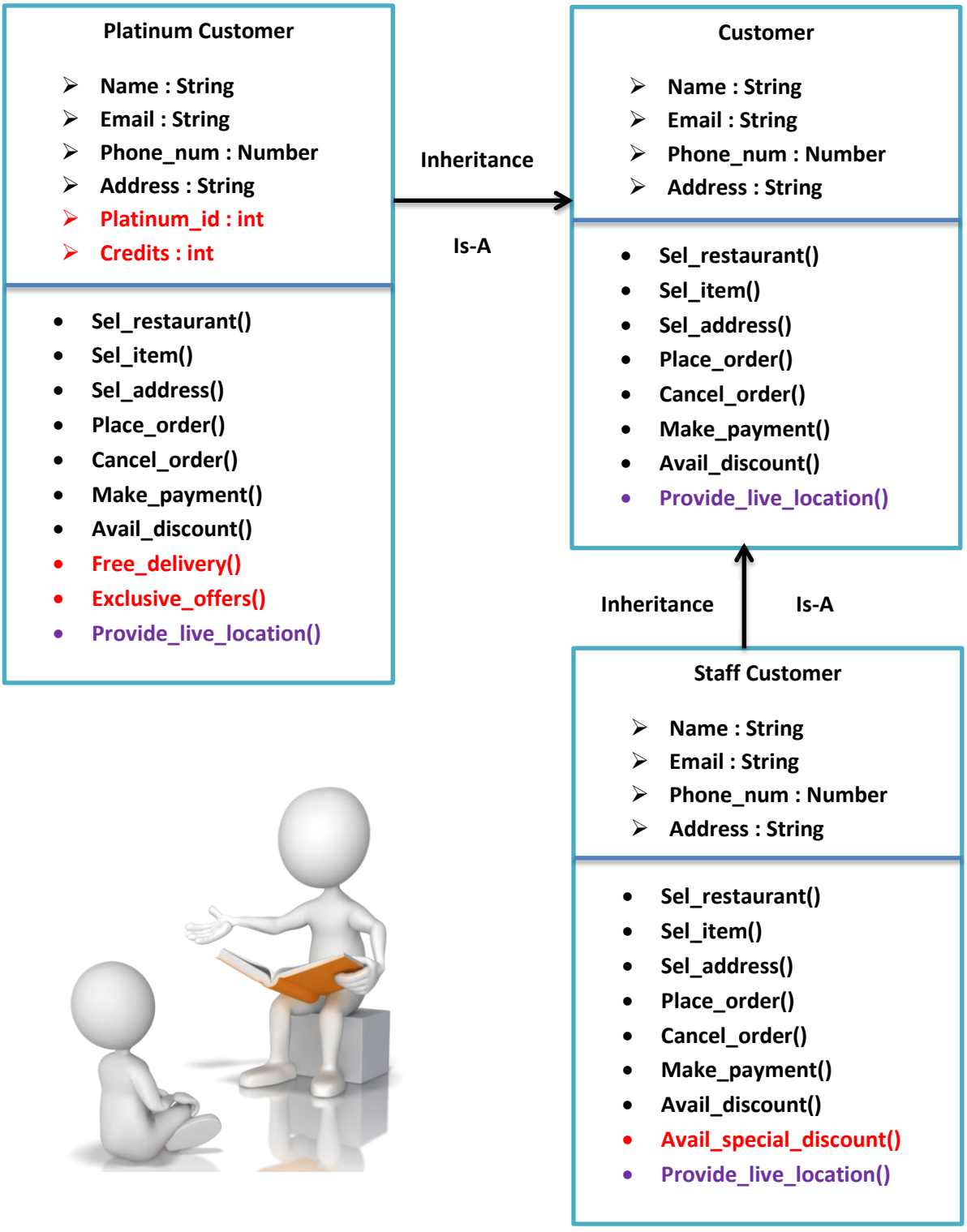
Now what if we want to add another feature and upgrade. It wouldn't be efficient way to start from the scratch. Therefore we have a feature where we can re-use the code and modify. Let us consider following as the new feature



Let us try to establish a relation between all the features



Now again we want to upgrade a feature where staff members are benefitted



Note: Any changes made in parent class reflect in child class as well.