CSC110 Lecture 29: Object-Oriented Modelling

David Liu and Tom Fairgrieve, Department of Computer Science

*Navigation tip for web slides: press **?** to see keyboard navigation controls.*

Announcements and Today's Plan

Assignment 4 and Term Test 3 done!!

Announcements

- Please complete the PythonTA Survey 2
  - Due December 8

Story so far

Representing data	**Operating on data**
Built-in data types	Operators
Data classes (bundles of data)	Functions

Python's general classes give us the ability to organize code into entities that specify both how data is represented (**attributes**) and how to operate on that data (**methods**).

In this lecture, you'll learn to:

- Model a real-world entity using a Python class
- Model a real-world problem domain using a collection of Python classes that interact with each other

Modelling Food Delivery

This week, we're going to build up a computational model of a food delivery system.

Problem domain description

Seeing the proliferation of various food delivery apps, you have decided to create a food and grocery delivery app that focuses on students. Your app will allow student users to order groceries and meals from local grocery stores and restaurants. The deliveries will be made by couriers to deliver these groceries and meals—and you'll need to pay the couriers, of course!

When creating a model of a large system, it's easiest to first design classes to represent individual entities in that system.

How do we identify "individual entities"?

Identifying entities: Strategy 1

1. Identify different roles that people/groups play in the domain.

> *Seeing the proliferation of various food delivery apps, you have decided to create a food and grocery delivery app that focuses on students. Your app will allow student users to order groceries and meals from local grocery stores and restaurants. The deliveries will be made by couriers to deliver these groceries and meals—and you'll need to pay the couriers, of course!*

Identifying entities: Strategy 1

1. Identify different roles that people/groups play in the domain.

> *Seeing the proliferation of various food delivery apps, you have decided to create a food and grocery delivery app that focuses on students. Your app will allow* ***student users*** *to order groceries and meals from* ***local grocery stores and restaurants****. The deliveries will be made by* ***couriers*** *to deliver these groceries and meals—and you'll need to pay the couriers, of course!*

Identifying entities: Strategy 2

2. Identify a bundle of data that makes sense as a logical unit.

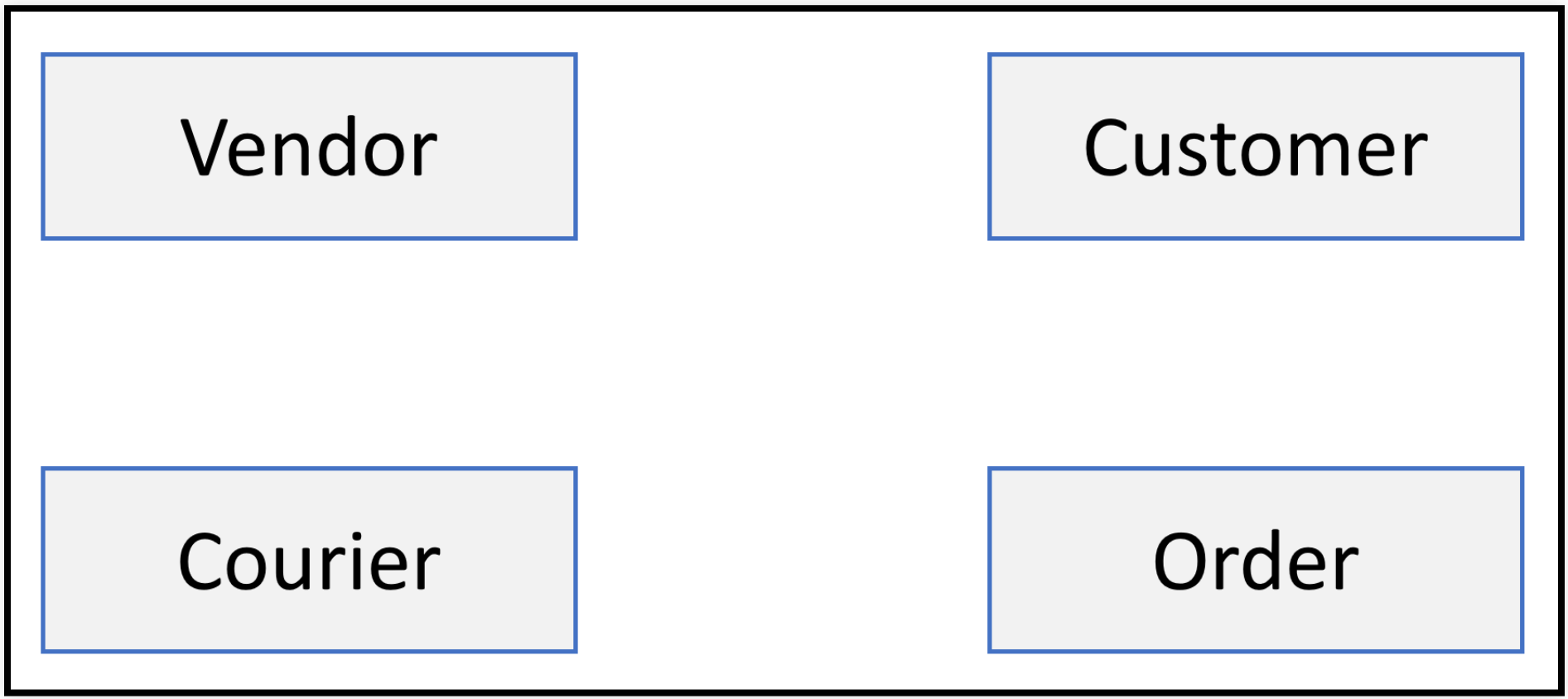
> *Seeing the proliferation of various food delivery apps, you have decided to create a food and grocery delivery app that focuses on students. Your app will allow student users to order groceries and meals from local grocery stores and restaurants. The deliveries will be made by couriers to deliver these groceries and meals—and you'll need to pay the couriers, of course!*

Identifying entities: Strategy 2

2. Identify a bundle of data that makes sense as a logical unit.

> *Seeing the proliferation of various food delivery apps, you have decided to create a food and grocery delivery app that focuses on students. Your app will allow student users to* ***order groceries and meals*** *from local grocery stores and restaurants. The deliveries will be made by couriers to deliver these groceries and meals—and you'll need to pay the couriers, of course!*

Identifying entities: putting it together



To start, we create a data class and focus on identifying **attributes** for each class.

```
@dataclass
class Vendor:
    """A vendor that sells groceries or meals."""
```

```
    name: str
    address: str
    menu: dict[str, float]
    location: tuple[float, float]  # (lat, lon) coordinates
```

To start, we create a data class and focus on identifying **attributes** for each class.

```
@dataclass
class Customer:
    """A person who orders food."""
```

```
name: str
location: tuple[float, float]
```

Attribute choice is a design decision

As the people doing the modelling, it is our responsibility to choose attributes for our classes.

Considerations:

1. What information is necessary for our application's functions?
2. What information will our users be willing to share?
3. What information may be useful for future extensions?
4. Does keeping track of certain information make our code more complex or slower?

Though we want to make these decisions carefully, it is also possible to change them over time (but must be careful when doing so!).

The `Order` data class

An `order` must keep track of which customer and vendor the order is placed for, and which items were ordered. We'll also keep track of when the order was placed.

```
@dataclass
class Order:
    customer: Customer
    vendor: Vendor
    food_items: dict[str, int]  # map food name to quantity
    start_time: datetime.datetime
```

We also need a way of assigning orders to a particular courier, and tracking whether the order has been delivered.

```
@dataclass
class Order:
    customer: Customer
    vendor: Vendor
    food_items: dict[str, int]  # map food name to quantity
    start_time: datetime.datetime
```

```
    courier: Optional[Courier] = None
    end_time: Optional[datetime.datetime] = None
```

`courier` and `end_time` have a default value of `None`.

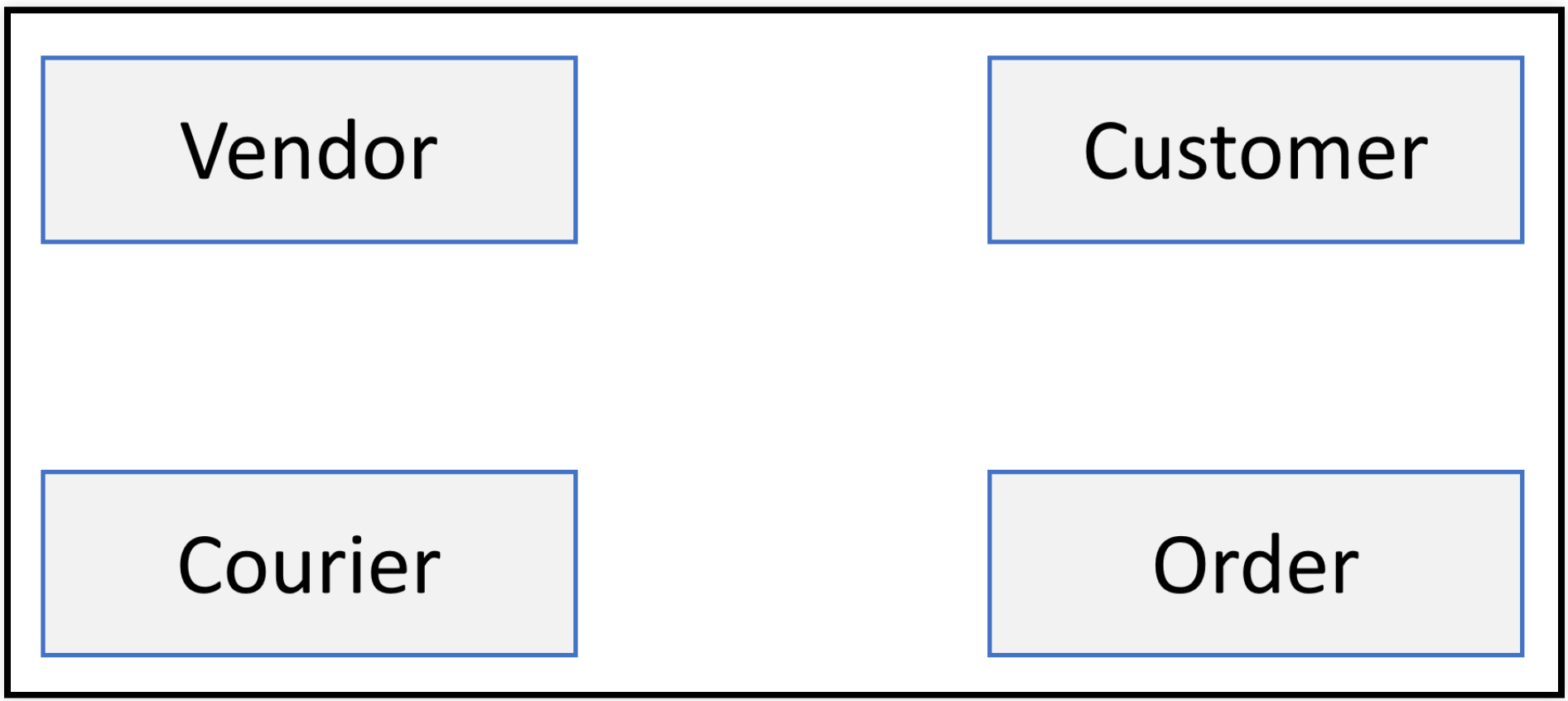
`Optional[T]` means "an object of type `T`, or `None`".

Class composition

Example of **class composition**: `Order` has attributes whose types are other classes we've created.

Inheritance indicates an "is-a" relationship, e.g. "`Stack1` is a `Stack`".

Composition indicates a "has-a" relationship, e.g. "An `Order` has a `Customer`".

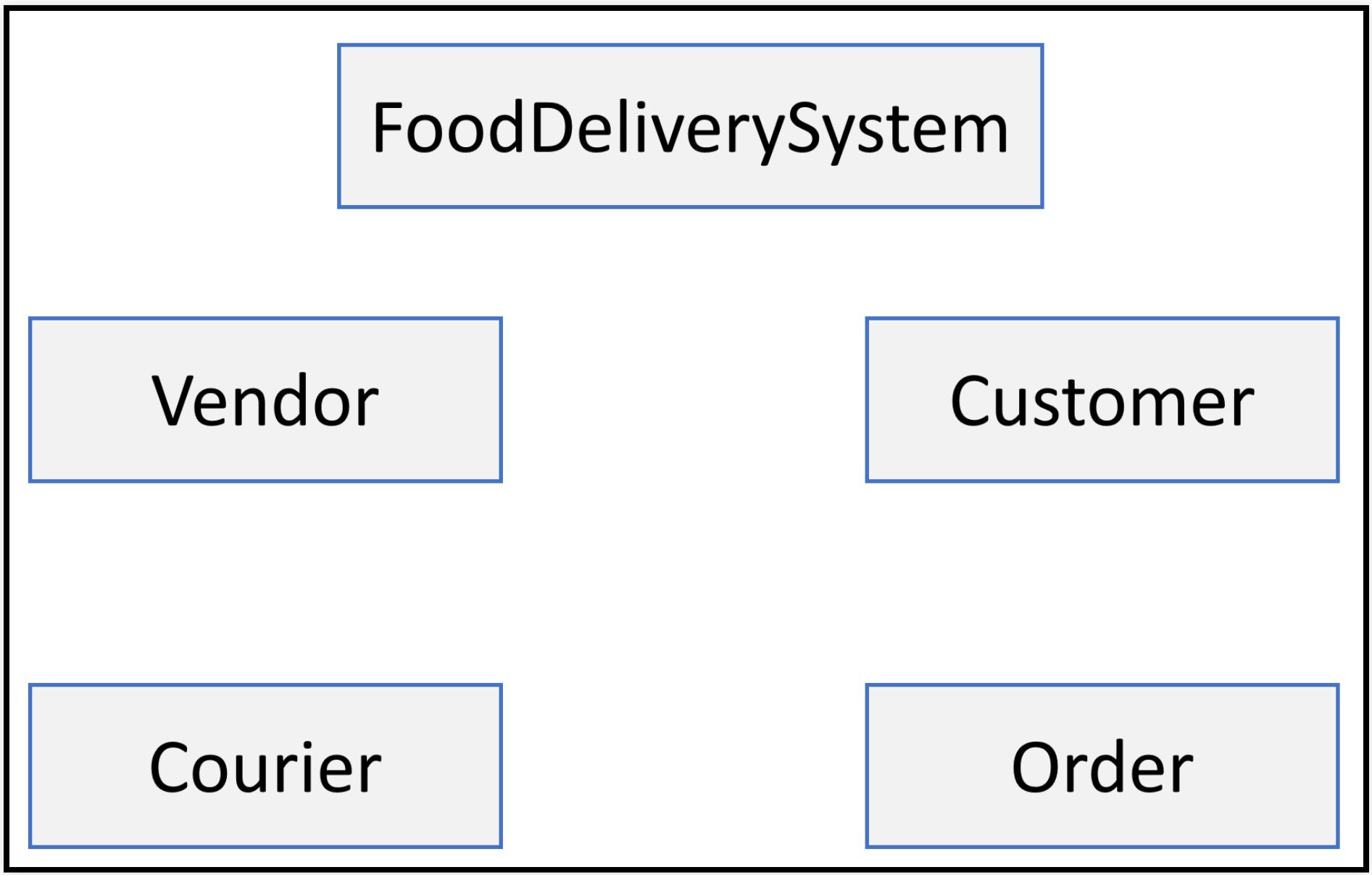

Vendor
Customer
Courier
Order

Exercise 1: Designing the Courier data class

Managing the entities

Question: how does our program keep track of all vendors/customers/couriers/orders?

Idea: create a "manager" class whose purpose is to keep track of all entities in the system.


FoodDeliverySystem
Vendor
Customer
Courier
Order

```python
class FoodDeliverySystem:
    """A system that maintains all entities
    (vendors, customers, couriers, and orders).
    """
    _vendors: dict[str, Vendor]
    _customers: dict[str, Customer]
    _couriers: dict[str, Courier]
    _orders: list[Order]
```

`_vendors`, `_customers`, and `_couriers` map name to object. (Support efficient lookup of entities later.)

`_orders` is just a list of `Order`s.

Why private attributes?

Communicating intent: the `FoodDeliverySystem` should be responsible for keeping track of the entities directly, and external code doesn't need to know about how.

Instead, external code should call methods to update the state of the system.

Handling mutation

`FoodDeliverySystem` is responsible for handling all mutating changes in the system, such as:

- Add a new vendor/customer/courier
- Remove an existing vendor/customer/courier
- Create a new order
- Mark an order as completed

Exercise 2: Developing the `FoodDeliverySystem` class

Managing orders

Now consider what happens when a customer places an order.

1. First, a new order gets created.
2. The order is assigned a courier.
3. The courier makes the delivery, and the order is complete.

Placing an order

```
class FoodDeliverySystem:
    def place_order(self, order: Order) -> bool:
        """Add an order to this system.

        Do NOT add the order if no couriers are available
        (i.e., are already assigned orders).

        - If a courier is available, add the order and assi
          it a courier, and return True.
        - Otherwise, do not add the order, and return False
        """
```

To PyCharm!

Completing an order

```
class FoodDeliverySystem:
    def complete_order(self, order: Order,
                       timestamp: datetime.datetime) -> Non
        """Record that the given order has been delivered
        successfully at the given timestamp.

        Make the courier who was assigned this order availa
        to take a new order.

        Preconditions:
            - order in self._orders
            - order.end_time is None
            - order.start_time < timestamp
        """
```

Summary

Today, you learned to:

- Model a real-world entity using a Python class
- Model a real-world problem domain using a collection of Python classes that interact with each other

Homework

- Readings:
  - From today: 11.1, 11.2, 11.3
  - For next class: 11.4, 11.5
- Please review posted code before lecture!
- Please complete the PythonTA Survey 2


DONE WITH CSC110
STUDENT
DONE WITH CSC110
FINAL EXAM
STUDENT
imgflip.com