Quiz 3 Review

Welcome back to CS 2100!

Prof. Rasika Bhalerao

Topics revisited from preious quizzes

- Using Objects (from Quiz 1)
  - `None`
  - State and aliasing
- Testing (from Quiz 1)
  - `setUp(self)` and `setUpClass(cls)`
  - Identifying test cases
- Sets and dictionaries (from Quiz 2)
  - Iterating through a dictionary
  - Rules about duplication
  - Set operations

New review topics

- Abstract methods
  - `@abstractmethod`
  - Rules for instantiation
- Inheritance
  - What subclasses inherit
  - Overwriting inherited methods
  - `super().`
- UML diagrams
  - Name, attributes, methods
  - Abstract class / method
  - Public / private
  - Relationships between classes
- Privacy
  - What type of info is shared?
  - Who is the subject? Sender? Potential recipients?
  - What principles govern the collection and transmission?
  - Determining tradeoffs

Abstract methods

We cannot instantiate a class that has an `@abstractmethod` (even inherited).

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def get_area(self) -> float:
        pass

    @abstractmethod
    def get_perimeter(self) -> float:
        pass

shape = Shape()  # TypeError
```

- Doing so will raise a `TypeError`.
- To instantiate a class that inherits a `@abstractmethod`, overwrite it with a **concrete** (non-abstract) method.

Poll: What happens?

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def make_sound(self) -> None:
        pass

class Dog(Animal):
    pass

dog = Dog()
```

1. A `Dog` object is created successfully
2. A `TypeError` is raised because `Dog` doesn't implement `make_sound()`
3. `Dog.make_sound()` returns `None`
4. A warning is printed but `dog` is created

Poll: Which is TRUE?

1. A class can only have one abstract method
2. You can instantiate an `ABC` if it has no abstract methods
3. Abstract methods cannot have parameters
4. `ABC` classes cannot have concrete (non-abstract) methods

Poll: What is output?

```python
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start(self) -> str:
        return "Starting..."

class Car(Vehicle):
    def start(self) -> str:
        return super().start() + " car engine"

c = Car()
print(c.start())
```

A) A TypeError is raised

B) "Starting..."

C) "Starting... car engine"

D) "car engine"

Poll: How many abstract methods must a concrete (non-abstract) subclass overwrite?

1. At least one
2. Exactly one
3. All of them
4. None, they are optional

Inheritance

1. A *subclass* is a more specific version of a *superclass*.
2. The subclass *inherits* all methods and attributes from the superclass.
   -> Except those named with two underscores
3. The subclass can overwrite any inherited methods / attributes.
4. The subclass can add more methods / attributes.

Calling a superclass's method (or constructor)

```python
class Cat:
    def __init__(self, name: str):
        self.name = name
        self.food = ['tuna', 'chicken']

    def knead(self) -> None:
        print('Kneading')

    def eat(self, food: str) -> None:
        if food in self.food:
            print(f'Eating {food}')

class Lion(Cat):
    def __init__(self, name: str):
        super().__init__(name)
        self.food += ['zebra']

    def roar(self) -> None:
        print('Roaring')
```

- `eat()` method inherited from `Cat` works by default in `Lion`
- `self.food` is defined in `Cat`'s constructor, so we overwrite it with a new constructor in `Lion`...
  - one that executes `Cat`'s constructor first, and then adds `'zebra'` to `self.food` (which is inherited)

Poll: What is output?

```python
class Vehicle:
    def __init__(self, brand: str):
        self.brand = brand

class Car(Vehicle):
    def __init__(
      self, brand: str, model: str):
        self.model = model

c = Car("Toyota", "Camry")
print(c.brand)
```

1. `Toyota`
2. `None`
3. An `AttributeError` is raised
4. `Camry`

Poll: Which method call would return `"Rex makes a sound"`?

```python
class Animal:
    def __init__(self, name: str):
        self.name = name

    def speak(self) -> str:
        return f"{self.name} makes a sound"

class Dog(Animal):
    def __init__(self, name: str, breed: str):
        super().__init__(name)
        self.breed = breed

    def speak(self) -> str:
        return f"{self.name} barks"

d = Dog("Rex", "Labrador")
```

A) `d.speak()`

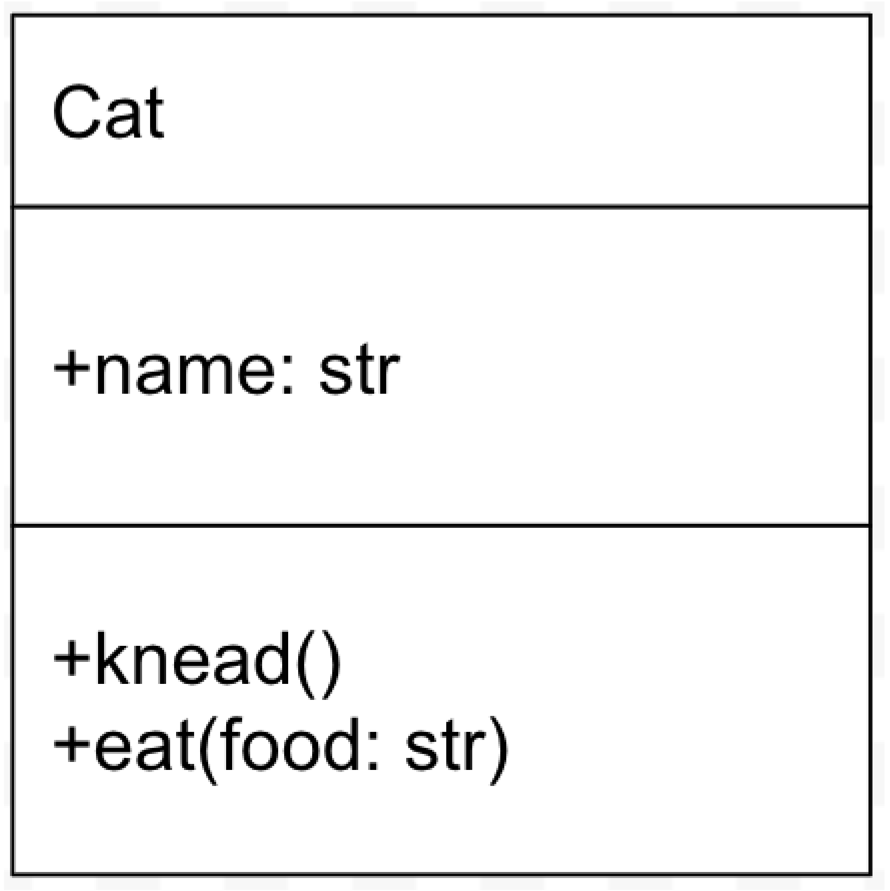
B) `super().speak()`

C) `Animal.speak(d)`

D) `d.Animal.speak()`

UML (Unified Modeling Language) Diagrams

A UML diagram visually shows us the classes and their relationships in a program.



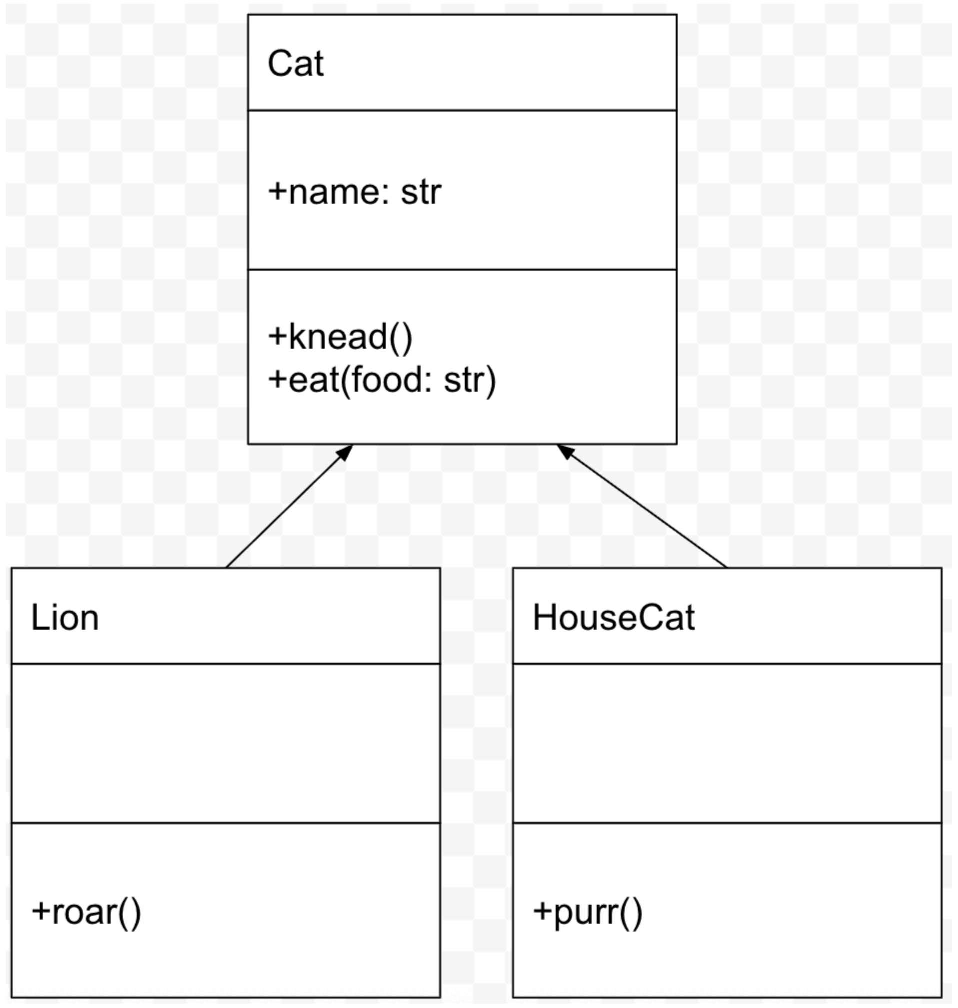
This UML diagram says:

- class name: `Cat`
- `str` attribute called `name`
- method called `knead()`
- method called `eat(food: str)`

`+` : intended to be publicly available
`-` : not public (two underscores `__`)

UML Diagram: Subclass / Superclass Relationship

Arrow from subclass to superclass:



Poll: Which statement is TRUE?

```
        Vehicle
           △
           |
      _____|_____
      |         |
    Car      Truck
```

1. `Vehicle` inherits from both `Car` and `Truck`
2. `Car` and `Truck` are superclasses of `Vehicle`
3. `Car` and `Truck` are subclasses of `Vehicle`
4. `Car` and `Truck` are sibling classes with no relationship

Poll: If `Account` has a method `deposit()`, what can you infer about `SavingsAccount`?

```
    Account
       △
       |
SavingsAccount
```

1. SavingsAccount must override the deposit() method
2. SavingsAccount inherits the deposit() method
3. SavingsAccount cannot use the deposit() method
4. SavingsAccount must implement deposit() from scratch

(Privacy) Polls: Algorithmic hiring

“Shamazon” (a fictitious company) is looking to hire software engineers, and you have been tasked with designing a tool to filter the submitted resumes and select the ideal candidates for hire.

Question	Answer
What type of information is shared?	
Who is the subject of the information?	
Who is the sender of the information?	
Who are the potential recipients of the information?	
What principles govern the collection and transmission of this information?	

Let's go through Practice Quiz 3!

Poll:

1. What is your main takeaway from today?
2. What would you like to revisit next time?