Name: Serrano, Mark Angelo Y.
Section: C203

7OOP1

Finals Task 3. Simple Polymorphism

Problem. Chirp and Tweet

Create a simple program to demonstrate basic polymorphism with bird sounds.

Class - Bird:

- Methods:
  - `def make_sound(self) -> None`: An abstract method that represents making a sound. It doesn't have a specific implementation in the base class `Bird`.

Class - Sparrow (extends Bird):

- Methods:
  - `def make_sound(self) -> None`: Overrides the `make_sound` method from the base class `Bird`. It prints the sound "Chirp Chirp" when called.

Class - Parrot (extends Bird):

- Methods:
  - `def make_sound(self) -> None`: Overrides the `make_sound` method from the base class `Bird`. It prints the sound "Tweet Tweet" when called.

Class - BirdCage:

- Methods:
  - `def make_bird_sounds(self, birds: List) -> None`: Accepts a list of `Bird` objects as input. Iterates through the list of birds and calls the `make_sound` method on each bird to make its sound.

Requirements

Test Cases

Test case 1

Should return ['Chirp Chirp'] when invoking the method [make_sound()] of Sparrow object returned when invoking the Sparrow() constructor of the Sparrow class.

Test case 2

Should return ['Tweet Tweet'] when invoking the method [make_sound()] of Parrot object returned when invoking the Parrot() constructor of the Parrot class.

Test case 3

Should return ['Chirp Chirp'] when invoking the method [make_sound()] of Bird object returned when invoking the Sparrow() constructor of the Sparrow class and return ['Tweet Tweet'] when invoking the method [make_sound()] of Bird object returned when invoking the Parrot() constructor of the Parrot class.

Test case 4

Should make Bird class an abstract.

Test case 5

Should return ['Chirp Chirp', 'Tweet Tweet'] when invoking the method [make_bird_sounds([Sparrow(), Parrot()])] of BirdCage object returned when invoking the BirdCage() constructor of the BirdCage class.

birds.py


birds.py
test_cases.py
from abc import ABC, abstractmethod
class Bird(ABC):
1 usage (1 dynamic)
@abstractmethod
def make_sound(self):
pass
3 usages
class Sparrow:
2 usages (1 dynamic)
def make_sound(self):
return "Chirp Chirp"
4 usages
class Parrot:
2 usages (1 dynamic)
def make_sound(self):
return "Tweet Tweet"
2 usages
class BirdCage:
1 usage
def make_bird_sounds(self, birds: list):
for bird in birds:
print(bird.make_sound())

test_cases.py

```python
from birds import Sparrow, Parrot, BirdCage


1 usage
def test_sparrow_sound():
    bird = Sparrow()
    print(bird.make_sound())
1 usage
def test_parrot_sound():
    bird2 = Parrot()
    print(bird2.make_sound())
1 usage
def test_bird_cage():
    bird1 = Sparrow()
    bird2 = Parrot()
    bird3 = Parrot()
    birds = [bird1, bird2, bird3]

    cage = BirdCage()
    cage.make_bird_sounds(birds)


1 usage
def main():
    test_sparrow_sound()
    print("===")
    test_parrot_sound()
    print("===")
    test_bird_cage()


if __name__ == '__main__':
    main()
```