

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.

Note:

- *The test cases are not outputs of your main file but of a hidden test file. Create and implement the classes instructed to test your code.*
- *Each class should be defined in its own file, with the file name following camelCase conventions (e.g., `bankAccount.py`).*

TEST CASES:

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

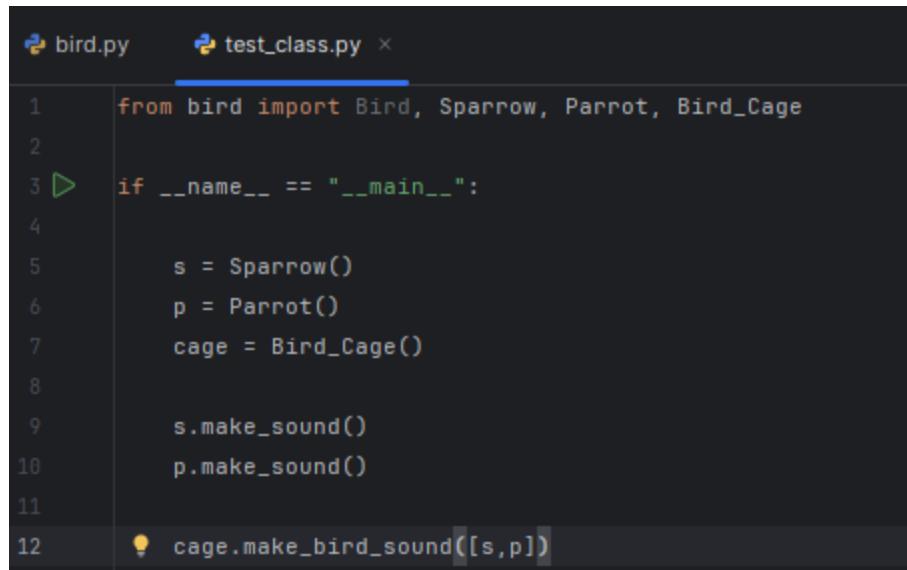
Should return ['Chirp Chirp', 'Tweet Tweet'] when invoking the method [make_bird_sounds([Sparrow(), Parrot()])] of BirdCage object returned when invoking the BirdCage() constructor of the BirdCage class.

CODE

```
bird.py ×

1  from abc import ABC, abstractmethod
2  @
3  class Bird(object): 5 usages
4  @
5      @abstractmethod 1 usage
6      def make_sound(self) -> None:
7          pass
8
9
10
11
12
13
14
15
16
17
18
```

The screenshot shows a code editor window with a dark theme. The file is named 'bird.py'. The code defines a Bird class with an abstract method 'make_sound'. It then defines two subclasses: Sparrow and Parrot, each overriding the 'make_sound' method to print specific chirps. Finally, it defines a Bird_Cage class with a 'make_bird_sound' method that iterates over a list of birds and calls their 'make_sound' methods. The code editor highlights syntax in green (strings), blue (keywords), and orange (comments). Line numbers are visible on the left, and there are small circular icons with arrows pointing up or down at various points in the code.



```
bird.py      test_class.py x
1  from bird import Bird, Sparrow, Parrot, Bird_Cage
2
3 D  if __name__ == "__main__":
4
5      s = Sparrow()
6      p = Parrot()
7      cage = Bird_Cage()
8
9      s.make_sound()
10     p.make_sound()
11
12     cage.make_bird_sound([s,p])
```

SAMPLE OUTPUT

```
Chirp Chirp
Tweet Tweet
Chirp Chirp
Tweet Tweet
```