Johnest Wheyne D. Ogania
BSCS-C204

7OOP
11-19-2025

Final Task 2
Inheritance

Problem:

Finals Task 2. Inheritance

Problem School Performance

Note: You are to create 4 separate python files for this task:

- **performer.py(base class)**
- **singer.py(sub class)**
- **dancer.py(sub class)**
- **test_class.py – following the required test cases**

In a school musical performance, different types of performers participate. For this program, we will be implementing the performers.

Base Class - Performer:

- Properties:
  - `name` (type: str): Represents the name of the performer.
  - `age` (type: int): Represents the age of the performer.
- Constructor:
  - `__init__(self, name: str, age: int)`: Initializes the `name` and `age` properties.
- Getters
  - `get_name(self) -> str`: Returns the name
  - `get_age(self) -> int`: Returns the age

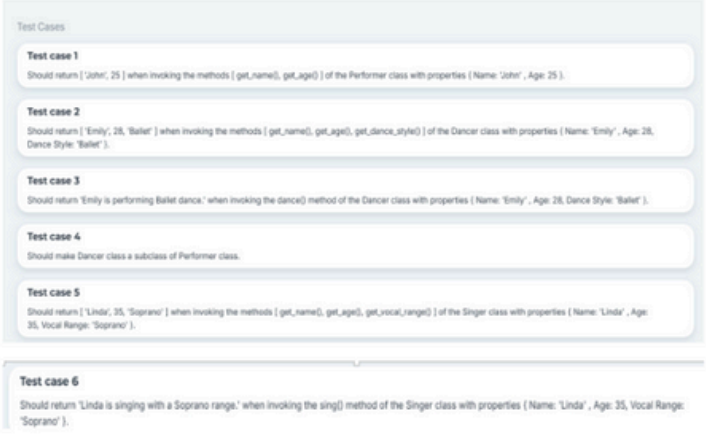
Subclass - Singer:

- Inherits From: `Performer`
- Additional Property:
  - `vocal range` (type: str): Represents the vocal range of the singer.
- Constructor:
  - `__init__(self, name: str, age: int, vocal_range: str)`: Initializes the `name` and `age` properties by calling the parent class's constructor and sets the `vocal_range` property.
- Getter:
  - `get_vocal_range(self) -> str`: Returns the vocal range of the singer.
- Method:
  - `sing(self) -> None`: Prints "{name} is singing with a {vocal_range} range."

Subclass - Dancer:

- Inherits From: `Performer`
- Additional Property:
  - `dance_style` (type: str): Represents the dance style of the dancer.
- Constructor:
  - `__init__(self, name: str, age: int, dance_style: str)`: Initializes the `name` and `age` properties by calling the parent class's constructor and sets the `dance_style` property.
- Getter:
  - `get_dance_style(self) -> str`: Returns the dance style of the dancer.
- Method:
  - `dance(self) -> None`: Prints "{name} is performing {dance_style} dance."

Sample output for the Test Class



Code:

performer.py

```python
class Performer:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

    def get_name(self) -> str:
        return self.name

    def get_age(self) -> int:
        return self.age
```

singer.py

```python
from performer import Performer

class Singer(Performer):
    def __init__(self, name: str, age: int, vocal_range: str):
        super().__init__(name, age)
        self.vocal_range = vocal_range

    def get_vocal_range(self) -> str:
        return self.vocal_range

    def sing(self) -> None:
        print(f"{self.name} is singing with a {self.vocal_range} range.")
```

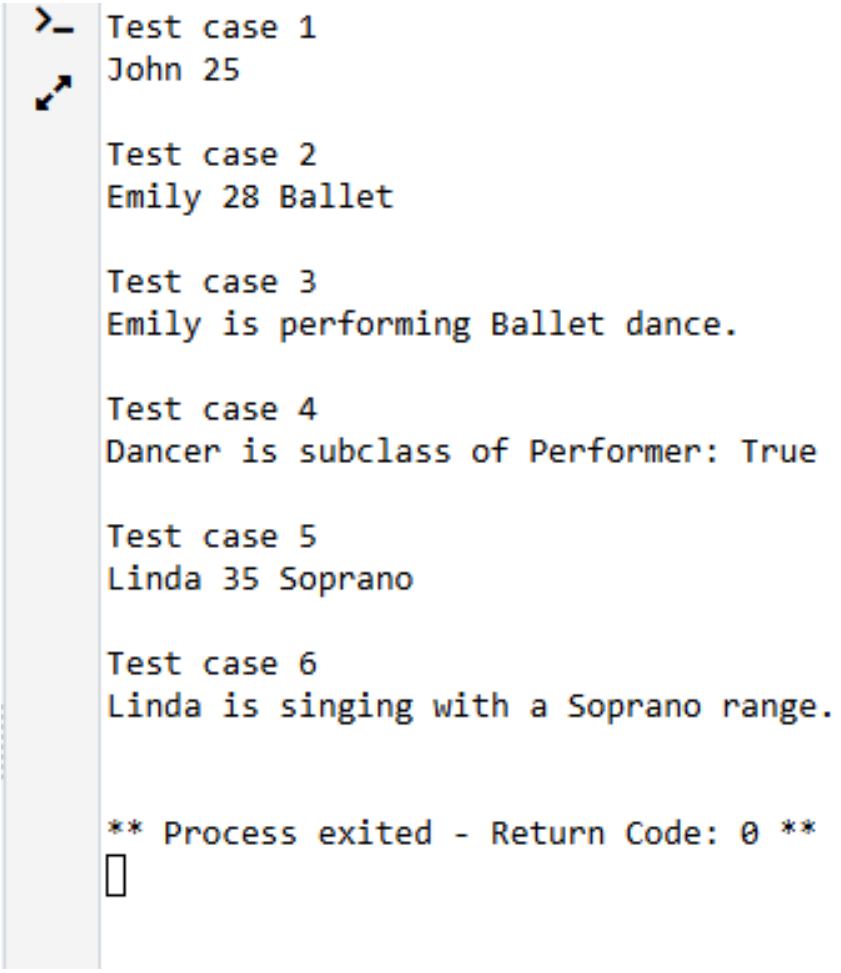
dancer.py

```python
from performer import Performer

class Dancer(Performer):
    def __init__(self, name: str, age: int, dance_style: str):
        super().__init__(name, age)
        self.dance_style = dance_style

    def get_dance_style(self) -> str:
        return self.dance_style

    def dance(self) -> None:
        print(f"{self.name} is performing {self.dance_style} dance.")
```

performer.py | singer.py | dancer.py | test_class.py | +

```python
import unittest
from performer import Performer
from dancer import Dancer
from singer import Singer

class TestPerformerClasses(unittest.TestCase):

    def test_case1(self):
        print("Test case 1")
        p = Performer("John", 25)
        print(p.get_name(), p.get_age())
        self.assertEqual(p.get_name(), "John")
        self.assertEqual(p.get_age(), 25)

    def test_case2(self):
        print("\nTest case 2")
        d = Dancer("Emily", 28, "Ballet")
        print(d.get_name(), d.get_age(), d.get_dance_style())
        self.assertEqual(d.get_name(), "Emily")
        self.assertEqual(d.get_age(), 28)
        self.assertEqual(d.get_dance_style(), "Ballet")

    def test_case3(self):
        print("\nTest case 3")
        d = Dancer("Emily", 28, "Ballet")
        result = d.dance()   # Call ONCE
        self.assertIsNone(result)

    def test_case4(self):
        print("\nTest case 4")
        d = Dancer("Test", 20, "HipHop")
        print("Dancer is subclass of Performer:", isinstance(d, Performer))
        self.assertTrue(isinstance(d, Performer))

    def test_case5(self):
        print("\nTest case 5")
        s = Singer("Linda", 35, "Soprano")
        print(s.get_name(), s.get_age(), s.get_vocal_range())
        self.assertEqual(s.get_name(), "Linda")
        self.assertEqual(s.get_age(), 35)
        self.assertEqual(s.get_vocal_range(), "Soprano")

    def test_case6(self):
        print("\nTest case 6")
        s = Singer("Linda", 35, "Soprano")
        result = s.sing()    # Call ONCE
        self.assertIsNone(result)

if __name__ == "__main__":
    unittest.main()
```

Output:


Test case 1
John 25
Test case 2
Emily 28 Ballet
Test case 3
Emily is performing Ballet dance.
Test case 4
Dancer is subclass of Performer: True
Test case 5
Linda 35 Soprano
Test case 6
Linda is singing with a Soprano range.
** Process exited - Return Code: 0 **