Mechure, Liann S. 7OOP1 BSCS - C204

Finals Task 2. Inheritance

Sample 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

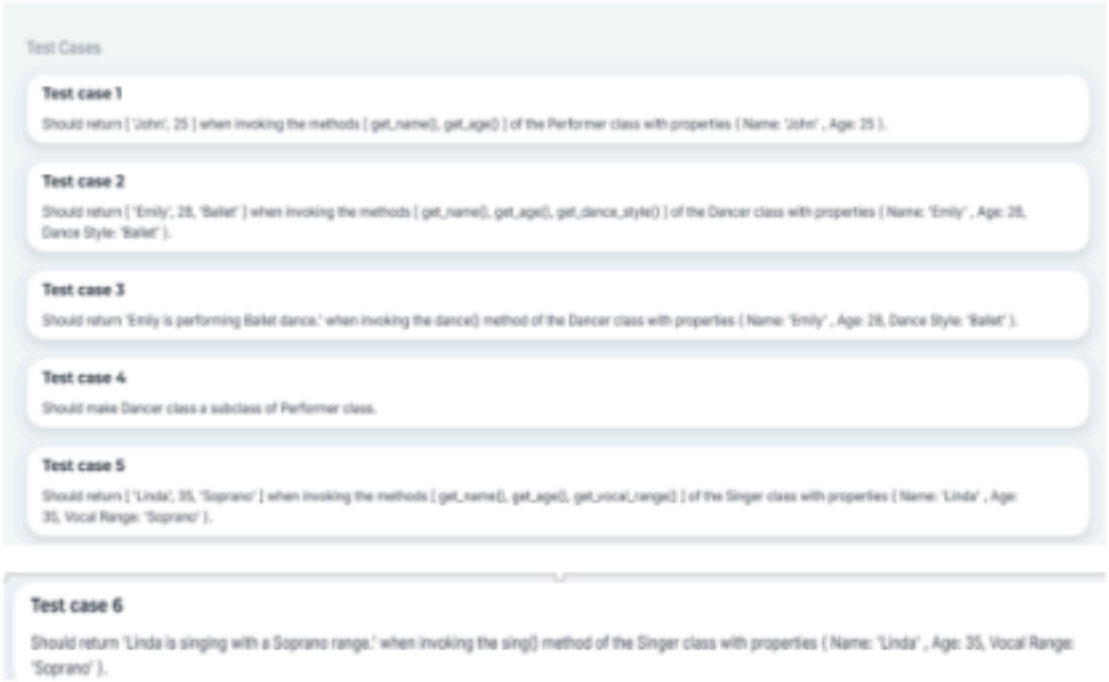
Subclass - Singer:

- 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



Source 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

    def introduce(self) -> None:
        print(f"Hello! My name is {self.name} and I am {self.age} years old.")

```

singer.py

```
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.get_name()} is singing with a {self.vocal_range}
range.")
```

dancer.py

```
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.get_name()} is dancing {self.dance_style}.")

```

test_class.py

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

# Test Case 1
performer = Performer("John", 25)
print(performer.get_name(), performer.get_age())

# Test Case 2
dancer = Dancer("Emily", 28, "Ballet")
print(dancer.get_name(), dancer.get_age(), dancer.get_dance_style())

# Test Case 3
dancer.dance()

# Test Case 4
print(isinstance(dancer, Performer))

# Test Case 5
singer = Singer("Linda", 35, "Soprano")
print(singer.get_name(), singer.get_age(), singer.get_vocal_range())

# Test Case 6
singer.sing()
```

Sample Output

```
John 25
Emily 28 Ballet
Emily is dancing Ballet.
True
Linda 35 Soprano
Linda is singing with a Soprano range.
PS C:\Users\Liann>
```