สมาชิก

1.ชลพัฒน์ ปิ่นมุนี 653380126-0 sec.1
2. ชนินทร์ ธัญสิริพัฒนธาดา 653380125-2 sec.1
3. ปวีณวัฒน์ สุขร่วม 653380136-7 Sec.1

ผลลัพธ์

Flyweight

Round 1

Prompt

Use flyweight pattern to write code in python and write pytest to test given code that have 100% statement coverage.

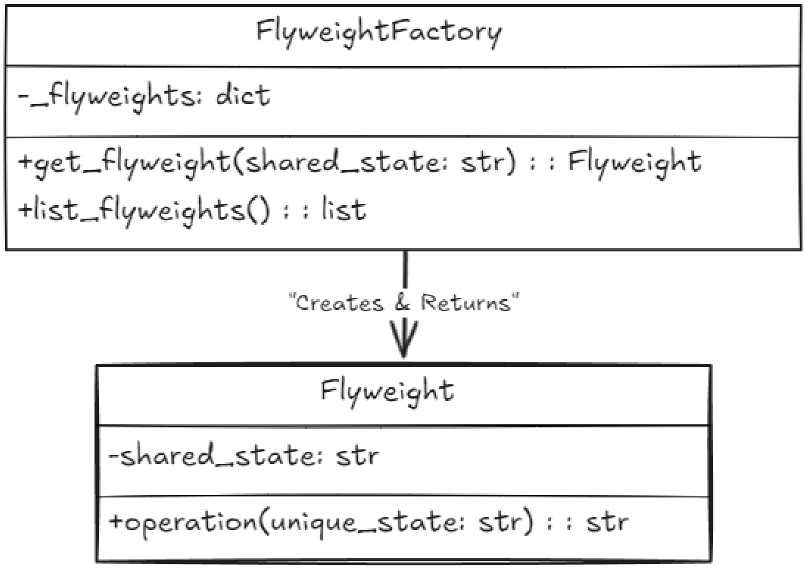
Source Code

```
class Flyweight:
    def __init__(self, shared_state):
        self.shared_state = shared_state

    def operation(self, unique_state):
        return f"Flyweight with shared ({self.shared_state}) and unique 
({unique_state}) states."


class FlyweightFactory:
    _flyweights = {}

    @staticmethod
    def get_flyweight(shared_state):
        key = str(shared_state)
        if key not in FlyweightFactory._flyweights:
            FlyweightFactory._flyweights[key] = Flyweight(shared_state)
        return FlyweightFactory._flyweights[key]

    @staticmethod
    def list_flyweights():
        return list(FlyweightFactory._flyweights.keys())
```

Test Code

```
import pytest
from code import FlyweightFactory

def test_flyweight_creation():
    factory = FlyweightFactory()

    # Create flyweights
    flyweight1 = factory.get_flyweight("shared_state_1")
    flyweight2 = factory.get_flyweight("shared_state_1")
    flyweight3 = factory.get_flyweight("shared_state_2")

    # Check if flyweight instances with the same shared state are the same
    assert flyweight1 is flyweight2, "Flyweights with the same shared state should be the same instance."
    assert flyweight1 is not flyweight3, "Flyweights with different shared states should be different instances."

    # Test the operation method
    assert flyweight1.operation("unique_state_1") == "Flyweight with shared (shared_state_1) and unique (unique_state_1) states."
    assert flyweight3.operation("unique_state_3") == "Flyweight with shared
(shared_state_2) and unique (unique_state_3) states."

def test_flyweight_list():
    factory = FlyweightFactory()

    # Create flyweights
    factory.get_flyweight("shared_state_1")
    factory.get_flyweight("shared_state_2")

    # List flyweights
    flyweights = factory.list_flyweights()

    # Check if flyweights are listed correctly
    assert "shared_state_1" in flyweights, "shared_state_1 should be listed."
    assert "shared_state_2" in flyweights, "shared_state_2 should be listed."
    assert len(flyweights) == 2, "There should be exactly two flyweights
listed."

if _name_ == "_main_":
    pytest.main()
```

Diagram


FlyweightFactory
-_flyweights: dict
+get_flyweight(shared_state: str) : : Flyweight
+list_flyweights() : : list
"Creates & Returns"
Flyweight
-shared_state: str
+operation(unique_state: str) : : str

a. โค้ดที่สร้างขึ้นถูกต้องตามข้อกำหนดที่ระบุไว้หรือไม่
ถูกต้องตาม Prompt ที่กำหนดไว้

b. โค้ดที่ถูกสร้างขึ้นใช้ภาษาโปรแกรมและ Design pattern ที่ระบุไว้หรือไม่
โค้ดถูกสร้างขึ้นโดยใช้ภาษา Python และถูกต้องตาม Flyweight pattern ที่กำหนด

Round 2

Prompt

Write code Use the Flyweight pattern in Python And Write pytest to test that code have 100% statement coverage

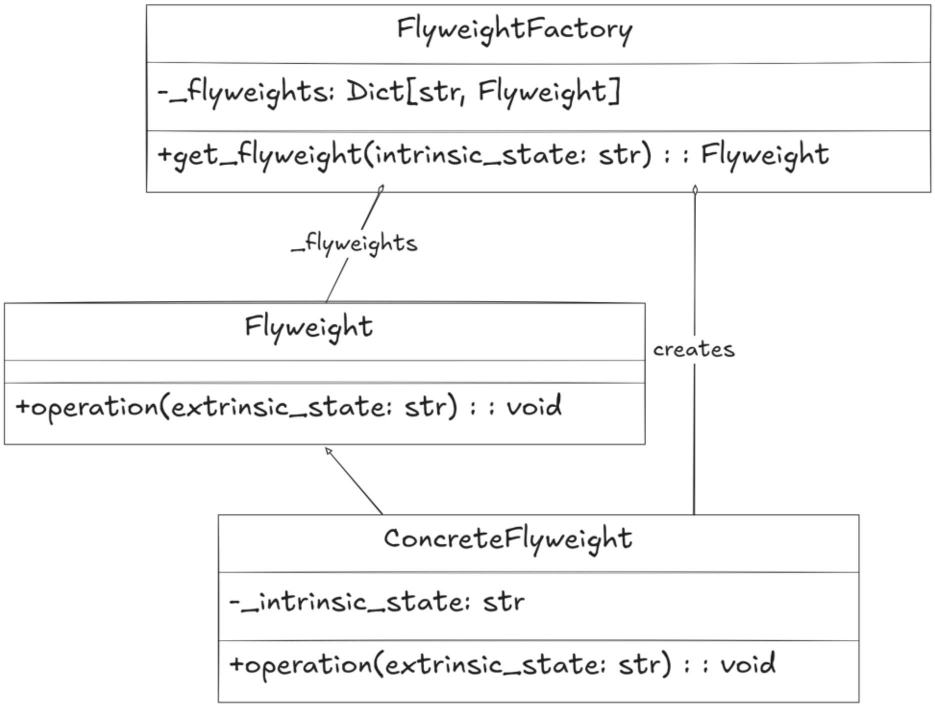
Source Code

```python
# flyweight.py

from typing import Dict

class Flyweight:
    def operation(self, extrinsic_state: str) -> None:
        raise NotImplementedError("Subclasses should implement this method")

class ConcreteFlyweight(Flyweight):
    def __init__(self, intrinsic_state: str) -> None:
        self._intrinsic_state = intrinsic_state

    def operation(self, extrinsic_state: str) -> None:
        print(f"ConcreteFlyweight: Intrinsic state = {self._intrinsic_state}, Extrinsic state = {extrinsic_state}")

class FlyweightFactory:
    def __init__(self) -> None:
        self._flyweights: Dict[str, Flyweight] = {}

    def get_flyweight(self, intrinsic_state: str) -> Flyweight:
        if intrinsic_state not in self._flyweights:
            self._flyweights[intrinsic_state] = ConcreteFlyweight(intrinsic_state)
        return self._flyweights[intrinsic_state]
```

Test Code

```
# test_flyweight.py

import pytest
from flyweight import FlyweightFactory, ConcreteFlyweight

def test_flyweight_factory():
    factory = FlyweightFactory()

    # Test that a new Flyweight is created and reused correctly
    flyweight1 = factory.get_flyweight("state1")
    flyweight2 = factory.get_flyweight("state1")
    assert flyweight1 is flyweight2  # Should be the same object

    flyweight3 = factory.get_flyweight("state2")
    assert flyweight1 is not flyweight3  # Should be different objects

def test_concrete_flyweight_operation(capfd):
    factory = FlyweightFactory()
    flyweight = factory.get_flyweight("state1")

    flyweight.operation("extrinsic1")

    captured = capfd.readouterr()
    assert "ConcreteFlyweight: Intrinsic state = state1, Extrinsic state = extrinsic1" in captured.out

def test_flyweight_creation():
    factory = FlyweightFactory()
    # Test creation of ConcreteFlyweight instances
    flyweight1 = factory.get_flyweight("state1")
    assert isinstance(flyweight1, ConcreteFlyweight)

    flyweight2 = factory.get_flyweight("state2")
    assert isinstance(flyweight2, ConcreteFlyweight)

    assert flyweight1 is not flyweight2  # Different states should have different instances

if _name_ == "_main_":
    pytest.main()
```

Diagram


FlyweightFactory
-_flyweights: Dict[str, Flyweight]
+get_flyweight(intrinsic_state: str) : : Flyweight
_flyweights
creates
Flyweight
+operation(extrinsic_state: str) : : void
ConcreteFlyweight
-_intrinsic_state: str
+operation(extrinsic_state: str) : : void

a. โค้ดที่สร้างขึ้นถูกต้องตามข้อกำหนดที่ระบุไว้หรือไม่
ตรงตามข้อกำหนดที่ให้ใน prompt ทั้งตัวโปรแกรมและ ตัวทดสอบ

b. โค้ดที่ถูกสร้างขึ้นใช้ภาษาโปรแกรมและ Design pattern ที่ระบุไว้หรือไม่
ใช้ภาษา Python และ Design pattern คือ Flyweight pattern ตามที่กำหนด

Round 3

Prompt

Write python code use Flyweight pattern and write pytest to test that given code have 100% branch coverage

Source Code: flyweight.py

```
from typing import Dict

class Flyweight:
    def __init__(self, shared_state: str):
        self.shared_state = shared_state

    def operation(self, unique_state: str) -> str:
        return f"Flyweight: Displaying shared ({self.shared_state}) and unique ({unique_state}) state."


class FlyweightFactory:
    def __init__(self):
        self._flyweights: Dict[str, Flyweight] = {}

    def get_flyweight(self, shared_state: str) -> Flyweight:
        if shared_state not in self._flyweights:
            self._flyweights[shared_state] = Flyweight(shared_state)
        return self._flyweights[shared_state]

    def list_flyweights(self) -> Dict[str, Flyweight]:
        return self._flyweights


# Example usage:
```

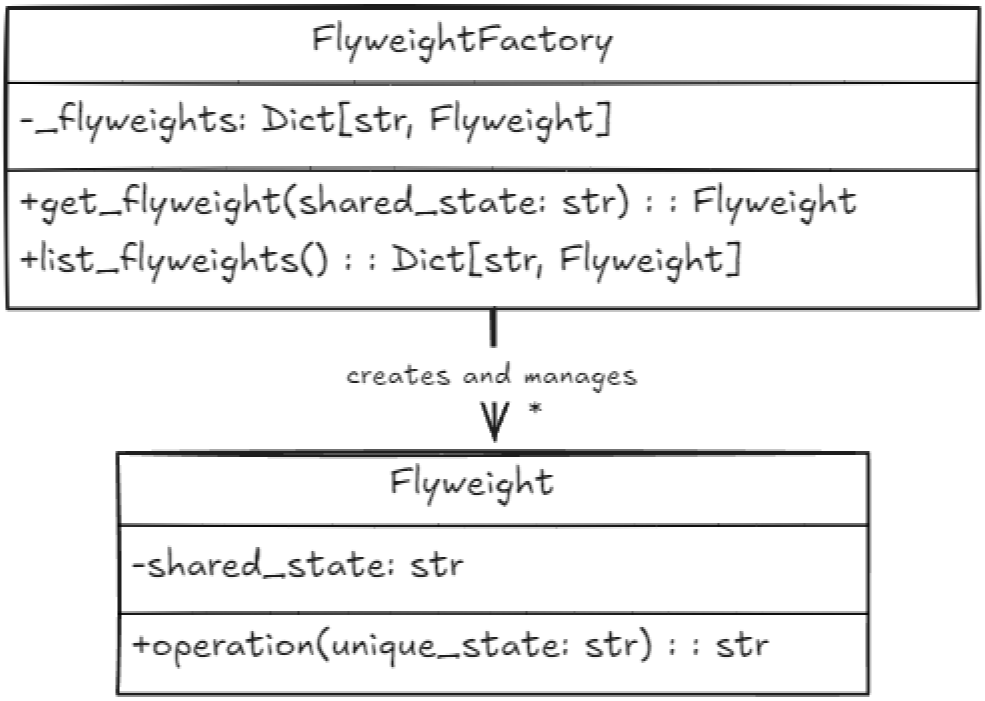
```
# factory = FlyweightFactory()
# flyweight1 = factory.get_flyweight("shared_state_1")
# flyweight2 = factory.get_flyweight("shared_state_1")
# assert flyweight1 is flyweight2
# print(flyweight1.operation("unique_state_1"))
```

Test Code

- import pytest: branch.py

```
from flyweight import Flyweight, FlyweightFactory

def test_flyweight_operation():
    flyweight = Flyweight("shared_state")
    result = flyweight.operation("unique_state")
    assert result == "Flyweight: Displaying shared (shared_state) and unique (unique_state) state."

def test_flyweight_factory_creates_new_flyweight():
    factory = FlyweightFactory()
    flyweight = factory.get_flyweight("shared_state_1")
    assert flyweight.shared_state == "shared_state_1"

def test_flyweight_factory_reuses_existing_flyweight():
    factory = FlyweightFactory()
    flyweight1 = factory.get_flyweight("shared_state_1")
    flyweight2 = factory.get_flyweight("shared_state_1")
    assert flyweight1 is flyweight2

def test_flyweight_factory_lists_flyweights():
    factory = FlyweightFactory()
    factory.get_flyweight("shared_state_1")
    factory.get_flyweight("shared_state_2")
    flyweights = factory.list_flyweights()
    assert len(flyweights) == 2
    assert "shared_state_1" in flyweights
    assert "shared_state_2" in flyweights
```

Diagram


FlyweightFactory
-_flyweights: Dict[str, Flyweight]
+get_flyweight(shared_state: str) : : Flyweight
+list_flyweights() : : Dict[str, Flyweight]
creates and manages
*
Flyweight
-shared_state: str
+operation(unique_state: str) : : str

a. โค้ดที่สร้างขึ้นถูกต้องตามข้อกำหนดที่ระบุไว้หรือไม่
  - ถูกต้อง

b. โค้ดที่ถูกสร้างขึ้นใช้ภาษาโปรแกรมและ Design pattern ที่ระบุไว้หรือไม่
  - ใช้ภาษา Python และ Flyweight pattern ที่ระบุไว้

Intepreter

Round 1

Prompt

Use interpreter pattern to write code in python and write pytest to test given code that have 100% statement coverage.

Source Code

```
class Expression:
    def interpret(self, context):
        pass


class NumberExpression(Expression):
    def __init__(self, number):
        self.number = number

    def interpret(self, context):
        return self.number


class AddExpression(Expression):
    def __init__(self, left_expr, right_expr):
        self.left_expr = left_expr
        self.right_expr = right_expr

    def interpret(self, context):
        return self.left_expr.interpret(context) +
self.right_expr.interpret(context)
```

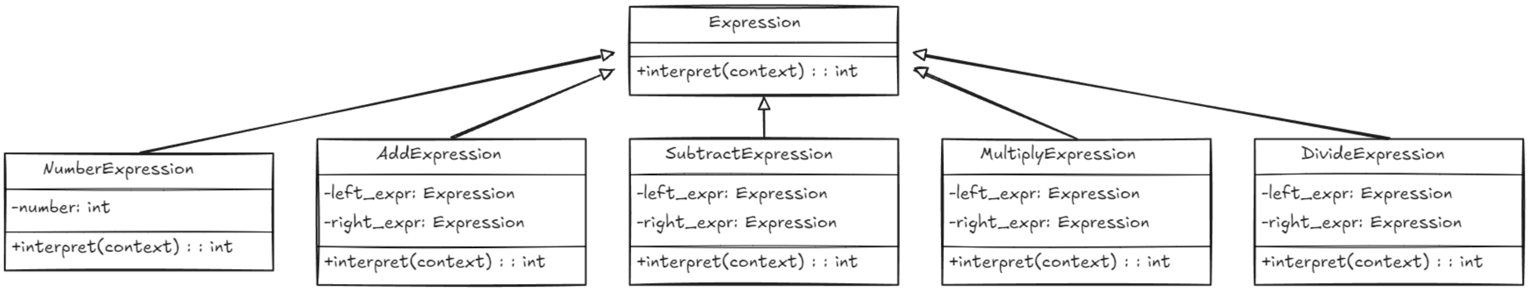
```
class SubtractExpression(Expression):
    def __init__(self, left_expr, right_expr):
        self.left_expr = left_expr
        self.right_expr = right_expr

    def interpret(self, context):
        return self.left_expr.interpret(context) -
self.right_expr.interpret(context)


class MultiplyExpression(Expression):
    def __init__(self, left_expr, right_expr):
        self.left_expr = left_expr
        self.right_expr = right_expr

    def interpret(self, context):
        return self.left_expr.interpret(context) *
self.right_expr.interpret(context)


class DivideExpression(Expression):
    def __init__(self, left_expr, right_expr):
        self.left_expr = left_expr
        self.right_expr = right_expr

    def interpret(self, context):
        denominator = self.right_expr.interpret(context)
        if denominator == 0:
            raise ValueError("Cannot divide by zero")
        return self.left_expr.interpret(context) / denominator
```

Test Code

```
import pytest
from code import Expression, NumberExpression, AddExpression, SubtractExpression, MultiplyExpression, DivideExpression

def test_number_expression():
    number = NumberExpression(5)
    assert number.interpret({}) == 5, "NumberExpression should return the number it holds."

def test_add_expression():
    expr = AddExpression(NumberExpression(5), NumberExpression(3))
    assert expr.interpret({}) == 8, "AddExpression should correctly add two numbers."

def test_subtract_expression():
    expr = SubtractExpression(NumberExpression(10), NumberExpression(4))
    assert expr.interpret({}) == 6, "SubtractExpression should correctly subtract the second number from the first."

def test_multiply_expression():
    expr = MultiplyExpression(NumberExpression(7), NumberExpression(6))
    assert expr.interpret({}) == 42, "MultiplyExpression should correctly multiply two numbers."

def test_divide_expression():
    expr = DivideExpression(NumberExpression(8), NumberExpression(2))
    assert expr.interpret({}) == 4.0, "DivideExpression should correctly divide the first number by the second."
```

```
def test_divide_by_zero():
    expr = DivideExpression(NumberExpression(8), NumberExpression(0))
    with pytest.raises(ValueError, match="Cannot divide by zero"):
        expr.interpret({})

if _name_ == "_main_":
    pytest.main()
```

Diagram



a. โค้ดที่สร้างขึ้นถูกต้องตามข้อกำหนดที่ระบุไว้หรือไม่
โค้ดที่สร้างขึ้นถูกต้องตาม Prompt ที่กำหนดไว้

b. โค้ดที่ถูกสร้างขึ้นใช้ภาษาโปรแกรมและ Design pattern ที่ระบุไว้หรือไม่
โค้ดถูกสร้างขึ้นโดยใช้ภาษา Python และถูกต้องตาม Flyweight pattern ที่กำหนด

Round 2

Prompt

Write code Use the Interpreter pattern in Python And Write pytest to test that code have 100% statement coverage

Source Code

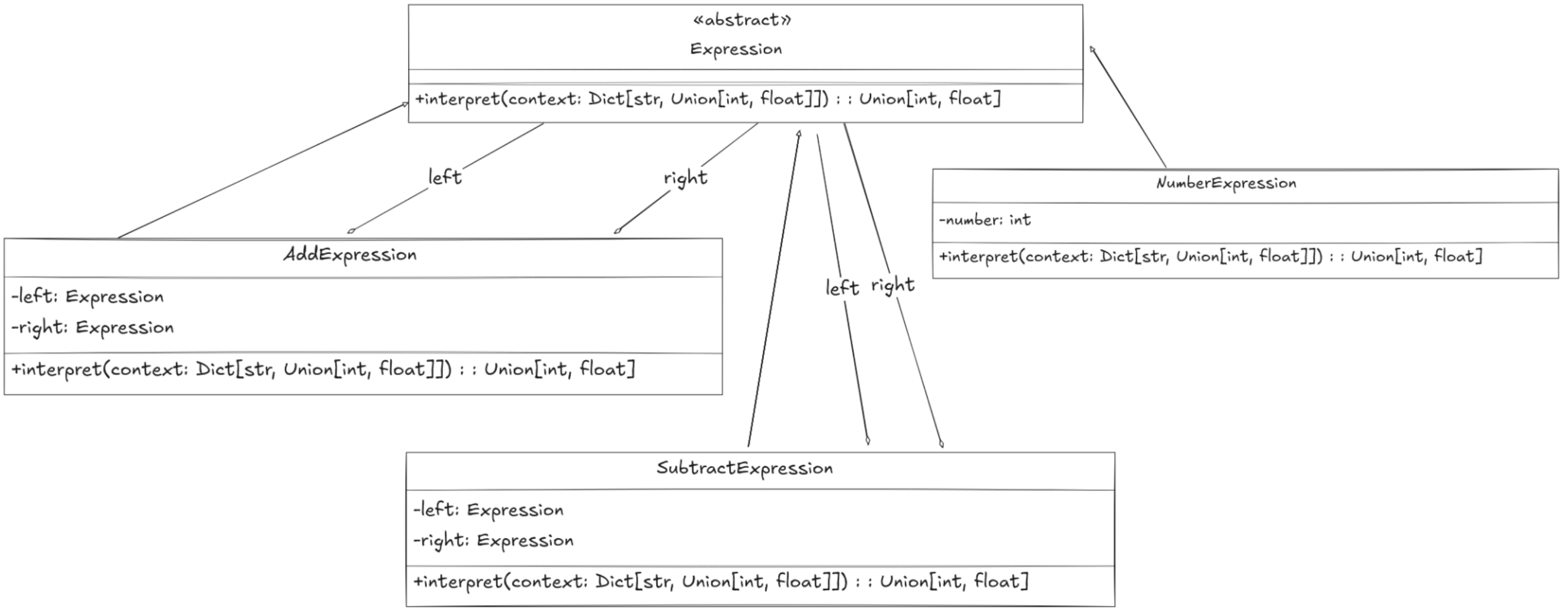
```
# interpreter.py

from abc import ABC, abstractmethod
from typing import Dict, Union

class Expression(ABC):
    @abstractmethod
    def interpret(self, context: Dict[str, Union[int, float]]) -> Union[int, float]:
        pass

class NumberExpression(Expression):
    def _init_(self, number: int) -> None:
        self.number = number

    def interpret(self, context: Dict[str, Union[int, float]]) -> Union[int, float]:
        return self.number

class AddExpression(Expression):
    def _init_(self, left: Expression, right: Expression) -> None:
        self.left = left
        self.right = right

    def interpret(self, context: Dict[str, Union[int, float]]) -> Union[int, float]:
        return self.left.interpret(context) + self.right.interpret(context)
```

```
class SubtractExpression(Expression):
    def _init_(self, left: Expression, right: Expression) -> None:
        self.left = left
        self.right = right

    def interpret(self, context: Dict[str, Union[int, float]]) -> Union[int, float]:
        return self.left.interpret(context) - self.right.interpret(context)
```

Test Code

```
# test_interpreter.py

import pytest
from interpreter import NumberExpression, AddExpression,
SubtractExpression

def test_number_expression():
    expr = NumberExpression(5)
    context = {}
    assert expr.interpret(context) == 5

def test_add_expression():
    left = NumberExpression(5)
    right = NumberExpression(3)
    expr = AddExpression(left, right)
    context = {}
    assert expr.interpret(context) == 8

def test_subtract_expression():
    left = NumberExpression(5)
    right = NumberExpression(3)
    expr = SubtractExpression(left, right)
    context = {}
    assert expr.interpret(context) == 2

def test_combined_expression():
    # (5 + 3) - (2 + 1)
    expr = SubtractExpression(
        AddExpression(NumberExpression(5), NumberExpression(3)),
        AddExpression(NumberExpression(2), NumberExpression(1))
    )
    context = {}
    assert expr.interpret(context) == 5

if _name_ == "_main_":
    pytest.main()
```

Diagram


«abstract»
Expression
+interpret(context: Dict[str, Union[int, float]]) : : Union[int, float]
left
right
NumberExpression
-number: int
+interpret(context: Dict[str, Union[int, float]]) : : Union[int, float]
AddExpression
-left: Expression
-right: Expression
+interpret(context: Dict[str, Union[int, float]]) : : Union[int, float]
left
right
SubtractExpression
-left: Expression
-right: Expression
+interpret(context: Dict[str, Union[int, float]]) : : Union[int, float]

a. โค้ดที่สร้างขึ้นถูกต้องตามข้อกำหนดที่ระบุไว้หรือไม่
ตรงตามข้อกำหนดที่ให้ใน prompt ทั้งตัวโปรแกรมและ ตัวทดสอบ

b. โค้ดที่ถูกสร้างขึ้นใช้ภาษาโปรแกรมและ Design pattern ที่ระบุไว้หรือไม่
ใช้ภาษา Python และ Design pattern คือ Interpreter pattern ตามที่กำหนด

Round 3

Prompt

Write python code use interpreter pattern and write pytest to test that given code have 100% branch coverage

Source Code: interpreter.py

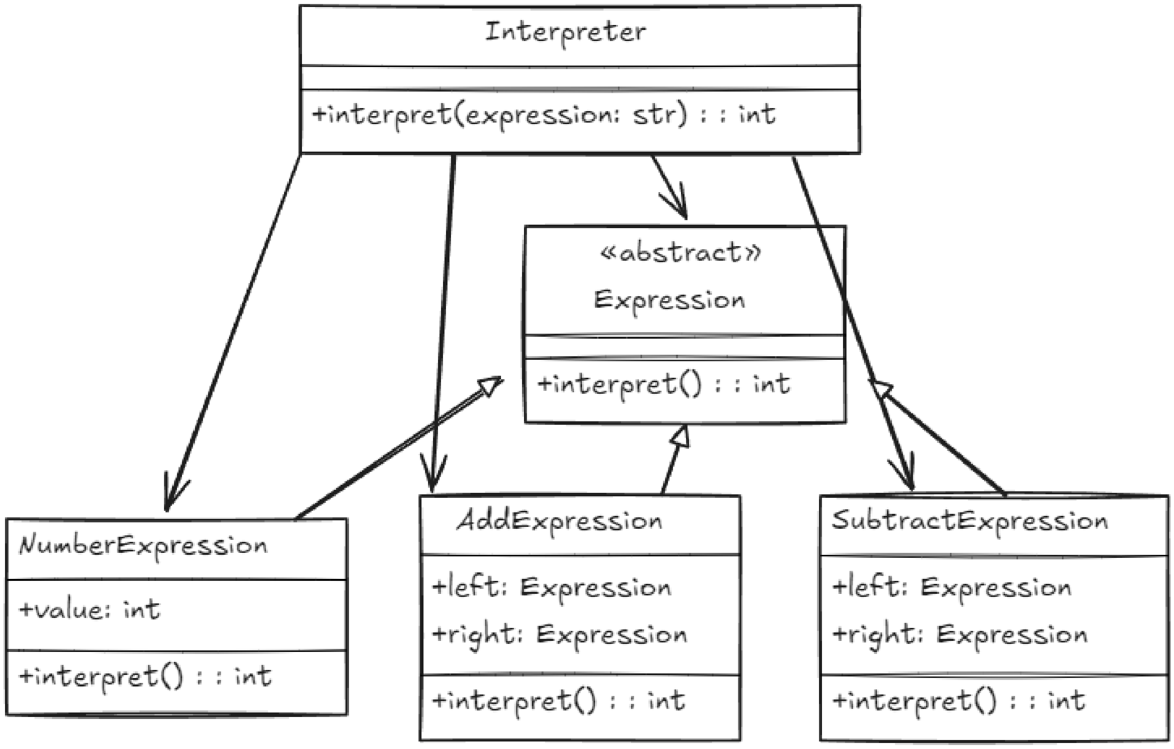
```
# interpreter.py

from abc import ABC, abstractmethod

# Abstract Expression
class Expression(ABC):
    @abstractmethod
    def interpret(self) -> int:
        pass

# Terminal Expressions
class NumberExpression(Expression):
    def _init_(self, number: int):
        self.number = number

    def interpret(self) -> int:
        return self.number

class AddExpression(Expression):
    def _init_(self, left: Expression, right: Expression):
        self.left = left
        self.right = right

    def interpret(self) -> int:
        return self.left.interpret() + self.right.interpret()
```

```
class SubtractExpression(Expression):
    def _init_(self, left: Expression, right: Expression):
        self.left = left
        self.right = right

    def interpret(self) -> int:
        return self.left.interpret() - self.right.interpret()

# Client
class Interpreter:
    def interpret(self, expression: str) -> int:
        tokens = expression.split()
        stack = []

        for token in tokens:
            if token.isdigit():
                stack.append(NumberExpression(int(token)))
            elif token == '+':
                right = stack.pop()
                left = stack.pop()
                stack.append(AddExpression(left, right))
            elif token == '-':
                right = stack.pop()
                left = stack.pop()
                stack.append(SubtractExpression(left, right))

        return stack.pop().interpret()
```

Test Code: test_interpreter.py

```
import pytest
from interpreter import Interpreter

def test_interpreter_addition():
    interpreter = Interpreter()
    result = interpreter.interpret("3 5 +")
    assert result == 8

def test_interpreter_subtraction():
    interpreter = Interpreter()
    result = interpreter.interpret("10 2 -")
    assert result == 8

def test_interpreter_complex_expression():
    interpreter = Interpreter()
    result = interpreter.interpret("10 5 + 3 -")
    assert result == 12

def test_interpreter_number_expression():
    interpreter = Interpreter()
    result = interpreter.interpret("7")
    assert result == 7

def test_interpreter_invalid_expression():
    interpreter = Interpreter()
    with pytest.raises(IndexError):
        interpreter.interpret("5 +")
```

Diagram



a. โค้ดที่สร้างขึ้นถูกต้องตามข้อกำหนดที่ระบุไว้หรือไม่
   - ถูกต้อง

b. โค้ดที่ถูกสร้างขึ้นใช้ภาษาโปรแกรมและ Design pattern ที่ระบุไว้หรือไม่
   - ใช้ภาษา Python และ Interpreter pattern ที่ระบุไว้